UML bruksmønstre

Manus SYS2

Knut W. Hansson
Førstelektor Informatikk
Høgskolen i Buskerud

Innledning

Kort repetisjon av systemteori

Overhead systemer

- ✓ Deler, relasjoner, grense og omgivelser
- ✓ Åpne/lukkede systemer
- ✓ Artefakters formål/hensikt. Hensikten med et lagd system kan ligge i
  - prosessen: Lage fordi det er lærerikt eller moro (elever/studenter)
  - resultatet: Lage fordi det skal brukes til noe (privat eller av en organisasjon)
  - begge deler: Prosessen er uansett hensikt alltid lærerik for dem som deltar – de lærer om interesseområdet (”universe of discourse”), om behov og muligheter, om dem som skal ha systemet, om andre deltakere i prosessen, om selve utviklingsprosessen og annet.
- ✓ Artefakter vurderes etter i hvilken grad hensikten oppnås (det teleologiske prinsipp).

Informasjonssystemer

- ✓ Informasjonssystemer er **systemer** som behandler **informasjon**.
- ✓ Informasjonssystemer er **åpne**. De omgivelsene som er mennesker, kalles **interessenter** og det er (noen) av dem som skal ha informasjonen.
- ✓ Informasjonssystemer er **artefakter**, og hvis hensikten ligger i resultatet, så er den **å fremskaffe (nyttig) informasjon**.
- ✓ Det er interessentene som må angi **hvilken informasjon som er nyttig for dem.**
- ✓ **Det blir da viktig å finne ut hvilken hensikt interessentene har med systemet.**

Minne om UMLs perspektiver (”views”)

Overhead

UML søker å beskrive systemet fra fem perspektiver/views. **Sentralt er ”Use Case View”**, vanligvis oversatt til **”bruksmønsterperspektivet”**. Her analyseres systemet **fra utsiden**, det vil si at vi er opptatt av **hvilken hensikt interessentene har med systemet** (jfr. det teleologiske prinsipp).

Begreper

Systemet under utvikling (**system under development** eller **SuD**) er det systemet vi skal lage, og som vi analyserer/designer. UML forutsetter at dette er et objektorientert system.

Interessenter (stakeholders) er alle som har en interesse av hvordan systemet ”oppfører seg”. Det kan være personer, avdelinger, andre systemer, organisasjoner osv., f.eks. skattekontoret som skal ha meldinger om skattetrekk og ledelsen som skal rapporter. De befinner seg i systemets omgivelser. Noen interessenter – men ikke alle – skal samhandle med systemet, og kalles da **aktører**. Det er aktørene som utgjør systemets omgivelser i systemteoretisk forstand. Interessent er altså en mer utvidet begrep, siden også noen utenfor omgivelsene kan ha interesse av systemets oppførsel.

Aktører (**actors**) er interessenter som samhandler med systemet. Aktørene har altså en oppførsel. Aktørene deles i **primæraktører** (**primary actors**) og **støtteaktører** (**supporting actors**). En aktør kan være primæraktør i forhold til et bruksmønster, og støtteaktør for et annet.

Primæraktører handler direkte med systemet og ofte – men ikke alltid – er det deres handlinger som tar initiativ i forhold til systemet og som systemet reagerer på. (Systemet kan også reagere på andre hendelser, f.eks. at bestemte tidspunkt – slutten av måneden, bestemt klokkeslett o.l. – blir nådd eller at noen setter et kort inn i en kortleser. Disse bryterne **(triggers**) regnes ikke som aktører, siden de ikke har egen oppførsel.) Noen primæraktører handler på vegne av andre, f.eks. en telefonselger som registrerer kjøp i systemet på vegne av en telefonkunde. Da kalles den som faktisk har interessen for **den egentlige primæraktør** (**ultimate primary actor**). Etter hvert som teknologien endres, vil gjerne den endelige primæraktøren endres til primæraktør, som når kjøperen selv registrerer kjøpet på en webside.

Støtteaktører er aktører som utfører en eller annen tjeneste for systemet, f.eks. et fakturasystem som skal gi data til vårt system på forespørsel fra det.

Bruksmønstre (**use cases**) er en avsluttet samling handlinger som til sammen gjør noe av signifikant verdi for en aktør. Det kan være produksjon av utdata, men like gjerne at systemets tilstand er endret, f.eks. at en ny karakter er registrert på en elev, at en elev er slettet osv. Bruksmønstre representerer systemets **formål** (**goals**). Bruksmønstre har egenskaper. UML har ingen definitiv liste over slike egenskaper, men flere foreslår følgende:

- ✓ Navn: Navnet på bruksmønsteret
- ✓ Navnet på (del-)systemet som bruksmønsteret tilhører
- ✓ Interessent
- ✓ Primæraktør
- ✓ Kort beskrivelse
- ✓ Detaljeringsnivå: Grov oversikt (summary level), brukernivå (user level) eller subfunksjon nivå (sub function). Noen anbefaler ikoner – henholdsvis en sky, sjø eller en fisk.
- ✓ Pre- og postbetingelser: Prebetingelser stiller krav om hva som må være sant *før* bruksmønsteret kan utføres – postbetingelser hva som bruksmønsteret skal garantere når det er ferdig utført. Postbetingelser kan være minimale garantier (alltid sanne) og suksessgarantier (sant hvis alt går bra).
- ✓ Brytere (triggers)

Scenarier (**scenarios**) er beskrivelser av handlingsmønstre. Det er en form for fortellinger, der aktørene og systemet inngår, og som viser hva som skjer når aktøren bruker systemet for å oppnå sitt mål. De enkleste scenariene ”ender godt” – alt går vel – og kalles hovedscenarier (**main success scenarios**). Vi vet jo av erfaring at ting kan gå galt (Murphys lov) og derfor er det også behov for å håndtere unntakene (**exceptions**) og de beskrives som **utvidelser** (**extensions**).

Hva beskrives i bruksmønsterperspektivet

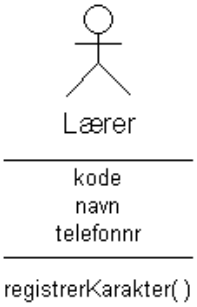
Bruksmønsterperspektivet konsentrerer analysen om hvem/hva som utgjør systemets omgivelser og hvilken funksjonalitet systemet skal ha. **Hensikten er å skaffe et overblikk over funksjonalitetskravene til systemet.** Kravene gjelder kun hva systemet skal kunne gjøre, ikke hvordan. F.eks. spør man *ikke* etter sikkerhetskrav, kvalitetskrav, responstid, rammer i form av kostnader, tidsfrister, operativsystemer, maskinmiljø osv. Som kravspesifikasjon er det altså nokså snevert. Mange av de resterende kravene tar man hensyn til senere, når systemet utformes.

Omgivelser

Omgivelsene er **mennesker og andre systemer som spiller en rolle** i forhold til systemet vi skal lage. De kalles derfor **actors** – vanligvis oversatt til **aktører** – men merk alluderingen til

skuespillere som har en rolle. Aktører er altså ikke ”Per” og ”Kari”, men ”personalsekretær” og ”personalsjef”. Symbolet for en aktør er en ”stickman” (= ”strekmenneske”?):
Overhead/tavle

Aktører benevnes med et **substantiv** som beskriver den **rollen de har i forhold til systemet**. Aktører er *klasser* (i objektorientert forstand) og kan følgelig *instansieres*, dvs. at det kan være mange av hver. F.eks. representerer aktøren *Lærer* ovenfor mange lærere med de samme egenskapene. (Dette tilsvarer slik sett entitetstyper i datamodellering.) Siden aktørene er objektorienterte klasser, kan det knyttes både attributter (data) og metoder (handlinger som aktøren kan utføre på anmodning) til dem:
Overhead/tavle



Aktørene vil senere bli representert inne i systemet ”rett innefor systemgrensen” av klasser som håndterer den faktiske, fysiske brukeren (aktøren) på utsiden av systemgrensen. Slike klasser kalles **grenseklasser** (”boundary classes”). Gjennom et grensesnitt (MMI, GUI) gjør aktøren ønskene sine kjent: Registrer karakteren 5 i engelsk på eleven Sylvester i klasse 2B”. Grenseklassen tar så – på vegne av aktøren – kontakt med andre klasser/objekter innover i systemet for å få dette utført.

Det er *ikke* vanlig å spesifisere aktørenes attributter og metoder før senere i analysen, og de vises sjelden i bruksmønsterdiagrammene. (Ref.: Prinsippet om ”need-to-know” i UML.)

Funksjonalitet

Systemets funksjonalitet defineres av **hva aktørene vil at systemet skal gjøre for dem**, hvilket tilsvarer systemets **formål**. Funksjonaliteten inndeles i **bruksmønstre**, det vil si **en avsluttet samling handlinger som til sammen gjør noe av signifikant verdi for en aktør**. Det kan være produksjon av utdata, men like gjerne at systemets tilstand er endret, f.eks. at en ny karakter er registrert på en elev, at en elev er slettet osv. Symbolet for et bruksmønster er en ellipse:
Overhead/tavle

Siden bruksmønstre representerer en avsluttet handling, benevnes de med et **verb**, helst som en ordre i imperativ, **som beskriver hva systemet skal gjøre for aktøren**.

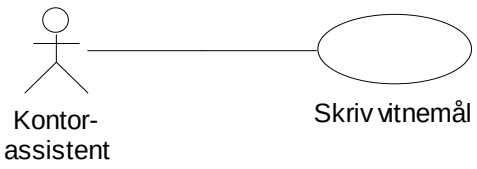
Relasjoner

Siden dette beskriver et system, er det **relasjoner (sammenhenger) mellom delene**. Det er flere, forskjellige relasjonstyper, kalt **assosiasjon**, **arv** og **avhengighet**.

Assosiasjon

Relasjonen mellom en aktør og et bruksmønster, angir hvilken funksjonalitet aktøren ønsker = ”hvem ønsker hva”, og kalles **assosiasjon**. Symbolet for assosiasjon er en enkelt strek, uten retningsangivelse[1]:

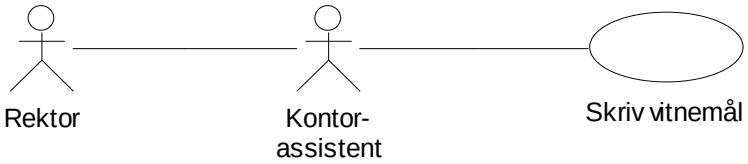
Overhead/tavle



Dette er den vanligste relasjonsformen i bruksmønsteranalysen. Legg merke til at dette ikke skal bety at kontorassistenten skriver vitnemål – dette dokumenterer at kontorassistenten vil at systemet skal skrive ut vitnemål når kontorassistenten ber om det.

Det kan også være **assosiasjoner mellom to aktører**:

Overhead/tavle



Dette er relativt sjelden og stort sett fordi brukere som er med på analysen insisterer. Det innebærer at det er rektor som er den **egentlige** interessent – kontorassistenten bare representerer rektor overfor systemet. Legg merke til at rektor ikke er i systemets omgivelser, og følgelig egentlig ikke skal være med i analysen.

Det kan *ikke* være **assosiasjoner mellom bruksmønstre**.

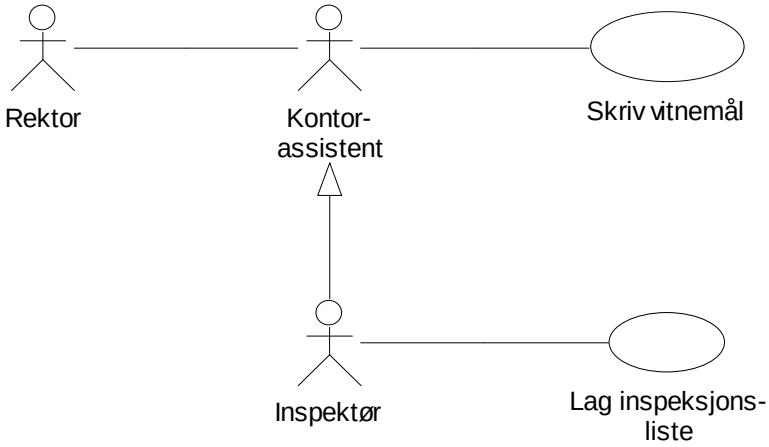
Arv

Arv innebærer at ett objekt overtar noen egenskaper fra et annet. Det objektet som arver, er subobjekt og det andre er metaobjekt (også kalt superobjekt).

Hvis en **aktør arver fra en annen aktør**, innebærer det at alle de ønskene metaaktøren har til systemet, har også subaktøren, men subaktøren har sine egne ønsker **i tillegg**. Symbolet for arv er en lukket pil (uten ”fyll”):

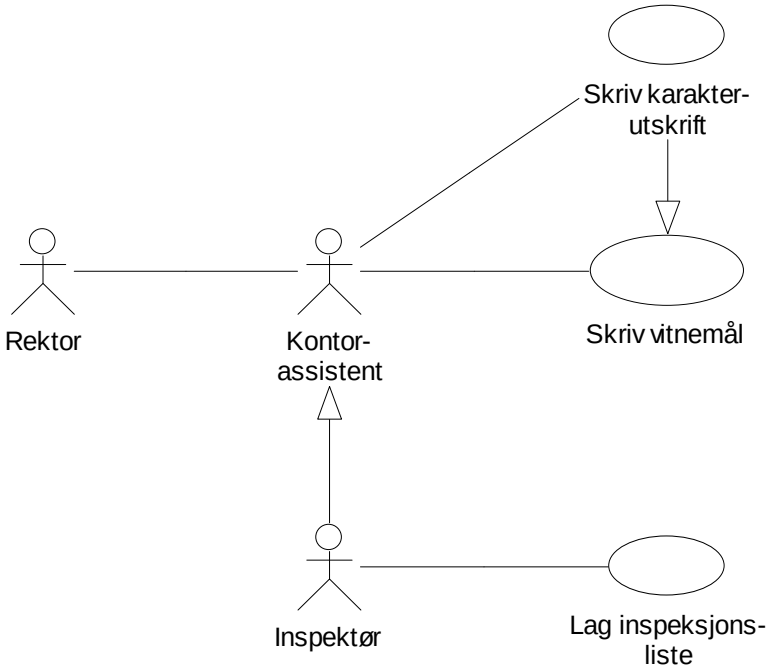
[1] Enkelte forfattere har advokert for at assosiasjonen kan påføres en pil, for å angi hvem som *initierer* bruksmønsteret. Dette er *ikke* UML standard, og bryter med ”helhetsprinsippet” ettersom en pilsatt assosiasjon ellers innebærer en *enveis* sammenheng. Jeg kan selv ikke se behovet for å angi hvem som tar initiativ og er uansett imot en slik endring av pilens betydning.

Overhead/tavle



Det kan være naturlig å tenke på arvehierarkier som et organisatorisk hierarki, men det bli helt feil. Dette er ikke et organisasjonskart, men viser at både kontorassistenten og inspektøren ønsker å få skrevet ut vitnemål, men i tillegg vil inspektøren få laget inspeksjonslister. (Faktisk vil de som står lavest i arvehierarkiet ha flest ønsker, og det kan ofte være de høyeste i organisasjonshierarkiet.)

Hvis et **bruksmønster arver fra et annet bruksmønster**, innebærer det at subbruksmønsteret utfører omtrent samme funksjonalitet, men på en litt annen måte:
Overhead/tavle



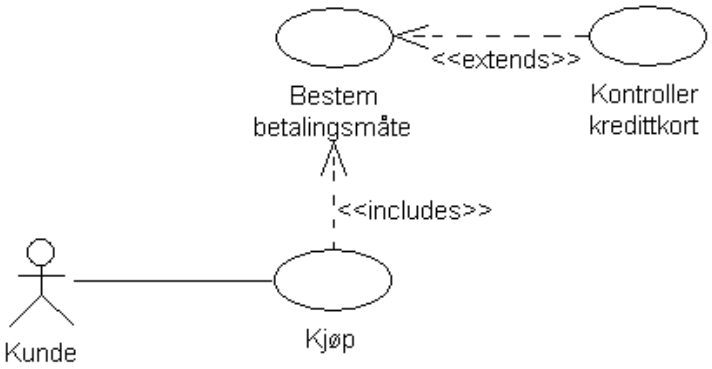
Å skrive en karakterutskrift er tydeligvis omtrent det samme som å skrive et vitnemål. Legg merke til at vi plasserer symbolene fritt – det er ikke nødvendig at subobjektet står under metaobjektet. Legg også merke til at symbolenes størrelse er uten betydning. Legg dessuten merke til at slik det nå er tegnet, er også inspektøren interessert i å få laget karakterutskrifter – det er arvet fra kontorassistenten.

Siden bruksmønstre og aktører er så forskjellige, er **arv mellom bruksmønstre og aktører** helt uaktuelt.

Avhengighet

Avhengighet oppstår bare mellom bruksmønstre og innebærer dypest sett at den avhengige kanskje må endres hvis den uavhengige blir endret. Mer spesifikt anvendes avhengighet for å uttrykke at et bruksmønster ”bruker” et annet:

Overhead



Et kjøp inkluderer alltid (=”includes”) å bestemme betalingsmåte, som på sin side kan ha behov for (=”extends”) å kontrollere kredittkort. Kommentarene <<includes>> og <<extends>> kalles stereotyper, og brukes til å modifisere symbolet. Den prikkede, åpne pilen alene betyr avhengighet, men stereotypen angir en spesiell type (subtype) avhengighet.

Det er ikke ofte det er aktuelt med slike avhengigheter, og de kan lett overdrives. Eksempelet er ikke helt godt, fordi det er tvilsomt om kunden faktisk ønsker at systemet skal sjekke kredittkortet – er det virkelig en funksjonalitet som kunde vil at systemet skal kunne gjøre for ham/henne? Det kan også diskuteres om ikke bestem betalingsmåte bør sees som en naturlig del av et kjøp og inngå i den ”avsluttede samling handlinger som til sammen gjør noe av signifikant verdi for en aktør” (jfr ovenfor). Tegnemåten ovenfor kan likevel være aktuell hvis andre aktører spesielt har ønsket seg *kontroll av kredittkort* (alene).

Sammenheng mellom bruksmønsterdiagrammer

Bruksmønsterdiagrammer blir fort ”overlesset” med for mange objekter og relasjoner, så vi mister oversikten. Siden et uttrykkelig formål med bruksmønsteranalysen er å skaffe overblikk over aktørenes ønsker for systemet, må diagrammene da deles. En vanlig regel er jo ”maks 7±2” objekter pr graf – kanskje ni ved enkle relasjoner, men ned mot fem ved komplekse.

Det er ingen notasjon for å angi sammenheng mellom diagrammene, som alle er på samme nivå. Det ene bruksmønsterdiagrammet er altså ikke et subdiagram til et annet (meta)diagram. Man kan velge hvordan man vil inndele diagrammene:

- ✓ Tegne en aktør bare på ett diagram. Alle bruksmønstre denne aktøren er interessert i tegnes inn der (hvis det er mulig – det kan jo være mange). Bruksmønstre må da ofte gjentas på flere diagrammer.
- ✓ Tegne et bruksmønster bare på ett diagram. Alle aktørene som er interessert i dette bruksmønsteret tegnes da inn der (hvis det er mulig). Aktører må da ofte gjentas på flere diagrammer.
- ✓ Forsøke å dele systemet opp i ”naturlige deler” og tegne alle aktører og bruksmønstre som tilhører en gitt del inn på samme diagrammer.

Det er sikkert andre strategier som jeg ikke har tenkt på. Poenget er at man tislutt får overblikk.

Detaljering med tekst

Det er nødvendig å spesifisere aktører, bruksmønstre og relasjoner ytterligere. Hensikten er da ikke å ta beslutninger om realisering, men å gi ytterligere forståelse. Det vanligste er å gjøre dette ved å knytte en vanlig tekst til forholdet, f.eks. ”Det er rektor som skal ha vitnemålene, for kontroll og underskrift.” eller ”Kontroll av kredittkort gjelder bare nye kunder, men gjøres også årlig for alle faste kunder”.

Til beskrivelse av bruksmønstre, brukes gjerne *maler*. Her er et eksempel på en slik mal, hentet fra Rational Unified Process (RUP):

Use Case xxx:

Overview:

Notes:

Actors:

Preconditions:

Scenario:

No.	Action (Stimulus)	Software Reaction
1		
2		
3		

Scenario Notes:

Post Conditions:

Exceptions:

Required GUI and GUI Sketches:

Dependencies and Relations:

Her er et eksempel på et scenario, skrevet med syntaks etter Cockburn[2]:

[2] Alistair Cockburn: ”Writing Effective Use Cases”, Addison-Wesley, 2001, ISBN 0-201-70225-8

Prebetingelse
Automaten viser teksten ”Sett inn kort”

Hovedscenario (suksess)
1) Sjåføren drar kortet sitt i parkeringsautomaten
2) Automaten sjekker at kortet er gyldig
3) Sjåføren angir parkeringstid
4) Automaten viser utløpstid
5) Sjåføren trykker knappen ”Billett”
6) Automaten usteder billett
7) Bruksmønsteret avslutes med suksess

Utvidelser (unntak)
2a) Kortet er ugyldig
- 2a1) Automaten viser teksten ”Ugyldig kort” i 10 sekunder
- 2a2) Automaten viser teksten ”Sett inn kort”
- 2a3) Bruksmønsteret avsluttes

2b) Kortet er uleselig
- 2b1) Automaten viser teksten ”Uleselig kort” i 10 sekunder
- 2b2) Automaten viser teksten ”Sett inn kort”
- 2b2) Bruksmønsteret avsluttes

5a Sjåføren trykker knappen ”Avbryt
- 5a1) Automaten viser teksten ”Avbrutt”
- 5a2) Bruksmønsteret avsluttes

Postbetingelse
Minimum: Automaten viser teksten ”Sett inn kort”
Suksess: Automaten viser teksten ”Sett inn kort” og har utstedt billett.

Bruksmønstre kan også detaljeres med aktivitetsdiagrammer (”activity diagrams”). De likner på (men er ikke helt som) de gamle flytdiagrammer (”flow charts”). Jeg går ikke nærmere inn på dem her. *Overhead*

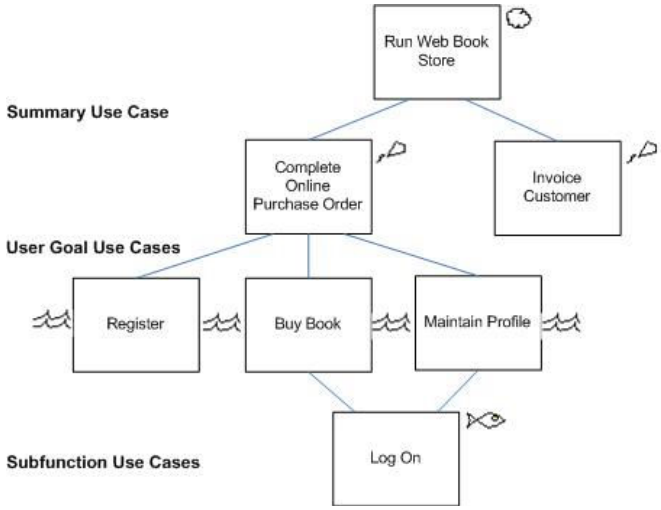
Pakking

Det er mulig å ”pakke” inn ett eller flere bruksmønstre i en ”pakke”. Dette gjøres kun for å lette oversikten, og gir fortsatt ingen prinsipiell nivåinndeling. Jeg velger å ikke ta med dette her.

Oversikter over bruksmønsterdiagrammene

Når det er sammenheng mellom bruksmønsterdiagrammer, kan det være aktuelt med en egen graf som viser sammenhengene. Her er et eksempel på syntaks fra Oracle.

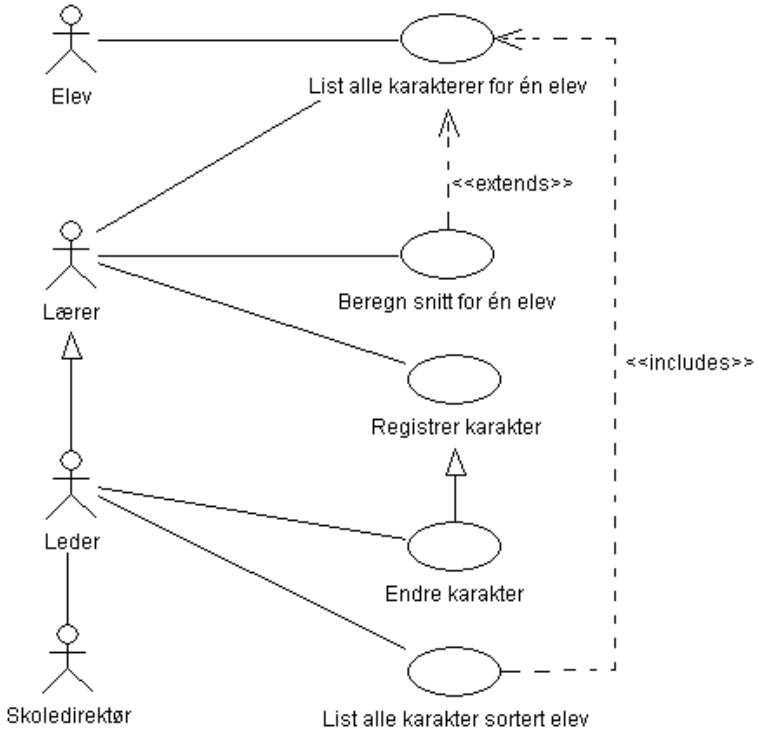
Grunnivået er brukernes nivå, det er her de får beskrevet hvilken funksjonalitet de ønsker at systemet skal ha. (Merk at eksemplet har med ”Log On” som bruksmønster – det er diskutert ovenfor.)



Bruksmønsterdiagram i praksis

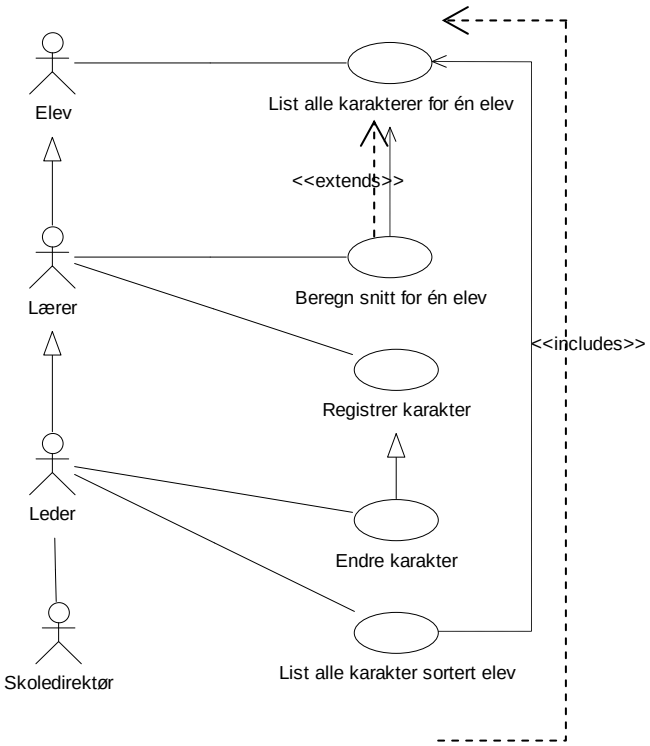
Et eksempel

Overhead



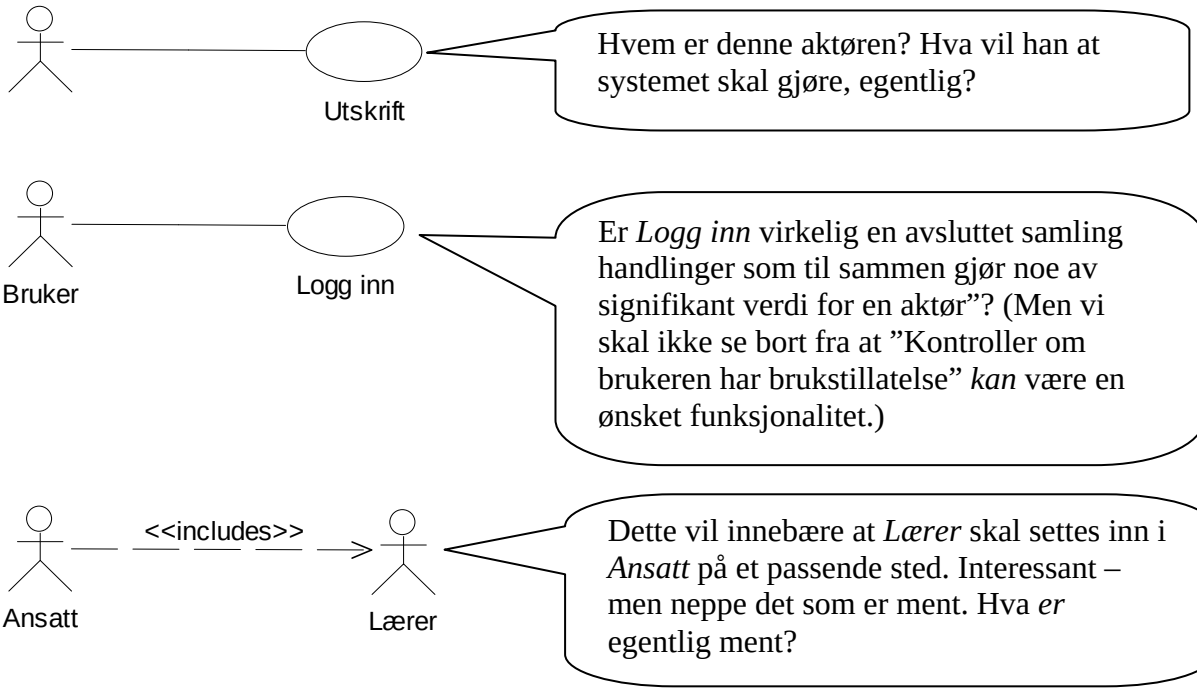
Note: I eksemplet ovenfor er det tatt med alle mulige sammenhenger for vise all bruk. Det vanlige er at bruksmønsterdiagrammer bare består av aktører, bruksmønstre og assosiasjoner mellom aktørene og bruksmønstrene. Andre relasjoner er uvanlige og bør benyttes med stor forsiktighet. Alistair Cockburn mener man ikke bør bruke ”extends”, bare ”include” som han sammenlikner med et vanlig subrutinekall. ”Extends” *kan* innebære at UC 2 selv vet når den skal utvide UC 1. Det passer bare for systemer med flere, parallelle løp (prosesser/tråder).

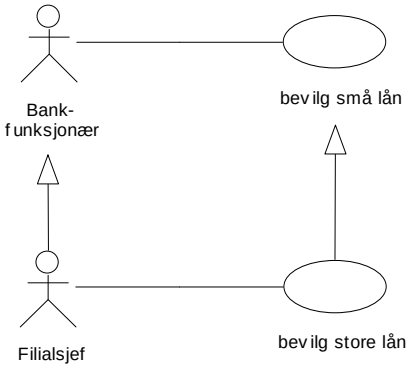
Assosiasjonen *Lærer⇳List alle karakterer for én elev*, kan erstattes av arv *Lærer⇨Elev*:
Overhead



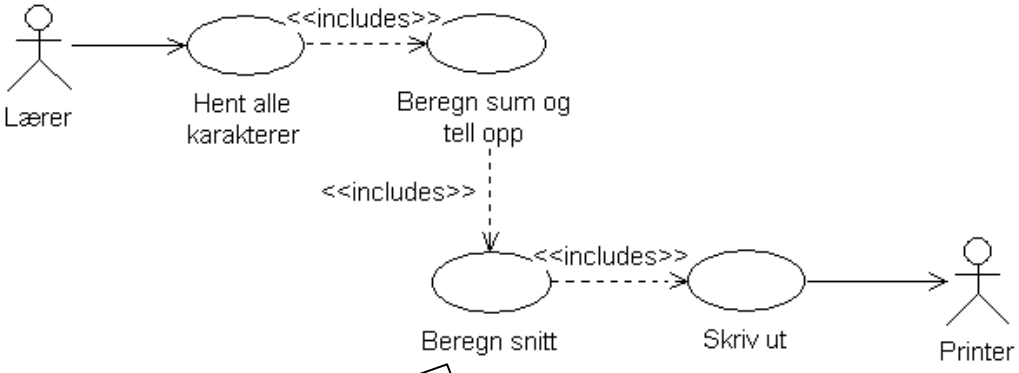
Noen typiske feil

Nybegynnere (f.eks. studenter) gjør enkelte typiske feil når de tegner bruksmønsterdiagrammer. Her er noen eksempler:
Overhead





Meningen er nok at bankfunksjonær kan bevilge små lån, men bare filialsjef kan bevilge store lån. Da er dette feil!
Siden "bevilg store lån" er en spesialvariant av "bevilg små lån", kan bankfunksjonær utføre *begge* (som i vanlig polymorfisme).



Dette er *helt* galt, men sees ofte fra studenthold. Tegneren tenker seg nok en *sekvens av handlinger* (først hentes alle karakterene, så beregnes sum og antallet telles opp, så…). *Printer* er ikke en aktør – den ønsker ikke funksjonalitet fra systemet. Enveis assosiasjoner (f.eks. *Lærer⇨Hent alle karakterer*) er ikke tillatt i bruksmønster-diagrammer (men brukes av noen – se fotnote side 5) – alle assosiasjoner er *toveis* (aktøren ønsker bruksmønsteret og bruksmønsteret er ønsket av aktøren). Dessuten innebærer <<includes>> at det ene bruksmønsteret *inkluderer* det andre, ikke at det overfører kontrollen til det. Ifølge dette diagrammet vil altså f.eks. *Beregn snitt* inkludere *Skriv ut* og det er neppe det som er ment.

Antagelig mener systemereren her helt enkelt at *Lærer* ønsker bruksmønsteret *Skriv karakterstatistikk* eller liknende. *Hvordan* det skal gjøres er temmelig trivielt, men kan eventuelt spesifiseres med tekst eller aktivitetsdiagram, og ikke i bruksmønster-diagrammet.

Seminaroppgave i fellesskap

Oppgave SOS trykket
Løsningsforslag SOS trykket/overhead/tavle

Sammenhengen mellom bruksmønster- og det logiske perspektiv

Bruksmønsterperspektivet er svært sentralt i UML. Det vil føre for langt her å gå inn på alle sammenhenger mellom perspektivene. Sammenhengen til det logiske perspektivet er imidlertid antakelig det viktigste, og om denne sammenhengen vil jeg derfor trekke frem noen få momenter knyttet til overgangen mellom bruksmønsterperspektivet og det logiske perspektivet.

I den logiske analysen, skal den statiske strukturen dokumenteres i klasse- og objektdiagrammer. Det viser hvilke klasser/objekter man tenker seg og hvilke relasjoner det er mellom dem. Det dynamiske aspektet – kalt oppførsel (”system behaviour”) – dokumenteres med samarbeidsdiagrammer (”collaboration diagrams”), sekvensdiagrammer (”sequence diagrams”), tilstandsdiagrammer (”statechart diagrams”) og aktivitetsdiagram (”activity diagrams”). Alle diagrammer kan suppleres med diverse tilleggsdokumentasjon etter behov, f.eks. andre figurer, blanketter, tekster og programkoder.

Det logiske perspektivet dokumenterer systemet innenfor systemgrensen, og det må naturligvis passe med dets utside som er dokumentert med bruksmønsterdiagrammer. Vi vil forvente at systemets deler – i samarbeid – kan utføre alle bruksmønstrene.

Videre vil vi forvente at bruksmønsterperspektivets aktører, er representert ved klasse/objekter innenfor systemet (som grenseklasser, ”boundary classes”).

Bruksmønstrene representerer *funksjonalitet*, men ofte vil funksjonalitet implisitt kreve datalagring. F.eks. vil bruksmønsteret *”Lage karakterliste”* kreve at systemet lagrer karakterer og andre opplysninger som skal være med på listen. Vi forventer at det logiske perspektivet tar høyde for det.

Hvis man vil være systematisk, kan man lage det logiske perspektivet (klassediagrammet) ”bruksmønsterdrevet” (”use case driven”) eller ”entitetsdrevet” (”entity class driven”):

- ✓ **Bruksmønsterdrevet logisk analyse** innebærer at man tar utgangspunkt i bruksmønstrene og lager klasser/objekter med metoder som kan utføre deler av bruksmønsteret. Gjennom samarbeid skal de kunne utføre hele bruksmønsteret. Data (attributter) plasseres deretter i samme klasse/objekt som de metodene som har bruk for dem.
- ✓ **Entitetsdrevet logisk analyse** starter med dataene og plasserer dem i entitetsklasser/-objekter i naturlige grupperinger. Dette likner svært på datamodellering. Senere plasseres metoder som kan bidra til bruksmønstrene sammen med de dataene de benytter.

Man kan forvente at bruksmønsterdrevet analyse vil gi mer effektive og enklere metoder, mens entitetsdrevet gir bedre datastrukturer. Enkelte forfattere anbefaler bruksmønsterdrevet analyse som hovedregel, mens jeg mener man bør analysere datakomplekse systemer entitetsdrevet og algoritmekomplekse systemer bruksmønsterdrevet. Tekniske systemer (der maskinenheter samarbeider i reell tid – ”RT-systemer) egner seg for bruksmønsterdrevet analyse, tror jeg.

Drøfting

Hva beskrives ikke i bruksmønsterperspektivet

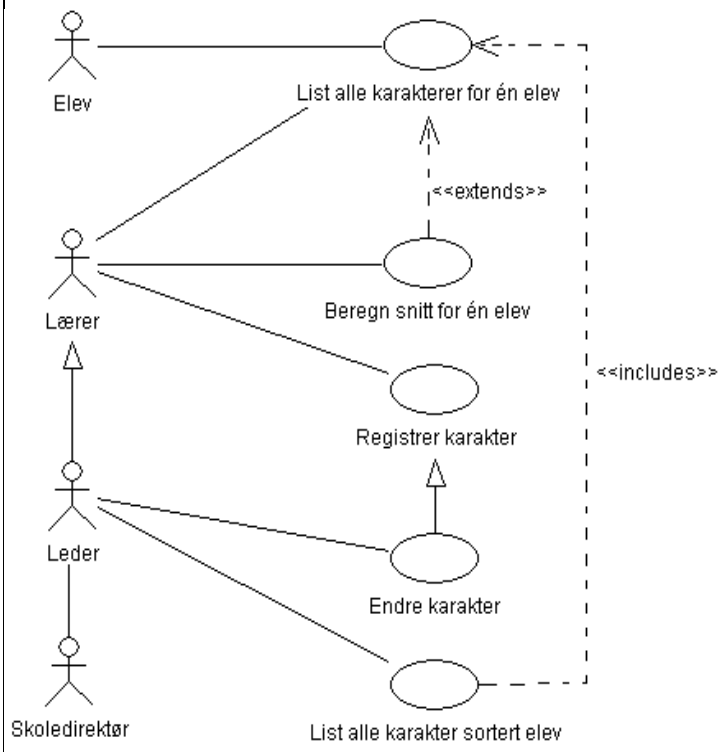
Et viktig aspekt når vi ser et system ”fra utsiden”, er hvordan samarbeidet foregår mellom aktørene og systemet, altså **grensesnittet** mellom dem.

Hvis aktøren er et annet *system*, vil det dreie seg om en (temmelig teknisk) protokoll (= en mengde formaliserte meldinger og signaler). Den vanlige dokumentasjonsformen for slikt, er tekster.

Hvis derimot aktøren er et *menneske* (en *bruker*) vil det være behov for å spesifisere MMI (”Man Machine Interface” = Menneske maskin interaksjon) evt. GUI (”Graphical User Interface” = grafisk brukergrensesnitt). UML har ingen dokumentasjonsform for dette, men anbefaler at man lager prototyper. Sannsynligvis vil det være fornuftig tidlig i utviklingen å lage prototyper av typen ”mock ups”, altså prototyper bare med grensesnitt og uten funksjonalitet.
Evt. demo mock up SOS.exe

Fordeler og ulemper med bruksmønsterdiagrammer

Bruksmønsterdiagrammer kan sees som en funksjonsbeskrivelse, som utgjør en del av kravspesifikasjonen. Slike funksjonsbeskrivelser kan vi også enkelt lage som lister:
Overhead



Elev ønsker:
1) List alle karakterer for én elev – benytter om nødvendig
   a) Beregn snitt for én elev

Lærer ønsker:
1) List alle karakterer for én elev
2) Beregn snitt for én elev – benytter om nødvendig
   a) Beregn snitt for én elev
3) Registrer karakter

Leder ønsker:
1) Det samme som **Lærer** med tillegg av:
   a) Endre karakter – variant av
      i) Registrer karakter
2) List alle karakterer sortert elev – benytter alltid
   a) List alle karakterer for én elev – benytter om nødvendig
      i) Beregn snitt for én elev

Det har imidlertid en del fordeler å benytte grafer:
1) Det er mer visuelt. Mange oppfatter og jobber bedre med bilder enn med tekst.

2) Grafer inviterer til kreativitet. Det er viktig å få alle med på det skapende arbeidet som kreves i analysearbeid.
3) Grafer kan gi bedre overblikk. Særlig kommer sammenhenger ofte bedre frem.
4) Grafer tvinger frem en bestemt måte å tenke på. Språket i en graf har svært få symboler (”ord”) og det som kan uttrykkes er begrenset. Dette bidrar til å konsentrere arbeidet om noen få aspekter ved systemet. Dette kan være begrensende, men vanligvis er systemene så komplekse at det er god strategi å dele opp problemet og ”se på en ting av gangen”.
5) Grafer er presise. Det er meget enklere å være uklar, vag og usikker i en tekst enn i en graf. Dette tvinger frem en diskusjon om ”hvordan det egentlig er”.

Men grafer har også ulemper.
1) Grafene har sin egen syntaks som må kunnes.
2) Syntaksen kan komme i veien for arbeidet. Man vet hva man vil si, men finner det vanskelig å få uttrykt det.
3) Grafer krever i praksis et tegneprogram – helst ett som har syntaksen programmert inn. Grafingen krever da også at man behersker tegneprogrammet, og at programmet har tilstrekkelig kvalitet. Det er vanligvis enklere – og billigere – å benytte tekstbehandling, og man unngår versjonsproblematikk.

Det er ikke uten videre gitt at utviklingsarbeidet bør begynne med bruksmønsterperspektivet. (Andre utviklingsmetoder har begynt med dataanalyse, med virksomhetsanalyse, rutineanalyse, organisasjonsanalyse og annet.)

Kritikk av UML

UML består av hele *åtte* diagramteknikker, men i realiteten er det mange flere fordi flere av dem har varianter. I tillegg kommer andre spesifikasjoner som DDL, tekster o.l. Det er sammenhenger mellom alle teknikkene, og man må passe på at spesifikasjonen er indre konsistent. Det er altså ikke enkelt å lære seg alt dette, og for en ”vanlig *bruker*” er det i virkeligheten umulig (en bruker som kan alt dette er dessuten ikke lenger en ”vanlig bruker”). Selv i UMLs egen dokumentasjon er ikke alltid sammenhengene mellom dokumentasjonsformene klar.

Dette åpner for *modellmakt* ved at IT-ekspertene anvender et språk og modeller som brukerne ikke har forutsetninger for å forstå. Brukernes innflytelse blir derved sterkt redusert, hvilket er problematisk i skandinavisk sammenheng. Jeg tror egentlig ikke det er realistisk at brukerne er med på annet enn bruksmønstrene, deretter ”etterlates de i støvet”. Heldigvis er bruksmønsterperspektivet svært sentralt i UML, men likevel vil mange beslutninger som tas senere påvirke systemet sterkt – også på måter som brukerne merker.

Det kan være problematisk at hele analysen forutsetter at systemet skal realiseres med edb, og at det skal være objektorientert. Det er slett ikke sikkert at det er den beste, eller eneste, løsningen. Kanskje skal bare deler av systemet realiseres med edb, kanskje bare deler av det igjen med OOP/OODBMS. Når systemet først er analysert objektorientert, er det imidlertid ikke alltid trivielt å realisere det med andre edb-verktøy. Det blir omtrent som å bygge et hus i betong, når det er tegnet i tre: Umulig er det ikke, men det krever betydelige tilpasninger (i IS-utvikling ofte kalt ”mapping”) og kanskje noen mer dokumentasjon underveis. F.eks. vet vi at de fleste realiserer datalagringen med relasjonsdatabase – allikevel forutsetter UMLs logiske perspektiv at det skal benyttes en objektorientert database.

Det er en ulempe at det ikke finnes noen ”offisiell” metode knyttet til UML. Det er ikke tilstrekkelig å kjenne et stort antall diagramteknikker – det gjorde vi fra før – man bør også få vite hvordan de er tenkt brukt. Hvilken rekkefølge jobber man i, hvem deltar osv? Bare med en slik metode er det f.eks. mulig å planlegge prosjektet. Fordelen med en ”offisiell” metode, er at den utvikles gjennom forskning og erfaringsutveksling. Rationals metode RUP tilfredstiller ikke kravene til åpenhet.

Når UML likevel er en ”vinner” – for det er den – skyldes det mest de tunge aktørene som står bak, med stor markedsmakt. Dessuten var mange lei av ”the Method War”. Videre er det en fordel for adapsjonen at UML er omfattende (med dokumentasjonsformer for det meste) og fleksibel. Man kan i stor grad bruke diagrammene som man vil og legge til egne notasjonsstandarder. Dermed står man sjelden fast når noe skal dokumenteres.

UML er virkelig objektorientert. Mange teknikker og metode som ble lansert under ”the Method War” kalte seg objektorienterte, man var det egentlig ikke. Problemet er at ordet ”objekt” egentlig bar betyr ”ting” og *alle* kan jo hevde at de analyserer ”ting”. Objektorientert som i ”Objektorientert Programmering” var det slett ikke alle som var, eller de tok bare for seg *noen* av de nødvendige perspektivene. Og at ”objektorientert” er et in-ord, kan ingen betvile.

Oslo, februar 2003/september 2007 ***Knut W. Hansson***