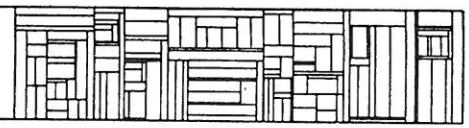
ISSN 0105-8517

NEDARVNING
(Resumé)

Kristine Stougård Thomsen

DAIMI PB - 212
April 1986

DATALOGISK AFDELING
Bygning 540 - Ny Munkegade - 8000 Aarhus C
tlf. (06) 12 83 55, telex 64767 aausci dk
Matematisk Institut Aarhus Universitet



PB - 212 K.S. Thomsen: Nedarvning (Resumé)

TRYK: RECAU (06) 12 83 55

Abstract

Der gives et resume af publikationerne PB-192, PB-209 og PB-210, som sammenfattes til en licentiatafhandling med titlen "Nedarvning". Emnet for afhandlingen er nedarvningsmekanismer i programmeringssprog. Resumeet indeholder en beskrivelse af den behandlede problemstilling, den anvendte metode og de opnåede resultater, herunder et mindre eksempel til illustration af den udviklede sproglige mekanisme. Desuden beskrives kort relationerne til den internationale forskning på området.

INDHOLD

RESUME AF LICENTIAT-AFHANDLING

Titel: Nedarvning

Forfatter: Kristine Stougård Thomsen

Emneområde: Programmeringssprog

Del-afhandlinger:

[1]: J.L.Knudsen, K.S.Thomsen: "A Conceptual Framework for Programming Languages", DAIMI PB-192

[2]: K.S.Thomsen: "Multiple Inheritance, A Structuring Mechanism for Data, Processes and Procedures", DAIMI PB-209

[3]: K.S.Thomsen: "Inheritance Used To Factorize Distributed Termination Detection Algorithms", DAIMI PB-210

1 Problemstilling

Gennemførelse af en dybtgående analyse af fordelene ved nedarvning i programmeringssprog. I tilknytning hertil udarbejdelse af et konkret forslag til udformning af en nedarvningsmekanisme, der kan realisere disse fordele ved beskrivelse af såvel data-strukturer som processer.

2 Metode og resultater

Overordnet betragtet, er der ved angreb af problemstillingen anvendt en tredelt metode:

1) Analyse:
   abstrakt, begrebsmæssig afklaring.
2) Design:
   af konkret sproglig mekanisme.
3) Afprøvning:
   af brugbarhed på konkret anvendelsesområde.

De tre delafhandlinger afspejler disse tre trin i metoden.

2.1 Analyse

Som udgangspunkt for den begrebsmæssige afklaring betragtes programmeringssprog som et værktøj med en dobbelt rolle i programmeringsprocessen:

- at understøtte programmørens abstraktioner og modelleringer af begreber fra anvendelsesområdet.
- at instruere datamaten i at realisere bestemte data-strukturer og processer.

Hovedvægten i begrebsafklaringen ligger på programmeringssprogs rolle som abstraktions- og modellerings-værktøj. Synet på programmeringssprog som middel til at instruere datamaten er inddraget, fordi det lægger begrænsninger på arten af abstraktions- og modellerings-mekanismer der realistisk kan indbygges i et sprog. Visse kompromiser mellem programmeringssprogs to roller er derfor nødvendige, hvis begrebsafklaringen skal have praktisk relevans.

Resultatet af begrebsafklaringen er præsenteret i [1], som udgør en begrebsmæssig ramme for diskussion af programmeringssprog. [1] består af fire dele, hvori der hhv. præsenteres:

a) en karakteristik af programmeringsprocessen som en modelleringsproces,
b) et syn på menneskelig begrebsdannelse og abstraktion,
c) en overordnet "deskriptor-entitet" model for programmeringssprog og programudførelser,
d) samt endelig, på basis heraf, en redegørelse for abstraktion i programmeringssprog.

Af særlig betydning er påpegningen af de centrale abstraktionsformer: klassificering, aggregering og generalisering, samt disses inverse: eksemplificering, dekomponering og specialisering. Herunder redegøres for en række fordele ved at understøtte generalisering/specialisering bedre end tilfældet er i hovedparten af eksisterende programmeringssprog.

Blandt disse fordele er:

- bedre begrebsmæssig modellering opnås
- trinvis forfinelse understøttes
- faktorisering af fælles del-beskrivelser understøttes
- adskillelse af forskellige anliggender understøttes
- et fleksibelt typebegreb med kvalifikation, også kaldet parametrisk polymorfi, kan baseres på specialiseringshierarkiet.

2.2 Design

Nedarvning er en fællesbetegnelse for en række sproglige udtryksmidler, der understøtter generalisering/specialisering ved at deskriptorer (typisk klasser) kan "arve" egenskaber fra andre deskriptorer.

I designet af en konkret nedarvningsmekanisme er følgende retningslinier brugt:

- Mekanismen skal give nedarvning på både data-strukturer og processer.
- En deskriptor skal kunne arve fra flere andre deskriptorer. (Multipel nedarvning i modsætning til singulær eller træstruktureret nedarvning.)
- Navnesammenfald skal behandles konsistent.
- Sammensætning af handlinger i en proces, der arver fra andre processer, skal være naturlig og fleksibel.

[2] beskriver den designede nedarvningsmekanisme i form af en skitse til et sprogdesign. Der er ikke tale om et egentligt sprogdesign, hvor alle udtryksmidler er integreret til en helhed, men om design af et enkelt sprogligt udtryksmiddel: nedarvning. Sproget tjener som ramme om udtryksmidlet og giver mulighed for at vise eksempler i rimelig detaljeringsgrad.

Sproget er baseret på et klassebegreb, der understøtter både data- og proces-beskrivelse. Objekter, dvs. instanser af klasser, er autonome processer, der indeholder data-strukturer og

procedurer og kan kommunikere med andre objekter via synkroniserede procedurekald. Multipel nedarvning er mulig for både klasser og procedurer.

Navnesammenfald

Ved navnesammenfald mellem egenskaber i en klasse (eller procedure), vælges konsekvent at betragte de forskellige forekomster af navnet som beskrivelser af den samme semantiske egenskab. Derfor kombineres beskrivelserne om muligt, og resultatet af kombinationen udgør den beskrivelse, der knyttes til navnet. For data-attributters vedkommende, fører kombinationen af sådanne beskrivelser med samme navn til en generaliseret fællesmængde af typerne. For procedurers vedkommende fører den samme kombinationsmåde til det, der i litteraturen om emnet kaldes "method combination", dvs. udførelse af alle procedurebeskrivelser knyttet til procedurenavnet.

Konventionen for navnesammenfald står i modsætning til virkefeltsreglerne i almindelige blokstrukturerede sprog, hvor et navn kan genbruges til et nyt formål i en indre blok. Sådanne muligheder for genbrug tjener hovedsagelig et praktisk/teknisk formål, mens muligheden for kombination af beskrivelser bidrager væsentligt til nedarvningsmekanismens udtryksstyrke. Dog må det i et komplet sprogdesign være ønskeligt også at inkludere mekanismer, der kan håndtere tilfældige navnesammenfald. Der findes sprog, der giver mulighed for begge slags navnesammenfald, men hvordan sådanne muligheder hensigtsmæssigt integreres i samme sprog, er ikke diskuteret i denne afhandling.

Der gives i [2] en række eksempler på brugbarheden af konventionen om kombination af beskrivelser ved navnesammenfald.

Sammensætning af handlingsdele

Forskellige handlingsdele i et objekt, arvet fra forskellige klasser, sammensættes ved alternering. (Det samme gælder i nedarvningshierarkiet for procedurer). Alternering er en art coroutine sekventiering, hvor skiftepunkter angives explicit i

handlingsdelene og forsynes med en betingelse, der angiver under hvilke omstændigheder, handlingsdelen kan genoptages efter skiftepunktet. Det angives ikke hvilken anden handlingsdel, der skal overtage kontrollen ved et skiftepunkt. Hermed fås en meget non-deterministisk sekventiering af handlingsdelene, der egner sig godt til mange sammenhænge, hvor forskellige klasser i et specialiserings/nedarvnings hierarki beskriver relativt uafhængige sider af de samme objekter.
Som typiske eksempler herpå nævnes i [2] database anvendelser, hvor procesdelen af et objekt tjener til kontrol af objektets kommunikation med omverdenen, f.eks. andre objekter eller brugere af databasen.

Eksempel

Eksemplet side 6 illustrerer konventionen for navnesammenfald og sammensætningen af handlingsdele. Eksemplet består af fire klasser fra en (tænkt og simplificeret) universitets-database.

Et objekt, x, som er instans af klassen Studenter-Instruktor, besidder samtlige variable, konstanter, operationer og handlinger beskrevet i klasserne Person, Student og Underviser. Konventionen for navnesammenfald sikrer, at x kun har een operation udskriv, som er sammensætningen af de tre specificerede versioner af udskriv. Dvs. et kald af udskriv på objektet x vil bevirke udskrift af samtlige egenskaber navn, adresse, årskort og lønnummer.

Objektet x vil udføre sine tre handlingsdele skiftevis med skiftepunkter ved hver mulig kommunikation ("? operationsnavn"). Genoptagelsesbetingelsen i alle skiftepunkter er ankomst af en passende henvendelse fra et andet objekt. Det betyder, at x vil afvente henvendelser fra andre objekter (f.eks. brugere af databasen) om at udføre Person-operationer, Student-operationer eller Underviser-operationer. Rækkefølgen af sådanne henvendelser bestemmer rækkefølgen, i hvilken x's handlingsdele bliver aktiveret og afbrudt. Hvis f.eks. den første henvendelse til x er et ønske om at foretage eksamenstilmelding, så aktiveres handlingsdelen fra klassen Student, eksamenstilmelding foretages,

handlingsdelen afbrydes, og x vil derefter afvente næste henvendelse fra omverdenen. En instans af klassen Studenter-Instruktor vil dermed være åben over for henvendelser fra objekter der i vilkårlig rækkefølge betragter den som en Person, en Student og en Underviser. Med andre ord kan studenter-instruktoren skifte vilkårligt mellem at opføre sig som en Person, en Student og en Underviser, hvilket intuitivt virker meget naturligt.

```
class Person;
  const navn: text;
  var   adresse: text;
  operation udskriv;
  begin  print(navn);
         print(adresse);
  end;
  operation adresseskift(in a: text);
  begin adresse:= a end;
  action
    do *
       ? adresseskift -> skip
     / ? udskriv      -> skip
    od
end;
```

```
class Student is-a Person;
  var årskort: integer;
  operation udskriv
       is-a Person.udskriv;
  begin
    print(årskort)
  end;
  operation eksamens-tilmeld;
  begin ... end;
  operation eksamens-frameld;
  begin ... end;
  operation eksamen;
  begin ... end;
  action
    do true ->
       * ? eksamens-tilmeld;
       if *
          ? eksamens-frameld -> skip
        / ? eksamen -> skip
       fi
    od
end;
```

```
class Underviser is-a Person;
  var lønnummer: integer;
  operation udskriv
       is-a Person.udskriv;
  begin
    print(lønnummer)
  end;
  operation registrer-kurser;
  begin ... end;
  operation kurser-slut;
  begin ... end;
  action
    do true ->
       * ? registrer-kurser;
       * ? kurser-slut
    od
end;
```

```
class Studenter-Instruktor is-a Student, Underviser;
end;
```

2.3 Afprøvning

Sammensætningen af handlingsdele er så forskellig fra tidligere forslag, at en grundigere afprøvning vurderes at være nødvendig. Derfor er der i [3] gennemført en sådan afprøvning inden for anvendelsesområdet distribueret programmering, nærmere betegnet på det klassiske problem omkring opdagelse af distribueret terminering.

Et let modificeret sprogdesign, tilpasset anvendelsesområdet, anvendes. Designet af nedarvningsmekanismen er den samme, mens resten af sproget blot udgør en rudimentær ramme for afprøvningen.

En række klassiske algoritmer til opdagelse af distribueret terminering realiseres vha. det modificerede sprogdesign. Fælles for realiseringerne er, at en termineringsalgoritme kan programmeres i to klasser, (en for lederprocessen og en for de øvrige processer) og at den egentlige distribuerede hovedberegning, hvis terminering skal opdages, kan programmeres i to andre klasser (leder og andre), der arver fra hver sin af de to termineringsklasser hørende til en specifik termineringsalgoritme. Det bærende element i denne realisering er alterneringen. Den coroutine-agtige sekventiering passer meget naturligt til det ønskede samspil mellem en termineringsalgoritme og en hovedberegning, idet termineringsalgoritmen skal sammenflettes med hovedberegningen på en sådan måde, at termineringsalgoritmen udføres når og kun når hovedberegningen af en eller anden grund er passiv.

Ved anvendelsen af nedarvning ved programmering af termineringsalgoritmerne er opnået store fordele. For det første er beskrivelsen af de to anliggender, terminering og hovedberegning, blevet adskilt, hvilket øger overskueligheden af dem begge væsentligt. For det andet betyder denne adskillelse, at termineringsalgoritmen er blevet faktoriseret ud, så den kan bruges af forskellige hovedberegninger uden at skulle gen- eller om-skrives. Desuden kan den samme hovedberegning sammensættes med en vilkårlig af termineringsalgoritmerne uden at skulle ændres.

Hermed har den designede nedarvningsmekanisme vist sin brugbarhed og nytteværdi i relation til en vigtig klasse af proces-beskrivelser inden for distribueret programmering. I [3] peges på mulige mere generelle anvendelser inden for samme område.

3 Relationer til Øvrige Forskning

[1] er baseret på en række arbejder af forskere fra forskellige områder: Systemarbejde, Databaser, Kunstig Intelligens og Programmeringssprog. Nytænkningen i [1] består i at bringe ideer fra disse områder sammen og bearbejde dem, så de danner en sammenhængende begrebsramme, samt på basis heraf at gennemføre en grundig analyse af abstraktion i programmeringssprog. Således er de første tre dele af [1] (afsnit 2.1 pkt. a,b og c) i stort omfang en bearbejdning og tilpasning af andres arbejder, mens den fjerde del, analysen af abstraktion i programmeringssprog, er egentligt original-arbejde.

Der er udviklet mange programmeringssprog med nedarvning, heraf også en del med multipel nedarvning, men kun meget få med nedarvning på processer. Et selvstændigt afsnit i [2] sammenligner den udviklede nedarvningsmekanisme med dem, der findes i tidligere sprog.

Jeg har ikke kendskab til tidligere forsøg på sproglig understøttelse af faktoriseret beskrivelse af termineringsalgoritmer, som det der er beskrevet i [3].

Afslutningsvis er det naturligt at sammenligne det syn på abstraktion i programmeringssprog, som denne afhandling repræsenterer, med R.D.Tennent's klassiker inden for litteraturen om abstraktion: "Language Design Based on Semantic Principles", (Acta Informatica 8, (1977) s. 97-112).

Analysen af abstraktion i Tennent's artikel er gennemført ud fra et semantisk udgangspunkt, som bygger på matematikkens traditioner. Analysen af abstraktion i [1] er derimod gennemført med udgangspunkt i begrebsmæssig modellering, som er en disciplin

inden for kunstig intelligens med forbindelser tilbage til den klassiske filosofi om menneskets erkendelse.

Fælles for Tennent og [1] er, at abstraktion basalt set er at skabe navngivne, parametriserede deskriptorer (typer, procedurer, klasser etc.).
I [1] skelnes mellem abstraktionsformerne klassificering, aggregering og generalisering. Klassificering er det at danne en abstraktion ud fra hvilken, der kan genereres instanser. Aggregering og generalisering implicerer klassificering, men er yderligere karakteriseret ved på forskellig vis at danne abstraktioner på basis af andre abstraktioner.
Tennent beskæftiger sig med klassificerings-aspektet af abstraktion og formulerer to vigtige principper. "Principle of abstraction" udtrykker, at der bør kunne foretages klassificering over alle semantisk meningsfulde syntaktiske kategorier i et sprog. "Principle of correspondence" udtaler sig om, hvilke roller instanser af abstraktioner bør kunne spille i relation til bl.a. parametrisering.
[1] kan ses som et supplement til Tennent's principper i form af nogle kvalitative betragtninger over hvordan abstraktioner bør kunne dannes på basis af andre abstraktioner via aggregering og generalisering.

I et konkret sprogdesign kan Tennent's principper benyttes sammen med udformning af mekanismer til understøttelse af klassificering, aggregering og generalisering. "Principle of abstraction" benyttes til at afgøre over hvilke syntaktiske kategorier, der skal kunne klassificeres. Herpå designes mekanismer til aggregering og klassificering, og alle deskriptorer, inklusive deskriptorer opnået ved aggregering og generalisering underlægges "principle of correspondence".

[1] skal altså ses ikke som et alternativt, men som et supplerende syn på abstraktion i programmeringssprog i forhold til Tennent's.