Centrum voor Wiskunde en Informatica
Centre for Mathematics and Computer Science

J. Heering

Eentalige programmeeromgevingen

Department of Computer Science Notitie CS-N8503 April

Monolingual programming environments

The Centre for Mathematics and Computer Science is a research institute of the Stichting Mathematisch Centrum, which was founded on February 11, 1946, as a nonprofit institution aiming at the promotion of mathematics, computer science, and their applications. It is sponsored by the Dutch Government through the Netherlands Organization for the Advancement of Pure Research (Z.W.O.).

EENTALIGE PROGRAMMEEROMGEVINGEN

Jan Heering
Centrum voor Wiskunde en Informatica
Amsterdam

De programmeur is ermee gediend als de verschillende componenten van een programmeeromgeving zoveel mogelijk dezelfde taal gebruiken. Integratie van bijvoorbeeld programmeertaal, commandotaal en taal van de symbolische debugger/tracer is in verregaande mate mogelijk en dit leidt tot een zeer homogene omgeving. Het ontwerp van een dergelijke eentalige omgeving wordt in grote lijnen besproken.

1983-84 CR Categories: D.2.6 [**Software Engineering**]: Programming Environments; D.4.7 [**Operating Systems**]: Organization and Design; D.3.3 [**Programming Languages**]: Language Constructs; D.4.9 [**Operating Systems**]: Systems Programs and Utilities; D.2.5 [**Software Engineering**]: Debugging aids.

1980 Mathematics Subject Classification: 68B20 [**Software**]: Supervisory systems.
6q D26, 6q D43, 6q D57, 6q D61, 6q D25

Inhoud

Notitie CS-N8503
Centrum voor Wiskunde en Informatica
Postbus 4079, 1009 AB Amsterdam

Inleiding

Er is reden om te geloven dat de complexiteit van de huidige bedrijfssystemen en programmeeromgevingen *veel hoger* is dan nodig

Hoe weet je dat?

Door een systeem aan te geven *dat hetzelfde doet maar veel eenvoudiger is* (maar misschien kan het nog wel veel eenvoudiger!)

Hoge complexiteit van huidige systemen gevolg van *overmatige modularisatie op het hoogste niveau*

Dit leidt tot

- Vele modes, vele talen
  - *Edittaal*
  - *Commandotaal*
  - *Programmeerta(a)l(en)*
  - *Taal van de symbolische debugger/tracer*
  - *Taal van de linkage editor*
- Heterogeen geheel
  - Gescheiden ontwikkeling van talen
  - Concurrentie tussen modes omdat bijbehorende talen te weinig verschillen
- Concurrentie tussen commandotalen en programmeertalen bijzonder sterk
- Op vele installaties wordt *helft van alle procedures in commandotaal geschreven*
- Programma's in commandotalen zijn echter
  - doorgaans onbegrijpelijk
  - volstrekt niet overdraagbaar naar andere systemen

Programmeertalen zijn afhankelijk van krachtige "buitenwereld" (=bedrijfssysteem) voor

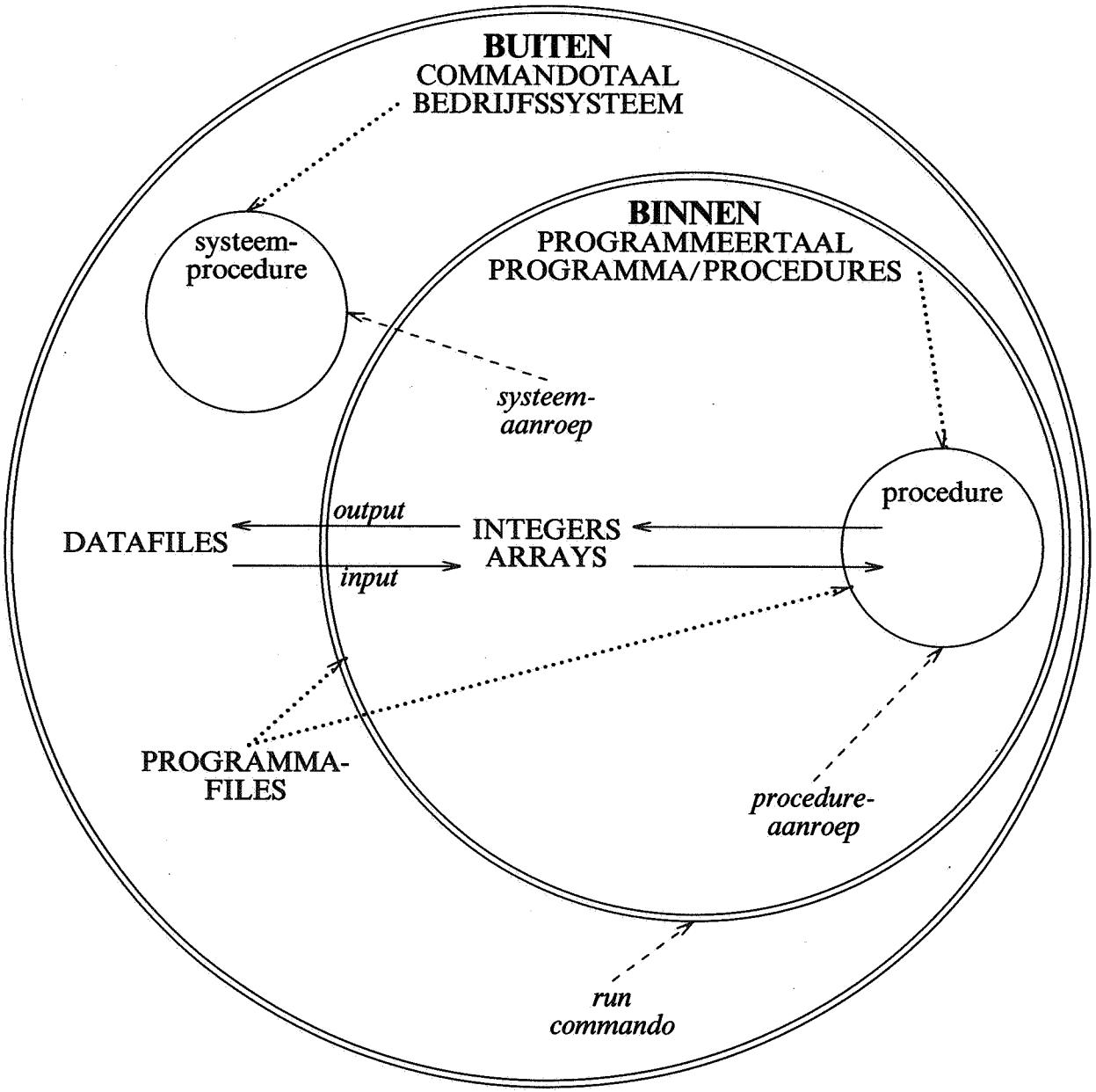
- beheer permanente gegevens (files)
- creatie nieuwe programma's en procedures

Uitzonderingen:

- SMALLTALK
- LISP
- PROLOG
- APL

Commandotalen zijn geheel of grotendeels self-supporting, d.w.z. niet afhankelijk van buitenwereld

Binnen- en buitenwereld van programma in conventionele omgeving



- Files zijn *permanente variabelen*, d.w.z. variabelen met onbeperkte levensduur, maar . . .
- Er is in bestaande systemen een *asymmetrie tussen files en programmavariabelen*
  - Types van files en programmavariabelen zijn drastisch verschillend
    Bijgevolg zijn er allerlei typeconversies en input/output operaties nodig
  - Structuur van de permanente omgeving (=file directory) verschilt totaal van structuur van de lokale omgeving
- Zelfde asymmetrie *ook in commandotalen*

Voorbeeld van asymmetrie tussen files en lokale variabelen in de UNIX "shell" commandotaal

Pseudo-Algol

```
permanent x ; local y
x := 'abc'
y := x
```

Shell equivalent (x is een file en y is een lokale shell variabele)

```
echo abc >x
y = `cat x`
```

Verwissel scope van x en y

Maakt in pseudo-Algol weinig uit

```
local x ; permanent y
x := 'abc'
y := x
```

maar shell programmaatje wordt heel anders

```
x = abc
echo $x >y
```

Integratie van commandotaal, programmeertaal en debug/tracetaal

- *Integratie van modes* in plaats van scheiding van modes
- Hoe ziet *geïntegreerde commando/programmeer/debugtaal* eruit?
- Integratie vindt plaats in twee fasen:
  - (1) Integratie van commandotaal en programmeertaal
  - (2) Integratie van commando/programmeertaal en debugtaal
- Integratie geeft scherpere randvoorwaarden voor taalontwerp

Eisen stellen aan taalconcepten om chaotische accumulatie van concepten te voorkomen:

(A) Elk taalconcept moet in alle drie modes zinvol zijn

(B) De semantiek van elk taalconcept moet mode-onafhankelijk zijn

(C) Gezamenlijk moeten de taalconcepten de gebruiker in elk van de drie modes voldoende faciliteiten bieden

Fase 1: Integratie van commandotaal en programmeertaal

- Elimineer verschillen tussen programma's en procedures
- Elimineer verschillen in type en naamgeving tussen files en lokale variabelen

Consequenties:

programma	=	procedure
run commando	=	procedure call
parameter-mechanisme voor programma's	=	parameter-mechanisme voor procedures
commandoniveau	=	interactief programmeren op hoogste procedurele niveau
file	=	permanente variabele
programmafile	=	permanente procedurevariabele

Datatypes in geïntegreerde omgeving

- "kleine" types
  (*integers, korte teksten, . . .*)
- "grote" types
  (*arrays, tabellen,*
  lange teksten, procedures, . . .)
- Veelheid van datatypes vereist dat gebruikers *zelf (abstracte) datatypes kunnen definiëren*
- Typesysteem moet "volledig" zijn, d.w.z. er zijn types
  - *procedure*
  - *type* (=het type van typedefinities!)
  - *bibliotheek/directory*

Voorbeeld

Bijhouden van telefoonboek in conventionele omgeving

Telefoonboek op file

(1) Activeer telefoonboek-editor

(2) Modificeer nummer

(3) Terug naar commando mode

In geïntegreerde omgeving kan dit met één commando

Telefoonboek is object van type *tabel* geassigneerd aan permanente variabele *telefoonboek*

$$telefoonboek[abonnee] := nieuwnummer$$

Type checking in conventionele omgeving

- Compiler controleert typeconsistentie van afzonderlijke programma's/procedures
- Linkage editor controleert of te linken programma's/procedures qua type verenigbaar zijn (maar deze controle wordt ook vaak achterwege gelaten!)
- Files worden dynamisch (d.w.z. run-time) gebonden zonder dat daarbij een typecontrole plaatsvindt
- Commandotaal is niet getypeerd

Type checking in geïntegreerde omgeving

- Types van permanente variabelen worden eveneens gecontroleerd
- Altijd typecontrole bij binding separate modules
- Lijst van inconsistente aanroepers na modificatie van procedure
- Lijst van inconsistente instanties na modificatie van typedefinitie

Regel: typeconflicten zo vroeg mogelijk melden

Fase 2: Integratie van commando/programmeertaal en debugtaal

- Geen verschil meer tussen incrementeel programmeren en interactief debuggen
- Debug-commando's niet te onderscheiden van "normale" commando's/statements
- Debug-hulpmiddelen voor programma's/procedures identiek aan die op commandoniveau

Debugging hulpmiddelen

Alle "gewone" taalconstructies zijn nuttig in debugging mode, maar er zijn bovendien een aantal speciale constructies vereist

- Met behulp van *event associations* kunnen tijdens programma-uitvoering met bepaalde gebeurtenissen inspecties of ingrepen geassocieerd worden ten behoeve van debugging zonder dat de programmeur van te voren precies hoeft uit te zoeken waar of wanneer deze gebeurtenissen optreden
- *Automatisch herstel van neveneffecten* maakt neveneffectvrije inspectie en look-ahead mogelijk

Event associations

Syntax

when <event>
  do
    <action>
  od

Voorbeelden

when *x* **is modified**
  do
    display *x*
  od

when *P* **is called**
  do
    interact
  od

Event asociations zijn niet alleen nuttig ten behoeve van debugging
Bijvoorbeeld: toepassing van event association om *een relatie invariant te houden* (vergelijk VISICALC)

1-1 relatie R

$R(x,y) \Leftrightarrow y = P(x) \Leftrightarrow x = Q(y)$

when x **is modified**
  do
    if $R(x,y)$ **fails**
      then $y := P(x)$
    fi
  od

when y **is modified**
  do
    if $R(x,y)$ **fails**
      then $x := Q(y)$
    fi
  od

Event associations zijn *gevaarlijk*
Kunnen op vrijwel elk moment getriggerd worden en dan ingrijpen in het verloop van de berekening
"Er staat niet wat er staat"
Dus zoveel mogelijk temmen:

- Neveneffecten van <event>-test worden ongedaan gemaakt
- Triggering van event associations veroorzaakt geen context switch

Automatisch herstel van neveneffecten ten behoeve van neveneffectvrije inspectie in debugging mode

Bijvoorbeeld

display probe *stack.pop* **endprobe**

display probe *text.nextline* **endprobe**

Eveneens nuttig in andere modes voor

- backtracking
- asserties (mogen geen neveneffecten hebben)
- ongedaan maken van commando's (*undo*)

Conclusies

Taalintegratie leidt *niet* tot monstrueuze opeenhoping van constructies ten gevolge van *conceptuele inefficiëntie/redundantie* van conventionele systemen

Eigenschappen eentalig systeem

- Geen verschil tussen programma's en procedures
- Files zijn permanente variabelen
  Geen expliciete input/output van/naar permanente variabelen
- Zelfde types op alle niveau's in het systeem
  Definieerbare datatypes
  System-wide type checking
- Procedures en typedefinities zijn zelf ook weer in de taal manipuleerbaar als objecten van respectievelijk type *procedure* en *type*
- Bibliotheken van procedures, typedefinities en andere permanente objecten zijn in de taal manipuleerbaar
- Event associations voor debugging/tracing zijn ook bruikbaar in andere modes als exception handlers, guards, etc.
- Automatisch herstel van neveneffecten ten behoeve van neveneffectvrije inspectie in debugging mode zijn ook nuttig in andere modes voor backtracking, asserties, *undo*, etc.

Literatuur

1. A. Goldberg *et al.*, *SMALLTALK-80*, 4 Vols., Addison-Wesley, 1983-1984.
(Beschrijving van het SMALLTALK-systeem. Monumentaal geheel. Duidelijk geschreven met veel voorbeelden.)

2. W. Teitelman & L. Masinter, "The INTERLISP programming environment", in: D.R. Barstow, H.E. Shrobe & E. Sandewall (Eds), *Interactive Programming Environments*, McGraw-Hill, 1984, pp. 83-96.
(INTERLISP is een grote LISP-omgeving, die niet strikt eentalig is te noemen, maar waarin de verschillende subsystemen wel in hoge mate met elkaar zijn geïntegreerd.)

3. S. Pemberton, "The *B* programming language and environment", *CWI Newsletter*, No. 3, Juni 1984, Centrum voor Wiskunde en Informatica, Amsterdam.
(Heldere beschrijving van de programmeertaal *B* en de geïntegreerde programmeeromgeving die daarvoor op het Centrum voor Wiskunde en Informatica ontwikkeld wordt. Wordt op aanvraag gratis toegezonden. Bevat tevens een lijst van over *B* beschikbare publicaties.)

4. J. Heering & P. Klint, "Towards monolingual programming environments", *ACM Transactions on Programming Languages and Systems*, 7, 2 (April 1985). Tevens verschenen als rapport IW 185/81, Centrum voor Wiskunde en Informatica, Amsterdam, 1981.
(Dit is het gedetailleerde artikel waarop het voorgaande is gebaseerd. Geeft ook verdere referenties.)

5. J. Heering, "Taaldefinities als kern voor een programmeeromgeving", in: J. Heering & P. Klint, Colloquium Programmeeromgevingen, MC Syllabus 30, Centrum voor Wiskunde en Informatica, Amsterdam, 1983, pp. 69-81.
(Eentalige programmeeromgevingen en hun beperkingen. Principes van homogene meertalige programmeeromgevingen.)