13
Programmeringsomgivelser til Lisp.

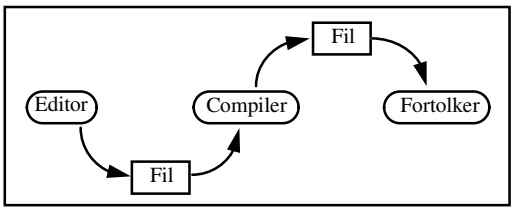
Model af en compiler-centreret omgivelse.
Workspace-basered omgivelser.
En simpel Lisp omgivelse.
Interlisp: en uniform omgivelse.
Struktur-orienterede liste editorer.
Emacs og Lisp omgivelser.

PS4 - Lisp omgivelser

Noter

Vi har indtil nu i dette kursus set på det funktionsorienterede programmeringsparadigme, funktionsorienterede programmeringssprog, og avancerede objekt-orienterede sproglige mekanismer. På den sproglige front har forskellige dialekter af Lisp spillet en væsentlig rolle. Det er naturligt at afrunde vores orientering mod Lisp med en forelæsning om Lisp programmeringsomgivelser. Dette er emnet i dette kapitel.

Model af en simpel compiler-centreret omgivelse.



- Integrationsgraden mellem værktøjer er lav.
  - Separate værktøjer.
  - Kommunikation mellem værktøj sker gennem filer, som bor i operativsystemet.
- Inkrementalitet:
  - Typisk bearbejdes relativt store programbeskrivelser.
- Persistens af programbeskrivelser:
  - Gennem filbegrebet.

Noter

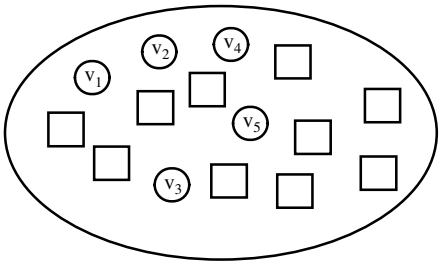
Omgivelsen, som er modelleret og beskrevet på denne side, kender vi alle talrige eksempler på. Det eneste specielle værktøj, som er strengt nødvendigt er compileren. Editoren er en teksteditor, som ikke er dedikeret til programudvikling. Fortolkern kan være den i maskinen iboende fortolker af maskinkode, eller en mere specialiseret fortolker af "mellemkode".

Ikke alle compiler-centrerede omgivelser er lige så simple som den, vi har modelleret på denne side. Der kan eksempelvis komme en **debugger** på banen, som i det simple tilfælde kun er afhængig af informationen fra compileren; en mere avanceret debugger vil slå bro mellem udførelsen og programbeskrivelsen, altså på en eller anden måde tillade os at følge programudførelsen via programbeskrivelsen.

Vi har kaldt ovenstående omgivelse for compiler-centreret. Vi kunne også kalde den for en **operativsystem-baseret omgivelse**, fordi operativsystemet er det interaktive bindeled mellem værktøjerne, og fordi integrationsmediet ene og alene er operativsystemets filsystem. Dette er en kontrast til den omgivelse, vi ser på på næste side.

Årsagen til, at vi bringer ovenstående model på banen, er at vi ønsker et modelmæssigt sammenligningsgrundlag i forhold til nogle af de omgivelser, vi skal se på i resten af dette kapitel.

En workspace-baseret omgivelse.



Objekttyper:

v_i værktøj

program-beskrivelse

En *workspace* er et fælles rum for programbeskrivelsesobjekter, værktøj, og programtilstand (dataobjekter).

Problem: programbegrebet er svagt. Hvad identificerer et program?

- Integration
  - Værktøj kan samarbejde og kommunikere i workspacet, direkte eller via programmør-skabte forbindelsesled.
- Inkrementalitet
  - Bearbejdning (skabelse og afprøvning) af små programenheder, uden dyb tekstuel indlejring.
- Persistens:
  - Af mængder af sammenhørende definitioner.
  - Af hele workspacet.

PS4 - Lisp omgivelser

Noter

En workspace-baseret programmeringsomgivelse repræsenterer en radikal anderledes model end den, vi studerede på forrige side. Vi er i en workspace-baseret omgivelse langt mindre afhængig af filsystemet og operativsystemet som integrations- og kommunikationsmedium. Filer kan dog spille en rolle som persistente enheder, hvorpå dele af, eller hele workspacet kan gemmes permanent.

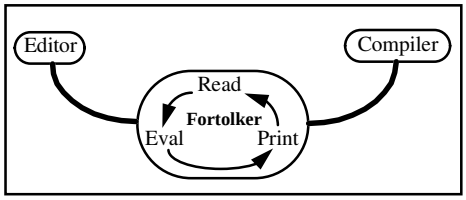
Hvis et workspace ækvialeres med RAM-lageret på en maskine vil en workspace-baseret omgivelse let blive til en **én-bruger omgivelse**. Man kan dog sagtens forestille sig en mere avanceret workspace, som er distribueret, og som giver flere programmører tilgang.

Programbegrebet: I og med at et program er en samling af små programbeskrivelser, som ikke er "indrammet" eller samlet syntaktisk i større sammenhørende enheder, er det et problem at identificere, hvad der udgør et program. Et program bidrager med en samling af beskrivelser, der flyder sammen med dem som er der i forvejen (hidrørende fra andre "programmer", eller stammende fra beskrivelser, som omgivelsen er født med). Det er derfor nødvendigt at køre en eller anden **administration af, hvilke dele der hører sammen i lidt større programenheder** (moduler, klasser, eller llignende). Oftest benytter man også ofte disse lidt større programenheder som *persistente enheder* (altså de enheder, der lagres persistent i det omkringlæggende filsystem).

Er der noget iboende i workspace ideen, som giver os inkrementaliteten? Egentlig ikke; men workspace-baserede omgivelser er traditionelt orienteret mod en **snæver og tæt dialog med programmøren**, hvor man uden nævneværdi forsinkelse forventer at få accepteret definitioner og re-definitioner, samt at man i dialog kan afprøve de enkelte definitioner på en bestemt data-scene. Det hører også med i billedet, at der **ikke skal genereres et resultat på en fil** for hver værktøjsanvendelse . Resultatet kan afspejles gennem et objekt, som bor i workspacet, og som senere kan tilgåes umidelbart.

Både Lisp systemer og Smalltalk er eksempler på omgivelser, som vi vil kalde workspace-baserede.

En simpel Lisp omgivelse.



- Lisp-systemet er **ét værktøj**: en fortolker.
- Det er ønskværdigt at **tilkoble en editor**, som tillader at en samling af definitioner editeres som en helhed.
  - Løs sammenhæng mellem fortolker og editor.
  - Attraktivt med en fastere forbindelse mellem editor og fortolker:
    - Editoren ind i fortolkeren (editoren som et værktøj i workspacet)?
    - Fortolkeren bringes ind i editoren?
- Man kan forestille sig at tilkoble **andet eksternt værktøj**: f.eks. en compiler.

PS4 - Lisp omgivelser

Noter

Den simple model, som er tegnet på denne side, er typisk for mange primitive Lisp systemer, og for den sags skyld for omgivelser for andre, inkrementelt håndterede sprog, der er baseret på en “read-eval-print” loop. ML er et sådant eksempel.

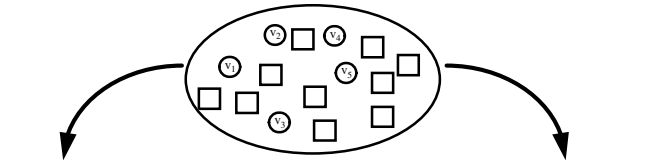
Vi har udnævnt read-eval-print loop-en, som tegnet ovenfor, til en fortolker. Der er intet til hinder for, at dette værktøj kan baseres på en **incrementel oversætter** og en mere primitiv fortolker (for maskinkode eller et mellemniveau sprog).

Det største problem i denne model er den svagt specificerede tilknytning mellem “Lisp systemet”(fortolkeren) og editoren. Editoren er et centralt værktøj i en programmeringsomgivelse, og det er derfor væsentligt, at der er en tilfredsstillende integration mellem netop editoren og udførelsesværktøjet. Det er også fristende at indpode noget kendskab til programmeringssproget i editoren.

Vi vil lidt senere i dette kapitel se, hvordan den simple model ovenfor kan udvikle sig. Man kan enten forestille sig at editoren bevæger sig ind i Lispsystemet, som altså på denne måde beriges, og måske bevæger sig hen imod en workspace model, ala den vi så på på tidligere i dette kapitel. Man kan også forestille sig at Lispsystemet på en eller anden måde trækkes ind i editoren, eller afvikles under kontrol af editoren.

Følger ovenstående **workspace-ideen**, som diskuteret på forrige side? Editoren og compileren er ikke værktøjer i omgivelsen, så set fra deres pespektiv er svaret et “nej”. Når selve fortolkeren er aktiv findes der et workspace, hvori Lisp systemets dataobjekter befinder sig. Så fra et isoleret fortolkerperspektiv er svaret et “ja”. Vi har også allerede erkendt, at vi har brug for at få i det mindste editoren ind som et værktøj i workspacet (eller i det mindste få en bredbåndsforbindelse mellem editoren og fortolkeren).

Forskellige typer af workspace-baserede Lisp omgivelser.



Uni-form model Workspace *Multi-form model*

- Én primær program repræsentation.
  - Listerepræsentation eller
  - Syntakstræ.
- Fælles for alt væsentligt værktøj.
- Editoren er struktur-orienteret.
- Kildeudtrykket kan tilgåes ved at pretty printe listrepræsentationen af en Lisp definition.

Interlisp

- To eller flere væsentlige program repræsentationer.
  - Tekst og
  - Udførbar beskrivelse.
- Editoren arbejder på tekstrepræsentationen.
- Teksten skal parses (*reades*) inden udførelsen.
- Det er kompliceret at finde kildeudtrykket for en givet Lisp definition.

Alle andre

PS4 - Lisp omgivelser

Noter

Udgangspunktet for denne diskusion er workspace modellen, som vi tidligere har introduceret. Afhængig af de forskellige former for væsentlige programrepræsentationer, når vi frem til det vi her kalder den uni-forme model og den multi-forme model.

Navnet “uni-form” har en dobblet betydning . Navnet hentyder dels til, at der kun er **én væsentlig form, hvorpå programmer repræsenteres**; dels til at dette giver en større grad af uniformitet end i den alternative model, hvor der er flere former af repræsentation.

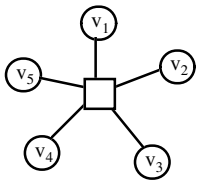
I den multi-forme model kan den *udførbare beskrivelse* være en liste-form (som fortolkes), eller det kan være mellemkode, eller sågar maskinkode.

Tilgang til kilde-formen af en Lisp definition: I den uni-forme model er dette altid muligt og relativt enkelt: Givet den interne repræsentation kan man generere en tekstuel form via pretty printing, som editoren kan præsentere. Pretty-printeren er en integreret del af liste struktur-editoren. I en multi-form omgivelse er dette ikke altid muligt. Det er ofte et problem at lokalisere kilde-formen af en Lisp definition, som man “falder over” i workspacet. Mange Lisp systemet har et bogholderi over i hvilken kildefil hvilke Lisp definitioner befinder sig.

Reading er processen hvormed med transformerer (parser) en tekstreng, som er et Lisp udtryk, til en liste. **Printing** er den omvendte transformation (også kaldet unparsing). **Pretty-printing** er en bestemt form for printing, hvorman forventer et *æstetisk behageligt layout* af den genererede tekst. Dette er et “must” i struktur-orienterede liste editorer.

Interlisp: en uni-form omgivelse.

- Interlisp: Mere omgivelse end sprog.
- En editor-iboende omgivelse:
  - Også kendt som en '*residential environment*'.
  - Flere generationer af listestruktur editorer.
- Righoldig repertoire af andet værktøj.
  - DWIM, Masterscope, profiler, grapher, filepackage, ...
- Filepackage:
  - Administrer filtilhørsforhold af definitioner.
  - Holder styr på, hvilke definitioner der er ændret siden sidste 'save'.
  - Mapper Lisp definitioner til (tekst)filer.
    - Råder bod på ustabilitet af workspacet.
    - Ulitimativ garbage collection.
    - Befordrer transport til andre omgivelser.



Fire editor generationer:

Cons, car, cdr editering.

Edit: Kommando-baseret TTY liste editor.

Dedit: Struktur-terroriserende.

Sedit: Liste struktur editor med tekst editor agtigt interface.

Noter

Interlisp havde sine velmagtsdage på det tidspunkt, hvor det kørte på store time-sharing systemer (ala Dec10) og ikke mindst da det blev overført til tidlige, Lisp dedikerede arbejdsstationer. Interlisp kom også på generelle arbejdsstationer, men det mistede hastigt terræn og popularitet.

Selv om Interlisp ikke er af stor relevans idag, har det sat sine spor i landskabet af programmeringsomgivelser. I sine velmagtsdage var Interlisp den ypperligste omgivelse, man kunne byde på, og der var en meget stor afstand til "konventionelle omgivelser". Pionerarbejdet omkring struktur-orienteret editering er foretaget i Interlisp, ligesom demonstrationen af, at tæt integrerede omgivelser overhovedet var muligt for en stor dels vedkommende blev foretaget i Interlisp. Interlisp har uden tvivl også haft en indflydelse på Smalltalk omgivelsen. Det er værd at bemærke, at begge omgivelser stammer fra samme laboratorium: fra Xerox Palo Alto Research Center (Xerox Parc), i Silicon Value syd for San Francisco i Californien.

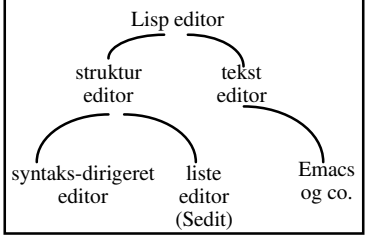
Sproget Interlisp, såvel som en del andre Lisp dialekter, er idag helt og holdent domineret af Common Lisp.

På næste side vil vi diskutere struktur-orienteret editering i større detaljer, end pladsen på denne side tillader.

Struktur-orienterede liste editorer.

- Direkte editering af lister og S-udtryk: den interne repræsentation.
- I brugergrænsefladen
  - præsenteres den interne repræsentation som sædvanlige teksuelle Lisp lister.
  - manipuleres præsentationen gennem veldefinerede transformationer på den underlæggende liste repræsentation.
    - kun strukturelt meningsfyldte editeringsoperationer kan foretages.

Klassificering af struktur-editorer:



- Automatisk pretty printing:
  - Man skal ikke bekymre sig om indrykningen af sine Lisp udtryk.
  - Programmøren kan ikke selv bestemme sin indrykningsstil.
  - Prettyprinteren skal have kendskab til formatteringsregler for de forskellige syntaktiske former (define, let, cond, if mv.) i sproget.
  - Editoren kan tilpasse sig bredden og/eller højden af vinduet.
    - Evt. kan bestemte delstrukturer undertrykkes i præsentationen.

Noter

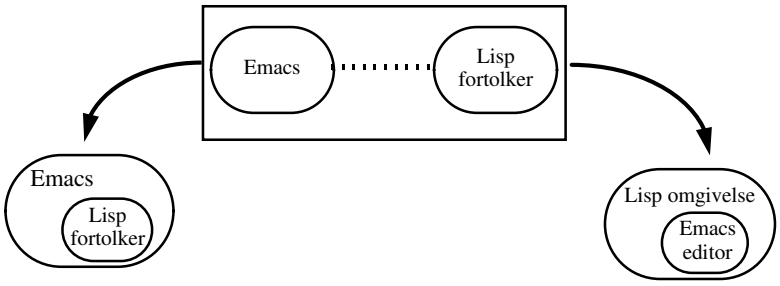
Syntaks-dirigerede editorer er medtaget i ovenstående klassifikation af struktur-orienterede listeeditorer, idet de syntaks-diregerede editorer er en naturlig videreudvikling af disse. En indlejret listestruktur er i bund og grund en træstruktur. En syntaks-dirigeret editor arbejder på en intern repræsentation, som er et syntakstræ, typisk (men ikke nødvendigvis) et abstrakt syntakstræ.

Læg mærke til, at de to udsagn

> Man skal ikke bekymre sig om indrykningen af sine Lisp udtryk, og
>
> Programmøren kan ikke selv bestemme sin indrykninsstil.

er to udlægninger af den samme basale observation. Man vælger udsagn ift. ens grundholdning til de to editeringsstilarter.

Emacs og Lisp omgivelser.



GNU Emacs er i sig selv en *Lisp omgivelse med et avanceret tekstuelt interface.*

Man kan starte en Lisp fortolker fra Emacs, og integrere fortolkeren med liste tekstediting.

Nogle moderne, kommercielle (Common Lisp systemer) indeholder som en integreret del en Emacs-baseret tekst-editor.

- Emacs lisp bibeholdt, men stærk integration med Common Lisp.
- Emacs Lisp udskiftet med Common Lisp.

PS4 - Lisp omgivelser

Noter

På denne slide vender vi tilbage til, hvordan en stærk og udvidelig tekst-editor ala Emacs, hvis eget udvidelsessprog er en simpel Lisp dialekt (E-lisp, eller Emacs-Lisp) kan tilpasses en Lisp fortolker.

Teknikken til venstre virker så at sige for alle Lisp systemer, i det mindste i en primitiv og generel udgave. Mange studerende på dette kursus har prøvet denne teknik på et af de Scheme systemer, vi har kørt. Vi bruger det også på CLOS. Man kan naturligvis forestille sig at meget tættere integrering af den fremmede Lisp og Emacs, som tekst editor. (Måden at lave dette på ville være at skrive noget understøttende programmel til dette i E-lisp).

Teknikken til højre, særligt med udskiftning af E-lisp med en egentlige Lisp (Common Lisp), er springbrættet til et gedigent multi-formt Lisp system. Flere sådanne er på markedet idag.