NTNU
Det skapende universitet

ProtoMODL: En hybrid notasjon for å beskrive interaksjon i grafiske brukergrensesnitt

Anders Eivind Bråten

Master i informatikk
Oppgaven levert: Juni 2006
Hovedveileder: Dag Svanæs, IDI
Medveileder(e): Hallvard Trætteberg, IDI

Norges teknisk-naturvitenskapelige universitet
Institutt for datateknikk og informasjonsvitenskap

Sammendrag

Innenfor tradisjonell systemutvikling, der hovedfokuset har ligget på systemenes funksjonalitet, har UML og andre modellbaserte utviklingsmetoder hatt stor suksess. Til tross for mye forskning de siste to tiårene, har slike metoder imidlertid ikke fått særlig aksept som et verktøy for å designe systemenes brukergrensesnitt. [Myers, et. al., 2000] En av grunnene til dette er at de har vært vanskelige å benytte i kombinasjon med bruker-sentrerte designmetoder. De fleste konstruksjonsmetoder har, ved å kreve interaksjonsmodeller før den konkrete designutviklingen tar til, en ovenfra og ned tilnærming til brukergrensesnittdesign. Iterative designmetoder nærmer seg som regel problemet fra den andre retningen, ved å ta utgangspunkt i en designide som så prototypes. [Trætteberg, 2002:1]

Innenfor brukersentrert design er det lagt stor vekt på utvikling av prototyper, spesielt tidlig i designprosessen. Slike utforskende prototyper er billige, raske å lage og gir mye informasjon om den foreslåtte designløsningen, brukerens aksept av denne, og hvilke feil og mangler som må utbedres i en senere iterasjon. [Beaudouin-Lafon et. al., 2003] En prototyp er en uformell og konkret artefakt, og i det ligger både dens styrke og svakheter. Interaksjonsprototyper kommuniserer sin hensikt på en intuitiv måte, men dette gjør det også vanskelig å dokumentere hvordan den interne interaksjonen foregår i brukergrensesnittet, og hvordan dette påvirker samhandlingen mellom brukeren og systemet.

Denne oppgaven adresserer disse problemene ved å utvikle en hybrid notasjon kalt ProtoMODL, basert på dialogmodellen DiaMODL og den kanonisk abstrakte prototypen. ProtoMODL gjør det mulig å kombinere, i en og samme representasjon, DiaMODL sin evne til å fremheve den underliggende interaksjonen i et brukergrensesnitt med den abstrakte prototypens evne til å presentere interaksjonen på en lettfattelig måte.

Brukskvaliteten til ProtoMODL har så blitt testet og sammenlignet med tre andre interaksjonsrepresentasjoner i et empirisk, kvalitativt eksperiment. Eksperimentet indikerer at notasjonen er rask å sette seg inn i og har relativt lav bruksterskel, men potensielt høy brukshøyde. Diagrammene ser ut til å ha høy dekningsgrad, fremstår som ryddige og lettleste og interaksjonen virker relativt presist beskrevet.

Ved å kombinere tankegangen bak tradisjonell systemutvikling med kreativ design åpnes det for en ny måte å betrakte designprosessen. Oppgaven viser at det er mulig å dra veksel på begge de to tilnærmingene når man skal utforske et designrom, spesifisere eller dokumentere interaksjonen som skjer innad i et brukergrensesnitt eller mellom et system og en bruker, eller kommunisere med andre interessenter sine tanker, ideer og forslag til interaksjonsløsninger.

Innhold

Figur- og tabelliste

Forord

Om oppgaven

Hensikten med hovedfagsoppgaven

Hensikten med hovedfagsoppgaven er å produsere et selvstendig vitenskapelig arbeid. I dette ligger det at man skal vise innsikt i aktuell forskning, og relevante teorier og metoder som har betydning for oppgavens problemstilling. [IDI, NTNU, Org:1, 200?]

Motivasjon

Bak denne oppgaven lå et ønske om å finne en måte å bedre kommunikasjonen mellom designere og utviklere, øke kvaliteten i den initielle designfasen hvor hurtig prototyping ofte er en viktig del av prosessen, og gi muligheten til å entydig dokumentere interaksjonen i det ferdige brukergrensesnittet.

Hovedbidrag

Hovedfagsoppgaven har resultert i to produkter: Det ene er en hybrid notasjon jeg har kalt ProtoMODL, basert på Larry Constantines kanoniske abstrakte prototyper og Hallvard Trættebergs DiaMODL. Det andre er et eksperiment som både undersøker brukskvaliteten og tester selve notasjonen til ProtoMODL.

Hvem er oppgaven myntet på

Oppgaven er først og fremst skrevet for designere, utviklere og andre fagpersoner innen feltene HCI, interaksjonsdesign, prototyping og systemutvikling.

Organisering av oppgaven

Hovedfagsoppgaven er satt opp slik at først presenteres motivasjonen for å skrive den og hypotesen som ligger i grunn, deretter gjennomgås grunnlagsstoffet, før notasjonen til ProtoMODL beskrives. Så kommer eksperimentet som tester hypotesen og evaluerer resultatet. Det anbefales å lese oppgaven i rekkefølge.

Innledning

I kapittel 1 beskriver jeg bakgrunnen for oppgaven og motivasjonen for å skrive den. Det diskuteres også en del grunnleggende problemstillinger som har vært viktig å ta i betrakning under utarbeidelsen av dette arbeidet.

Teori og State of the art

I kapittel 2 undersøker jeg hva interaksjonsdesign egentlig handler om, hvilke roller en designprosess har, hvilke forskjeller som finnes mellom tradisjonell systemutvikling og kreativ design og hva vi mener med en designrepresentasjon og hvilke egenskaper den har.

I kapittel 3 tar jeg for meg noen forskjellige interaksjonsmodeller, hva som menes med interaksjonsmodellering og hva vi faktisk ønsker å modellere.

I kapittel 4 ser jeg på hva som kjennetegner interaksjonsprototyper, og hvordan de brukes i interaksjonsdesign.

Hypotesebygging og utvikling av modell

I kapittel 5 diskuterer jeg hvilke kriterier som må ligge til grunn når jeg skal lage en hybrid modell av en dialogmodell og en abstrakt prototyp. Ut fra disse kriteriene, de aktuelle problemstillingene og innsikt i den teori som allerede eksisterer på dette området setter jeg så opp en falsifiserbar hypotese. Deretter beskriver jeg min hybridnotasjon, ProtoMODL, som tar sikte på å føre de beste egenskapene fra DiaMODL og den kanonisk abstrakte prototypen sammen i en ny , hybrid modell.

Eksperiment

I kapittel 6 tester jeg hypotesen ved hjelp av et empirisk, kvalitativt eksperiment som sammenligner ProtoMODL med tre andre representasjoner som også beskriver interaksjonen i et brukergrensesnitt. Eksperimentet undersøker samtidig i hvilken grad ProtoMODL er i stand til å gi informasjon om interaksjonen i et brukergrensesnitt, om tvetydigheter kan avklares ved denne representasjonen og om andre representasjonsformer er bedre egnet til oppgaven.

Evaluering

I kapittel 7 evaluerer jeg arbeidet denne hovedfagsoppgaven har resultert i og beskriver hvilket arbeid som gjenstår.

Begrensninger

Oppgaven vil konsentrere seg utelukkende om prosessen rundt brukergrensesnittdesign og mekanismene rundt dette. Underliggende domene- og objektmodeller vil bare bli belyst der hvor de har direkte betydning for utformingen av interaksjonen i et brukergrensesnitt. Systemarkitekturteori blir belyst i så liten grad som mulig.

Modellering av datasystemer er et enormt fagfelt. Jeg har avgrenset oppgaven til å omhandle ren interaksjonsmodellering samt noen modeller som grenser tett opptil dette temaet. Ideelt skulle ProtoMODL vært konstruert for å være kompatibel med UML (Unified Modeling Language), men dette har ikke vært prioritert. Språket bygger imidlertid i stor grad på DiaMODL som er laget nettopp med tanke på dette.

Når det gjelder temaet prototyping er dette et fagfelt det ikke er forsket like mye i. Ettersom litteraturen dermed er begrenset, og fagfeltet er svært stort, har jeg valgt å konsentrere meg om den typen som går under begrepet ‘papirprototyper’. Prototyper som kjører på maskin blir derfor bare belyst der deres egenskaper er sammenfallende med papirprototyper eller der det er viktig å understreke forskjellen mellom dem.

Design- og utviklingsprosesser er et tema jeg også skulle ønske hadde fått plass mellom disse permene. Temaet er såpass langt fra fokuset til oppgaven at det ikke virket naturlig å fordype seg i det. Det har det allikevel blitt plass til noen avsnitt om om emnet.

Det empiriske grunnlaget blir naturlig avgrenset av det begrensede antall personer som innehar nok kompetanse innenfor interaksjonsmodellering og abstrakt prototyping til å kunne være med å gjennomføre av eksperimentet.

Takksigelser

Det er mange som har bidratt med hjelp under skrivingen av denne oppgaven. Først og fremst vil jeg takke mine veiledere Dag Svanæs og Hallvard Trætteberg. Kirsten Sandersen og de andre i administrasjonen ved IDI har vært hjelpsomheten selv når jeg har spurt om noe. Og uten at testdeltakerene “A” og “B” hadde stilt opp ville ikke oppgaven hatt noe empirisk grunnlag å bygge på. Stor takk til dere!

Nils Jørgen Mittet har vært en svært dyktig og inspirerende sparringpartner i disse årene og skal selvsagt ikke glemmes. Harald Øverby kom med gode råd i når jeg trengte det mest og fortjener stor ære for det. Til slutt må jeg rette en stor takk til min kone Oddrun som har støttet meg hele veien og min sønn Brage som har vært en stor kilde til glede og inspirasjon.

Definisjoner og ordforklaringer

Her presiseres hva som menes med en del spesielle ord og begreper som benyttes i oppgaven.

abstrahere	Prosessen med å fjerne (unødvendig) informasjon.
abstrakt	Ikke konkret. En representasjon som inneholder abstrakt informasjon uttrykker *ideen* om systemet og ikke det konkrete systemet slik som det eksisterer i tid og rom.
aktør	Enhver person, objekt eller komponent som påvirker interaksjonen i et system
artefakt	Enhver representasjon av et system, fop eksempel prototyper, modeller og spesifikasjoner.
aspekt	En karakteristikk som skal undersøkes
brukergrensesnittkomponent, komponent	Ethvert element i brukergrensesnittet som har i oppgave å presentere informasjon til brukeren og/eller motta input fra brukeren. Også: En widget.
datasystem, system	Et fullstendig datasystem inkludert brukergrensesnitt og underliggende funksjonalitet.
design	En konkret eller abstrakt ide eller representasjon av et brukergrensesnitt.

designe	Prosessen med å utvikle et design
designer	Enhver person som lager et design, uansett jobbtittel.
diagram	En skjematisk tegning som viser hvordan en underliggende mekanisme til et brukergrensesnitt eller system fungerer.
GUI	*Graphical User Interface* - grafisk brukergrensesnitt.
interaksjon	Samspillet mellom aktører i et system. Avhengig av konteksten kan en aktør være en bruker, systemet selv, en del av et system, et objekt eller en brukergrensesnittkomponent.
interaktor	Representasjon av en brukergrensesnittkomponent.
interessent	Enhver person eller gruppe som har et mål med det ferdige systemet og/eller bruken av det.
kanonisk	Ensartet
konkret	Håndfast, noe som framstår slik som vi sanser det.
MMI	*Menneske-maskin-interaksjon.* Studiet av hvordan mennesker og maskiner kommuniserer med hverandre.
modell	En, som regel abstrakt, representasjon av et system som beskriver et eller flere aspekter ved funksjonaliteten eller virkemåten til systemet. En modell kan bestå av et eller flere diagrammer.
OO	*Objektorientering*. En metode for å utvikle programvare
prototyp	En, som regel konkret, representasjon av et framtidig eller eksisterende datasystems GUI. Om prototyper brukt for å teste et eksisterende eller framtidig datasystems funksjonalitet vil benevnelsen ‘funksjonell prototyp’ bli brukt.
representasjon	Noe som beskriver eller fanger essensen av et eller flere aspekter av noe konkret.
utvikler	Enhver person som lager funksjonaliteten til et system, uansett jobbtittel.
UML	*Unified Modeling Language*. Standard for modellering av programvareartefakter.
validere (validate)	Undersøke at noe er (logisk) gyldig
verifisere (to make ascertain)	Bekrefte; undersøke og fastslå riktigheten av
verktøy	Noe som hjelper en interessent med å lage en artefakt.
widget	Ferdiglagde basiselementer som benyttes når man designer et grafisk brukergrensesnitt. Som regel samlet i et bibliotek eller applikasjonsrammeverk

Kapittel 1

Innledning

Dette kapittelet beskriver bakgrunnen for denne oppgaven. Det tar opp en del grunnleggende problemstillinger som det har vært viktig å ta i betrakning under utarbeidelsen av oppgaven og diskuterer disse.

1.1 Bakgrunn for oppgaven

Systemarbeid og Menneske-Maskin Interaksjon

Main Entry: **in·ter·ac·tion**
Pronunciation: "in-t&-'rak-sh&n
Function: *noun*
: mutual or reciprocal action or influence

FIGURE 1-1: Fra Webster Online Dictionary: 'in-ter-ac-tion'

Under arbeidet med denne oppgaven har jeg vært tilknyttet fagruppe for Systemarbeid og menneske-maskin-interaksjon , en gruppe tilknyttet Institutt for Datateknikk og Informasjonsvitenskap ved Norges Teknisk-naturvitenskapelige Universitet.

Systemarbeid tar for seg det vitenskapelige perspektivet om hvordan man utvikler og forvalter datasystemer, og hvilke arbeidsmåter og metoder som benyttes i dette arbeidet. Feltet er dypt forankret i tradisjonen rundt ingeniørarbeid [Trætteberg, 2002:1], og fokuserer på datasystemers funksjonalitet og deres evne til å utføre de oppgavene som er pålagt dem. Den tradisjonelle systemutviklingen tar utgangspunkt i nøyaktige kravspesifikasjoner og argumenterer for bruk av modeller når problemområdet skal utforskes og analyseres.

Menneske-maskin interaksjon (MMI) er en akademisk gren innenfor informatikk som fokuserer på hvordan man kan øke brukbarheten til interaktive datasystemer ved å sette brukeren i sentrum. Det er et flerdisiplinært fagfelt med røtter i systemutvikling, kognitiv psykologi og ergonomi, som ved å kombinere elementer fra vitenskap, ingeniørarbeid og kreativ design tar for seg design, evaluering og imple-

mentering av datasystemer og studiene rundt dette. [Beaudouin-Lafon et. al., 2003] [Preece et. al., 2002] [Marion, Nettside:1, 200?] [Wikipedia, Org:4, 2006] Forskning innenfor MMI har vært svært vellykket, og har endret måten vi bruker datamaskiner fundamentalt. [van der Aalst, 1998] MMI legger vekt på at all systemutvikling skal være iterativ og inkrementell. Dette betyr at den initielle kravspesifikasjonen ikke er av like stor betydning som ui tradisjonell systemutvikoing, siden mange av kravene blir til ettersom prosessen går sin gang.

Siden denne oppgaven prøver å kombinere metoder brukt innenfor disse to feltene, legger den seg i krysningspunktet mellom systemarbeid og MMI.

MMI og OO

Flere workshops holdt på MMI og OO konferanser (bl.a. CHI og ECOOP), har siden 1997 diskutert rollen til objektmodeller og oppgave/prosessanalyse i brukergrensesnittdesign. Under disse konferansene ble det understreket, både fra industrielt og akademisk hold, viktigheten av å bygge bro mellom tradisjonell programvareutvikling og menneske-maskin interaksjon. Et generelt rammeverk for å illustrere en objektorientert konseptuell arkitektur for interaktive systemer ble diskutret frem under CHI'97. [Nunes et. al., 2000]

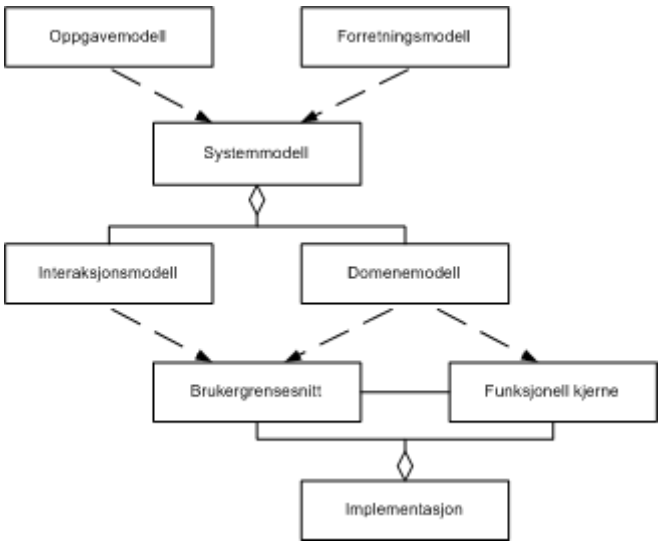


FIGURE 1-2: Wisdom '99 varianten av CHI '97 rammeverket [Nunes et. al., 2000]

Denne konseptuelle modellen skiller logisk, om ikke nødvendigvis fysisk, den funksjonelle kjernen fra brukergrensesnittet. Modellen viser at mens brukergrensesnittet baserer seg på både interaksjonsmodellen og domenemodellen, er den funksjonelle kjernen kun basert på domenemodellen. Denne modellen støtter også separering av datapresentasjonen i brukergrensesnittet fra den konseptuelle spesifiseringen av dialogen som skjer i interaksjonsmodellen. [Nunes et. al., 2000]

Egne erfaringer

Grunnlaget for denne oppgaven ble lagt i den tiden jeg jobbet som utvikler i en mellomstor bedrift rettet mot webapplikasjonsmarkedet. Som i mange andre fir-

maer i dette markedet var utviklingsavdelingen delt opp i utviklere som hadde ansvaret for å utvikle funksjonalitet og designere som hadde ansvaret for å lage selve brukergrensesnittet. Kommunikasjonen mellom utviklere og designere var uformell. Stort sett hadde utviklerne bare direkte kontakt med brukere eller kunder i den initielle fasen av et prosjekt. Mitt inntrykk var at dette i stor grad også gjaldt designerne.

Jeg og mine kolleger ble ofte bedt om å lage nye webapplikasjoner og portaler eller modifisere eksisterende kode til en ny kunde. De fleste av disse utviklingsprosjektene ble gjennomført på en ad hoc måte: Formell dokumentasjon manglet, endringer som ble gjort underveis ble ikke dokumentert skikkelig og den dokumentasjon som ble produsert manglet ofte relevans til hverandre eller var vanskelig å kryssreferere. Modellering og systemering ble sjelden utført, og da kun i starten av et utviklingsprosjekt. Som regel skjedde nyutvikling og endring gjennom uformell ad-hoc prototyping, der prototypene ble videreutviklet direkte til den endelige webapplikasjonen.

Bare unntaksvis ble nye prosjekter og endringer i gamle systemer gjenstand for grundig analyse og abstrahering i form av modeller eller. I disse få tilfellene var mitt inntrykk at designere, kunder og sluttbrukere, hadde liten kjennskap til mer abstrakte former for dokumentasjon, og liten motivasjon for å sette seg inn dette området. Modellene som ble laget av systemet ble derfor enten svært forenklede og overfladiske, men forståelig for alle interessenter, eller abstrakte og kompliserte, men bare forstått og benyttet av utviklerne.

Selv om det ofte ble diskutert over lunsjbordet hvordan økt fokus på modellering og dokumentasjon kunne gjøre utviklingsprosessen mer effektiv, produktet bedre og endringer enklere, gjorde tidsnød og manglende styring at det ble med diskusjonene. Når det gjaldt den delen av systemet som hadde med direkte interaksjon mot brukeren å gjøre, ble denne i beste fall spesifisert ut fra skjermbilder eller raske skisser, i verste fall ble brukergrensesnittet designet på sparket.

Systemene som var klare for levering til kunde ble dokumentert i form av skjermbilder eller skisser sammen med tekstlige beskrivelser av hvordan systemet håndtererte interaksjonen med brukeren. Denne dokumentasjonen ble som regel utført av personer som i svært liten grad hadde vært med på å utforme selve designen eller koden til systemene. Ofte hadde de heller liten utdanning eller erfaring innenfor disse to feltene.

Uformelle samtaler jeg har hatt med venner ansatt i andre firmaer spredt rundt i Norge har gjort lite for å avkrefte mistanken jeg har om at dette er et generelt problem. Problemet virker også å gjelde de fleste grener av systemutviklingsbransjen, ikke bare webutvikling.

Utgangspunkt for oppgaven

Våren 2002 leverte Hallvard Trætteberg sin doktoravhandling med tittelen “Model-based User Interface design”. I denne oppgaven lanserer Trætteberg DiaMODL, et dialogmodelleringsspråk basert på ‘Pisa interactor abstraction’ og ‘UML tilstand-skart’ som gjør det mulig å representere hvordan informasjonen flyter mellom bruker og system via et brukergrensesnitt. [Trætteberg, 2002:1] Målet med DiaMODL var å definere et språk som både kunne brukes som et praktisk design-verktøy og samtidig gjøre det enklere å lage verktøystøtte for å generere koden til interaktorbaserte brukergrensesnitt. Siden tilstandskart er en del av UML stand-arden, ble det antatt at en implementasjon basert på dette er realistisk og praktisk for bruk i industrien. [Trætteberg, 2002:1]

Starten på to oppgaver

Dette modelleringsspråket dannet grunnlaget for to oppgaver, denne og en skrevet av Nils Jørgen Mittet. Siden mye av den underliggende teorien er felles for disse oppgavene, og noen av de grunnleggende løsningene som er jobbet fram også er felles, er det derfor et poeng i presentere begge disse oppgavene kort.

Ettersom både Mittet og jeg hadde praktisk erfaring fra utviklingsbransjen, var det viktig for oss at arbeidet vi gjorde skulle ha praktisk betydning og kunne være til direkte nytte for designere og utviklere når de skal utarbeide et brukergrensesnitt. Vi startet med å ta for oss forskjellige motsetninger innen feltet interaksjonsdesign som burde belyses mer. Noen av disse var “brukergrensesnitt vs. system”, “abstrakt analyse vs. konkret konstruksjon” og “metode vs. verktøy”.

Matpapirmodellen

Etter en del forberedende runder, satte vi sammen en ide om en modell som repre-senterte tre lag av abstraksjon i brukergrensesnittet; det konkrete brukergrenses-nittet, klassestrukturen som beskriver det og dialogmodellen som er bindeleddet mellom disse to ytterkantene.

Den ene måten vi kunne representere dette på var å plassere det konkrete bruker-grensesnittet og den abstrakte objektmodellen side-ved-side, og tenke oss at DiaMODL ble tegnet oppå disse representasjonene. På denne måten ville vi koble de to representasjonene sammen. Dette kalte vi for ‘matpapirmodellen’. Matpair-modellen så vi for oss kunne benyttes til å: a) vise dataflyt mellom grafiske kompo-nenter, og mellom grafiske komponenter og funksjonelle objekter i systemlaget, b) påpeke metoder som styrer dataflyten innad i brukergrensesnittet, og mellom bruk-ergrensesnittet og den funksjonelle objektstrukturen, c) påpeke hvilke “eventhan-dlere” som aktiverer denne dataflyten og d) vise avhengighet (‘dependencies’) mellom brukergrensesnittobjekter, og mellom et brukergrensesnittobjekt og den funksjonelle objektstrukturen.

Når vi testet ut matpapirmodellen viste det seg at det bare var praktisk å benytte denne modellen på enkeltkomponenter. Så snart interaksjonen inneholdt flere brukergrensesnittkomponenter eller funksjonelle objekter ble det så å si umulig å beholde oversikten i diagrammet. Diagrammene ble også arealmessig veldig store.

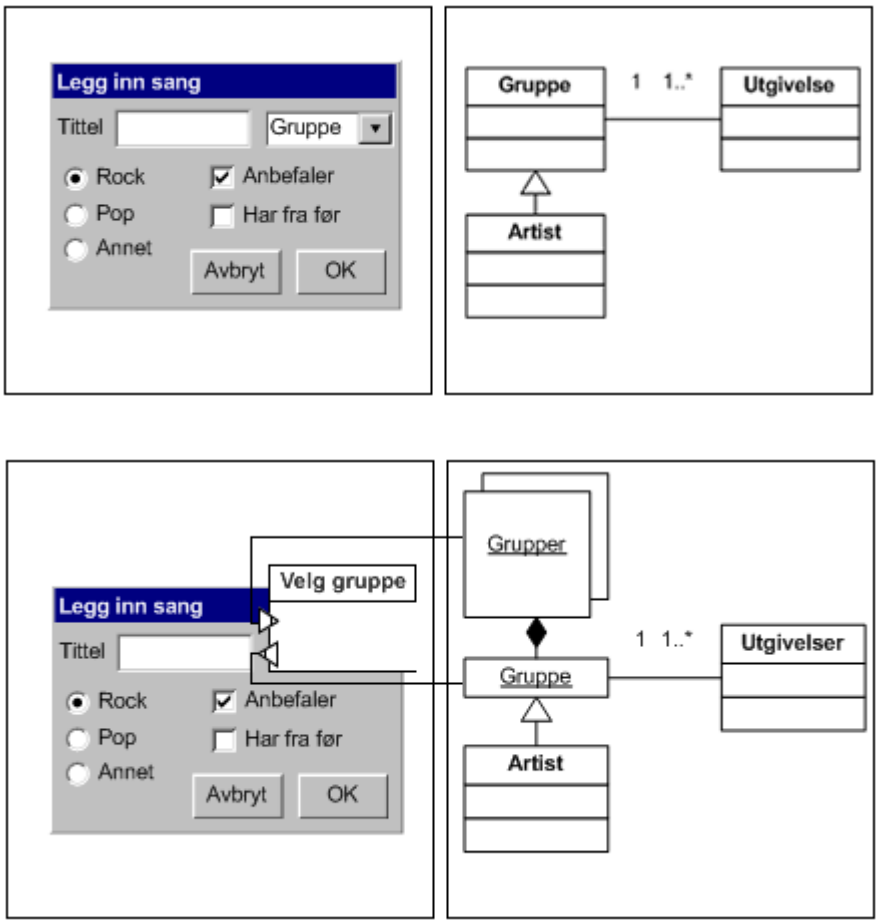


FIGURE 1-3: Matpapirmodellen (med og uten dialogmodell)

Sandwichmodellen

En annen løsning gikk ut på å lage en lagdelt modell, der det konkrete brukergrensesnittet og klassestrukturen representeres som hver sin "brødskive" og dialogmodellen er "smøret" som holder de sammen. I denne modellen er det ingen direkte utveksling mellom representasjonene. Hver representasjon undersøker dermed forskjellige aspekter av det samme domenet. Denne løsningen kalte vi Sandwichmodellen.

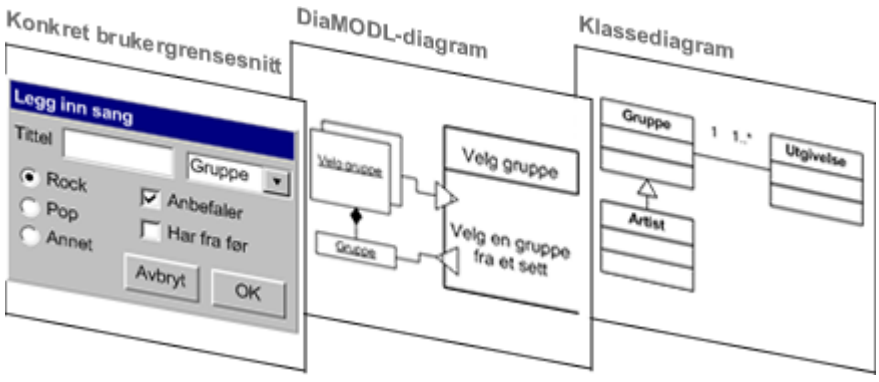


FIGURE 1-4: Sandwichmodellen

Vi så raskt at det ville kreve mye arbeid og forskning dersom vi for eksempel skulle lage verktøystøtte for dette. Vi valgte derfor å fokusere på hver vår side av dialogmodellen slik som den framstår i sandwichmodellen. Nils Jørgen Mittet beskriver i sin oppgave en løsning på hvordan man kan generere ferdig kode ut fra en modell laget i DiaMODL-notasjonen, og fyller på denne måten et hull i området mellom DiaMODL og objektmodellen. Hans oppgave ligger dermed nær systemsiden, og har sterk fokus på abstrakt analyse og verktøyutvikling.

Denne oppgaven

Denne oppgaven tar imidlertid for seg hvordan man kan lage en notasjon som gjør det enklere å overføre et konkret brukergrensesnitt til et DiaMODL-diagram og omvendt, og legger seg på denne måten mellom DiaMODL og det konkrete brukergrensesnittet. Den er rettet mot hvordan man kan benytte modellering i praktisk design, og fokuserer på fordelene med å benytte abstrakt analyse i konkret brukergrensesnittdesign og hvilke representasjoner som må utvikles for å få til dette.

Målet med oppgaven

Jeg har valgt å fokusere på det området som jeg personlig finner mest interessant, nemlig design av brukergrensesnitt og interaksjonen mellom brukeren og datamaskinen. Det jeg ønsker å finne ut av kan oppsummeres i et enkelt spørsmål:

Forskning starter med et spørsmål

"*Hvordan kan man på en enkel måte beskrive en interaksjonsprototyp, slik at den underliggende interaksjonen i brukergrensesnittet kan dokumenteres?*"

Problemet med uformell systemutvikling

Mange utviklere mener modellering ikke gir nok tilbake til at det lønner seg tidsmessig og resultatmessig. Den foretrukne måten å jobbe på blir derfor ofte å føre uformelle samtaler med de andre medlemmene i utviklingsteamet og kode applikasjonene direkte, altså tar prosessen form av 'uformell prototyping'. Andre aktuelle interessenter, spesielt designere og sluttbrukere, gir uttrykk for at formell dokumentasjon i form av modeller er kjedelig å lese og tung å sette seg inn i.

Problemet med uformell systemutvikling er først og fremst at det i ettertid er vanskelig å gå tilbake i prosessen og se hva som faktisk har blitt gjort, når det har blitt gjort og hvem som har hatt ansvaret for å gjøre det. Dette fører til problemer dersom man må gjøre endringer i systemet. Også ved utskiftninger av medlemmer i utviklingsteamet vil ofte manglende formell dokumentasjon kunne føre til at det blir vanskeligere for nye medlemmer å sette seg inn i prosessen og koden til systemet.

Formalisering av prototypingsprosessen

Siden det skal mye til å endre de holdningene som finnes, og prototyping ser ut til å være den foretrukne metoden å utvikle systemer på, har denne oppgaven valgt å angripe problemet med uformell systemutvikling fra en annen vinkel; nemlig å se på hvordan man ved å lage en hybrid modell kan formalisere *prototypingsprosessen* og dermed kvalitetssikre utviklingsprosessen.

Målet med oppgaven blir derfor å utvikle en notasjon som gjør det mulig å beskrive interaksjonen i en interaksjonsprototyp på en utvetydig måte og med en notasjon som er relativ enkel for ikke-tekniske interessenter både å tolke og bruke.

Denne notasjonen kan da bli et hjelpemiddel for utviklere og designere som kan brukes til å avdekke uklarheter og tvetydigheter i et design, spesifisere og kommunisere interaksjon når brukergrensesnittet skal designes og implementeres, og dokumentere interaksjonen i et ferdig brukergrensesnitt på en entydig måte.

1.2 Grunnleggende problemstillinger

Rollen til en designrepresentasjon

Å designe informasjonssystemer av en viss størrelse, er en komplisert oppgave hvor et bra resultat krever samarbeid og god kommunikasjon både internt mellom medlemmene i utviklingsteamet og mellom utviklingsteamet og sluttbrukere. Dette krever at alle er i stand til å forstå de designrepresentasjoner som benyttes.

Generelt kan man si at brukergrensesnitt enten blir *konstruert* eller *designet*. I begge tilfeller er det behov for å uttrykke designen visuelt ved hjelp av en eller annen form for representasjon, både for å kunne analysere løsninger og kommunisere sine ideer til andre. Det er imidlertid sånn at de representasjoner som blir benyttet av den konstruerende retningen spiller en forskjellig *rolle* enn de som benyttes av den kreative, og personer som tilhører den ene retningen kan derfor ha problemer med å tolke de representasjoner som blir brukt av den andre.

Representasjoner benyttet i konstruerende design

I konstruerende designarbeid har det tradisjonelt blitt mest fokusert på den semantiske rollen til designrepresentasjonen [Trætteberg, 2002:1] og fokuset har i stor grad vært rettet mot hvordan brukergrensesnittet passer inn som en konkret, logisk løsning på toppen av en lagstruktur. Det tas utgangspunkt i en nøyaktig spesifikasjon som bestemmer hvordan brukergrensesnittet skal se ut og oppføre seg. Så blir problemstillinger og løsninger analysert ut fra de spesifiserte kravene.

Konstruerende designere er opplært til å se tekniske løsninger, identifisere og analysere spesifike krav og problemer knyttet til disse, å lage abstrakte modeller av en konkret virkelighet og beskrive dataflyt mellom komponenter og mellom forskjellige systemlag. De har derimot sjeldnere innsikt i å vurdere mange forskjellige grafiske løsninger og hvordan de forskjellige alternativene oppleves av en bruker med liten innsikt i systemet eller designerens tankegang.

Representasjoner benyttet i kreativ design

I kreativ design er det viktigste hvordan man kan bygge opp et brukergrensesnitt slik at brukbarheten ivaretas i størst mulig grad. Eksperimentering med forskjellige medium samt utforskning og testing av forskjellige designløsninger er noe som kjennetegner denne retningen. De initielle kravene tenderer å være færre og relativt fleksible, ettersom de ofte endrer seg underveis i designprosessen.

Når det gjelder personer tilknyttet kreativ design er de ofte flinke til å se grafiske løsninger, vurdere alternativer, forestille seg hvordan komponentene som finnes i et brukergrensesnitt henger sammen og påvirker hverandre, og hvordan alt dette oppleves av andre personer uten samme innsikt som dem selv. På den annen side

er det nok vanskeligere for kreative designere å lese tekniske diagrammer, modeller og spesifikasjoner, enn for konstruerende designere. Kreative designere har i så måte behov for en modell som er abstrahert ned til et nivå der de konkrete detaljene ikke lenger virker forstyrrende, men samtidig er så konkret at det lar seg gjøre å forestille seg det endelige designet. Det er også en fordel om modellen er i stand å *konkretisere* abstrakte analyse- , objekt- og interaksjonsmodeller.

En hybrid representasjon

Hver av disse retningene har sine fordeler og ulemper. Denne oppgaven vil resultere i en ny notasjon som vil kombinere den analytiske tankegangen og bruken av abstrakte representasjoner fra konstruerende design, med den utforskende tankegangen og fokus på problemforståelse fra kreativ design.

Interaksjon mellom menneske og maskin

Det er ingen tvil om at hvordan interaksjonen mellom en bruker og en maskin oppleves av brukeren har stor innvirkning på brukbarheten til systemet. Det samme gjelder interaksjonen innad i brukergrensesnittet. Det er viktig at denne er usynlig for brukeren, at responsen er rask og at data flyter lett. Det er imidlertid vanskelig å teste interaksjonen til et datasystem tidlig i et utviklingsløp, siden brukergrensesnittet på dette stadiet ofte kun er representert i statisk form. All interaksjon må på dette stadiet enten simuleres eller presenteres i form av en abstrakt representasjon som beskriver interaksjonen. I sluttfasen har det vist seg at for en sluttbruker som skal evaluere interaksjonen, kan en abstrakt representasjon være like enkel å forstå som intrikate grensesnittskisser. [Preece et. al., 2002] En lettfattelig, abstrakt representasjon av interaksjonen, som kan evalueres direkte av sluttbruker, kan derfor være et betydelig hjelpemiddel i brukersentrert design.

Abstrahering av brukergrensesnitt

Når man ønsker å abstrahere et brukergrensesnitt kan det være vanskelig å beskrive oppførselen til komponenter eller komponenter på en enkel, lettfattelig og utvetydig måte. All abstrahering er i prinsippet en forenkling av en mer kompleks virkelighet. En forenkling gjort rett vil føre til at problemområdet trer klarere fram og gjør det lettere å analysere og finne løsninger på det. Det er på den annen side fort gjort at man gjør en feil i denne prosessen, slik at modellen ikke lenger blir nøyaktig i forhold til den virkeligheten den er ment å beskrive.

Massive og komplekse brukergrensesnitt

Det kan være spesielt vanskelig å abstrahere brukergrensesnitt som er ‘massive’ og/eller ‘komplekse’. Massive brukergrensesnitt kjennetegnes ved at de inneholder mange komponenter på et begrenset område, for eksempel en kalkulator eller en medieavspiller. I komplekse brukergrensesnitt benyttes komponenter som utfører flere oppgaver, fyller flere roller, etc. Et eksempel på dette er en kalenderfunksjon i en pda eller mobiltelefon, der en og samme trykknapp har flere oppgaver alt etter som hvilket skjermbilde som er aktivt. I disse tilfellene kan det være vanskelig å holde oversikten over dataflyten til og fra de enkelte komponentene.

Grafikk kontra funksjonalitet

Når komponenter i et brukergrensesnitt skal representeres i en modell, er det ikke alltid at komponentens størrelse stemmer overens med hvor mye funksjonalitet

den inneholder. En grafisk, liten komponent kan behøve et stort areal når den skal beskrives abstrakt. Et brukergrensesnitt som inneholder mange slike komponenter kan bli vanskelig å representere på papir.

Brukbarheten til en designrepresentasjon

Å ha fokus på brukbarheten til det systemet som skal designes og designprosessen som leder fram til det har lenge vært høyt prioritert i de fleste design- og utviklingsmiljøer. Et viktig poeng er imidlertid at for å sikre brukbarheten til det endelige produktet, bør også de representasjoner og verktøy som benyttes for å lage brukbare design også ha høy brukbarhet.

Tolkning av en designrepresentasjon

For uøvde personer er det ofte vanskelig å beholde oversikten når modellene blir for abstrakte. Å lese dataflyt, se hva forskjellige symboler betyr og hvilke representasjonsobjekter som gjør hva, er egenskaper som må trenes opp. En representasjon som benytter kjente strukturer om igjen, krever ikke at brukeren setter seg inn i så mye nytt som en representasjon som definerer alt på nytt.

Det er ofte slik at jo mer konkret en representasjon er, jo mer intuitiv er forståelsen av denne. De fleste prototyper baserer seg på dette prinsippet. En måte å gjøre modeller av brukergrensesnitt og interaksjonen i det mer intuitive, er å sette inn konkrete brukergrensesnittkomponenter i det øverste viste abstraksjonsnivået. Det er imidlertid en fare for at dette vil avgrense designrommet ved at man låser designet til å bruke bestemte komponenter.

Faktisk bruk av designrepresentasjoner

Notasjonen til abstrakte representasjoner kan være vanskelige å lære og/eller tungvinte i bruk. Dette kan føre til at modellering ikke blir benyttet i designprosessen, eller bare benyttes i liten grad. For at en abstrakt representasjon skal bli tatt i bruk bør den derfor ha en lav bruksterskel. Dette oppnås ved oppfylle enkelte krav: Den må være enkel å lære; den må være effektiv å lage; den må være lett å forstå, det vil si ha en høy lesbarhet; og det må være relativt enekelt å omsette representasjonen til en *konkret* artefakt. For å oppnå dette kan man for eksempel se for seg en modell som baserer seg på en kombinasjon av visuelle og abstrakte komponenter, og som har i seg strukturer hentet fra både modellering og prototyping.

Bruksterskel kontra brukshøyde

Det doble problemet rundt brukerterskel kontra brukshøyde, er et kjent dilemma i brukergrensesnittmodelleringen. Det ser ut til at de systemene som hittil har hatt best suksess enten har lav brukerterskel, som i de fleste tilfeller medfører en lav brukshøyde, eller høy brukerterskel, men da med tilsvarende høy brukshøyde. [Myers, et. al., 2000] En av de mest fundamentale utfordringene for brukergrensesnittverktøy er hvordan man skal benytte kraftige tilnærminger og fortsatt gjøre dem oppnåelige og forståelige for typiske brukere, altså lage et verktøy med lav bruksterskel men med høy brukshøyde. En mulig løsning på dette kan være å kombinere de egenskaper som gir lav bruksterskel (intuitive løsninger) med de egenskaper som gir høy brukshøyde (god funksjonalitet).

Hypotese

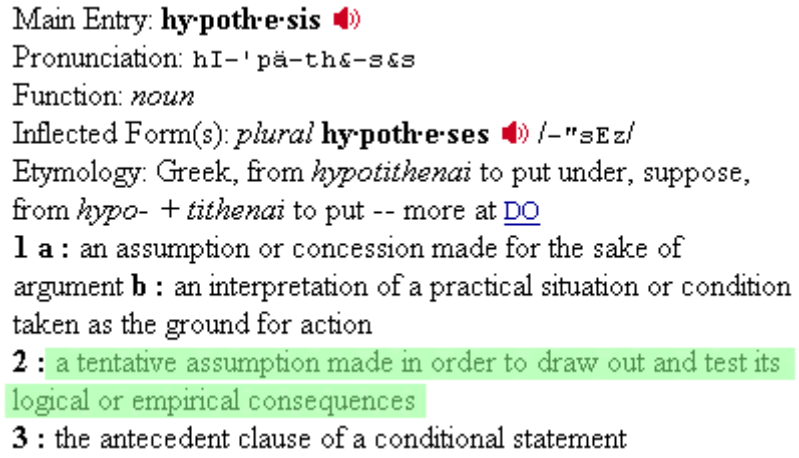

Main Entry: **hy·poth·e·sis**
Pronunciation: hI-'pä-th&-s&s
Function: *noun*
Inflected Form(s): *plural* **hy·poth·e·ses** /-"sEz/
Etymology: Greek, from *hypotithenai* to put under, suppose, from *hypo-* + *tithenai* to put -- more at DO
1 a : an assumption or concession made for the sake of argument **b :** an interpretation of a practical situation or condition taken as the ground for action
2 : a tentative assumption made in order to draw out and test its logical or empirical consequences
3 : the antecedent clause of a conditional statement

FIGURE 1-5: Fra Webster Online Dictionary: 'hy-poth-e-sis'

Ut fra de forutsetningene som er omtalt her, og den teorien som de neste tre kapitlene beskriver innenfor emnene interaksjonsdesign, -modellering og -prototyping, er det mulig å sette sammen en testbar hypotese:

Ved å kombinere notasjonen til den kanonisk abstrakte prototypen med notasjonen til DiaMODL, kan man lage en designrepresentasjon som viser både interaksjon og generiske komponenter i samme diagram. En slik representasjon vil kunne spesifisere den underliggende interaksjonen i et brukergrensesnitt på en slik måte at man fjerner eller reduserer tvetydigheter, og dermed gjøre det lettere å dokumentere denne interaksjonen, utforske alternative designløsninger, samt kommunisere ideer rundt dette til andre interessenter.

Notasjonen til en slik designrepresentasjon blir begrunnet og beskrevet i kapittel 5. Selve hypotesen testes i kapittel 6.

1.3 Oppsummering

Utgangspunktet for det videre arbeidet er spørsmålet om det er mulig å lage en modell eller prototyp som kan presentere interaksjon i et brukergrensesnitt på en lettfattelig og praktisk måte. For å undersøke dette er det viktig å skaffe mer innsikt i de aktuelle problemstillingene som er gjennomgått og se om det allerede finnes noe som enten kan brukes direkte eller som det er mulig å dra veksel på i et videre arbeid.

Spesielt tre områder er interessante i denne sammenhengen: Interaksjonsdesign, interaksjonsmodellering og prototyping. De neste kapitlene vil derfor ta for seg den eksisterende teorien som finnes på disse tre områdene.

Kapittel 2

Interaksjonsdesign

Dette kapittelet gir en generell oversikt over fagfeltet interaksjonsdesign. Det tar for seg noen standarder som er utviklet for å støtte oppunder designprosessen, forskjellen mellom abstraksjon og konstruksjon og hvilke forskjeller det er på tradisjonell systemutvikling og kreativ design. Til slutt belyser det hva som kjennetegner designrepresentasjoner; hvilke egenskaper de har og hvordan vi bruker dem.

2.1 En innføring i emnet interaksjonsdesign

Interaksjon

Main Entry: **in·ter·ac·tion**
Pronunciation: "in-t&-'rak-sh&n
Function: *noun*
: mutual or reciprocal action or influence
- **in·ter·ac·tion·al** /-shn&l, -sh&-n$^{&}$l/ *adjective*

FIGURE 2-1: Fra Webster Online Dictionary: 'des-ign'

Med interaksjon menes enhver kommunikasjon mellom en bruker og et datasystem, være seg direkte eller indirekte. Direkte interaksjon involverer en dialog med tilbakemelding og kontroll gjennom hele utføringen av oppgaven, via et brukergrensesnitt. Indirekte interaksjon kan for eksempel involvere bakgrunns- eller batchprosessering. Det vesentlige er at brukeren har en interaksjon med et system *i den hensikt å få gjennomført noe*. [Dix et. al., 1993]

Interaksjonen i et brukergrensesnitt kan være svært komplekst og er ofte preget av mye reaktivitet.[1] Disse to faktorene er kanskje de viktigste grunnene til at det er så vanskelig å modellere hvordan et brukergrensesnitt reagerer på ytre stimuli.

1. "Et reaktivt system kjennetegnes ved at det kontinuerlig må reagere på hendelser som inntreffer, enten i omgivelsene (eksterne hendelser) eller internt i systemet." [Flataukan, 2006]

Brukergrensesnitt

Et brukergrensesnitt kan defineres både som et verktøy for å utføre en oppgave og som "den kommunikasjonskanalen som benyttes for å utføre interaksjonen mellom en bruker og et datasystem". [Trætteberg, 2002:1] (s84 - oversatt)

Et brukergrensesnitt kan også defineres som den delen av et system som en bruker kan se, føle, høre eller oppleve. En algoritme vil ikke være direkte sansbar for en bruker, men resultatet av den og den tiden det tar å kjøre den vil som regel være det. Sånn sett er det viktig at man betrakter utvikling av brukergrensesnitt som en integrert del av den samlede programvareutviklingsprosessen, ikke en tilføyning eller en ettertanke. [Hix et. al., 1993] Og ettersom brukergrensesnittet definerer hvilke oppgaver en bruker kan få utført og hvordan disse oppgavene utføres, er det essensielt at interaksjonen mellom datasystemet og brukeren blir tatt hensyn til tidlig i utviklingsprosessen.

Alle systemer som krever inn- eller utdata fra en bruker, må altså ha en eller annen form for brukergrensesnitt. I [Myers, et. al., 2000] vises det at design og utvikling av et brukergrensesnitt, i snitt krever 48% av kildekode og 50% av utviklingstid i et prosjekt. Selv om disse tallene kan ha gått ned det siste tiåret pga økt kunnskap på området og bedre verktøystøtte, er det ingen tvil om at design av brukergrensesnitt fortsatt krever store ressurser.

Interaksjonsdesign

Main Entry: **[1]de·sign**
Pronunciation: di-'zIn
Function: *verb*
Etymology: Middle English, to outline, indicate, mean, from Middle French & Medieval Latin; Middle French *designer* to designate, from Medieval Latin *designare,* from Latin, to mark out, from *de-* + *signare* to mark -- more at SIGN
transitive senses
1 : to create, fashion, execute, or construct according to plan **:** DEVISE, CONTRIVE
2 a : to conceive and plan out in the mind <he *designed* the perfect crime> **b :** to have as a purpose **:** INTEND <she *designed* to excel in her studies> **c :** to devise for a specific function or end <a book *designed* primarily as a college textbook>
3 *archaic* **:** to indicate with a distinctive mark, sign, or name
4 a : to make a drawing, pattern, or sketch of **b :** to draw the plans for
intransitive senses
1 : to conceive or execute a plan
2 : to draw, lay out, or prepare a design

FIGURE 2-2: Fra Webster Online Dictionary: 'des-ign'

Datasystemer med et høyt innslag av interaksjon mellom systemet og brukeren blir i tradisjonell systemutvikling ofte konstruert slik at funksjonaliteten og systemets evne til å utføre oppgaver er godt ivaretatt, men med liten tanke på hvordan systemet faktisk blir brukt. Interaksjonsdesign retter fokuset på hvordan man kan konstruere systemer med høy brukbarhet, altså interaktive systemer som er enkle, effektive og brukervennlige, og som kan gi god støtte til brukere i deres hverdag og arbeidsliv. [Preece et. al., 2002] Interaksjonsdesign handler kort sagt om hva en bruker ønsker å gjøre og hvordan et system adresserer deres behov, interesser, mål og egenskaper. [Marion, Nettside:2, 200?]

I interaksjonsdesign undersøkes systemets bruksområde ved å benytte en brukersentrert vinkling til utviklingen. Dette betyr at brukernes behov dirigerer utviklingen istedenfor tekniske krav. Når brukeren settes i fokus under designprosessen ønsker man på denne måten å skape positive brukerefaringer ved å fornye og forbedre måten mennesker arbeider, kommuniserer og omgås. [Preece et. al., 2002]

Interaksjonsdesign er mer enn visuell design

Interaksjonsdesign er bygd opp av konseptuell og fysisk design. Konseptuell design omhandler utvikling av en konseptuell modell som fanger hva produktet kommer til å gjøre, og hvordan grensesnittet og det underliggende systemet vil oppføre seg. Fysisk eller visuell design omhandler detaljene rundt designen; utvelgelsen eller designen av visuelle komponenter som skal benyttes og hvordan disse skal plasseres i grensesnittet. [Constantine et. al., 2003] [Preece et. al., 2002]

Når man utvikler et interaktivt design, er det viktig å tenke at man bygger en komplett brukeropplevelse, - et design som er bygd opp av rom og interaksjoner, ikke vinduer og knapper. Brukeropplevelsen skapes av koblingen mellom arkitektur og brukergrensesnitt, mellom struktur og mening. [Marion, Nettside:2, 200?]

Et annet viktig poeng er at brukerens opplevelse av interaksjon blir påvirket av informasjonsarkitekturen til systemet. Informasjonsarkitekturen er den delen av et system som administrerer informasjon, og det er svært viktig at den er designet slik at det tilsvarer brukerens forventninger, altså at rett informasjon hentes ut på en intuitiv måte. [Sinha, 2004]

Utforskning av designrommet

Konseptet om et ‘designrom’ omhandler de begrensninger som skapes langs noen designretninger, samtidig som andre åpnes for utforskning. For hvert spesifikt designproblem, identifiserer designeren et sett av designbegrensninger. Disse begrensningene skaper rammen rundt designrommet, som så formes av et initielt sett av ideer. Designeren og brukeren utforsker så dette rommet sammen, utvider det og trekker det sammen, og velger en spesiell retning de ønsker å følge videre. Slik stenges enkelte enkelte deler av rommet av, mens andre åpnes opp for videre utforskning. Hele prosessen med stadig nye designvalg og sykliske utvidelser og

sammentrekninger fortsetter inntil en tilfredsstillende løsning nås. [Beaudouin-Lafon et. al., 2003]

2.2 Designprosessen

Main Entry: **[1]pro·cess**
Pronunciation: 'prä-"ses, 'prO-, -s&s
Function: *noun*
Inflected Form(s): *plural* **pro·cess·es** /-"se-s&z, -s&-, -"sEz/
Etymology: Middle English *proces,* from Middle French, from Latin *processus,* from *procedere*
1 a : PROGRESS, ADVANCE <in the *process* of time> **b :** something going on **:** PROCEEDING
2 a (1) **:** a natural phenomenon marked by gradual changes that lead toward a particular result <the *process* of growth> (2) **:** a natural continuing activity or function <such life *processes* as breathing> **b :** a series of actions or operations conducing to an end; *especially* **:** a continuous operation or treatment especially in manufacture
3 a : the whole course of proceedings in a legal action **b :** the summons, mandate, or writ used by a court to compel the appearance of the defendant in a legal action or compliance with its orders
4 : a prominent or projecting part of an organism or organic structure <a bone *process*>

FIGURE 2-3: Fra Webster Online Dictionary: 'pro-cess'

Generelt kan man si at en designprosess primært har fire roller: Sørge for å veilede rekkefølgen til en designgruppes aktiviteter; spesifisere hvilke artifakter som skal utvikles og når i prosessen det skal skje; dirigere oppgavene til både individuelle designere og til gruppen som helhet; og tilby kriterier for overvåking og måling av prosjektets produkter og aktiviteter.

Designprosesser brukes både for å forhindre at prosjekter blir utviklet ad hoc og påse at suksess ikke avhenger av noen få dedikerte nøkkelpersoner, men av hele organisasjonen som helhet. Organisasjoner som benytter veldefinerte prosesser kan lettere utvikle komplekse systemer på en repeterbar og forutsigbar måte. [Kruchten, juni 2001]

Standarder for design av interaktive systemer

Det finnes noen internasjonale standarder som regulerer utvikling og design av interaktive systemer. Basert på andres erfaringer om hvilke designprinsipper og

-regler som har vært vellykkede, gir de designere et rammeverk som støtter dem i designprosessen.

ISO 9241-11: Veiledning i brukbarhet

ISO 9241-11 er en internasjonal standard som ble ferdigstilt i 1998. Den argumenterer for fordelene ved å måle brukbarhet etter i hvor stor grad bruksmålene er oppnåd, ressursene som må brukes for å oppnå disse målene, og i hvilken grad brukeren finner bruken av systemet akseptabel. [ISO 9241-11, Org:2, 1998]

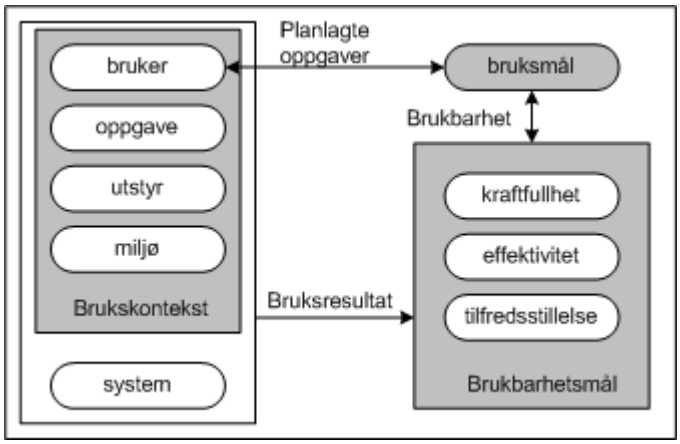


FIGURE 2-4: Skjematisk framstilling av ISO 9241-11 [ISO 9241-11, Org:2, 1998]

Brukbarhetsmålene som beskrives i standarden er: ‘Kraftfullhet’ (effectiveness), som definerer i hvilken grad et mål eller en oppgave er oppnådd; ‘effektivitet’ (efficiency), som beskriver hvor anstrengende utførelsen av dette målet er; og ‘tilfredsstillelse’ (satisfaction) som refererer til hvor tilfredsstillende bruken av et system er. [Tajakka, Nettside:6, 2005] [ISO 9241-11, Org:2, 1998]

Bruksmålene handler om hva brukerne ønsker å oppnå gjennom å bruke systemet. Disse målene beskrives ut fra brukernes perspektiv, og skal formuleres på et slikt sett at de er målbare. Selve formuleringen skal være et resultat av en dialog med brukerne. [Tajakka, Nettside:6, 2005] Hvert mål bør deles opp i delmål som spesifiserer enkeltkomponentene hovedmålet er bygd opp av, og hvilke brukskriterier som tilfredsstiller hver enkelt av disse målene. [ISO 9241-11, Org:2, 1998]

For at brukbarheten skal kunne måles er det viktig å forstå og spesifisere brukskonteksten til systemet som skal designes. Dette inkluderer identifisering av brukere, oppgaver, utstyr, samt det fysiske og sosiale miljøet systemet skal brukes i. Det man ønsker å oppnå er først og fremst en bedre forståelse av brukerne: Hva de ønsker å oppnå med systemet, hvilke prosedyrer som trengs for å gjennomføre en oppgave og hvordan brukere utveksler informasjon med andre, før eller under utførelsen av en oppgave.[Dick, 2001] [ISO 9241-11, Org:2, 1998]

Av figuren ser vi at både de planlagte bruksmålene og det ferdige systemet må oppfylle målekriteriene som definerer systemets brukbarhet dersom standarden skal følges.

iterative prosesser

Iterative prosesser baserer seg på sykliske gjennomganger av forskjellige designaktiviteter på forskjellige detaljeringsnivå. Den sykliske basismodellen åpner for utstrakt bruk av prototyping, noe som gir rom for en utforskende, eksperimentell og evolusjonær tankegang. [Bråten, 2004]

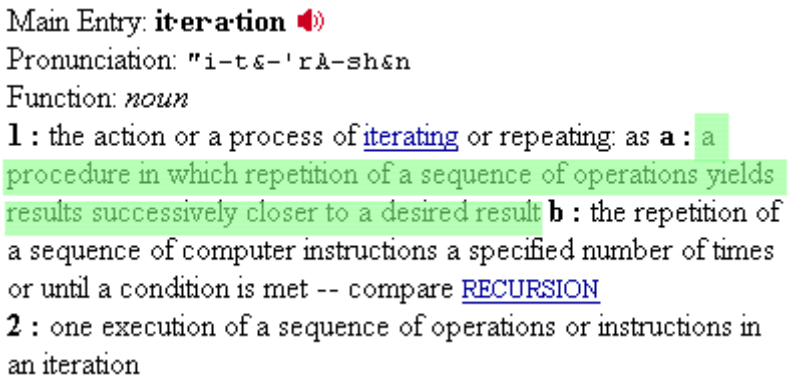

Main Entry: **it·er·a·tion**
Pronunciation: "i-t&-'rA-sh&n
Function: *noun*
1 : the action or a process of iterating or repeating: as **a :** a procedure in which repetition of a sequence of operations yields results successively closer to a desired result **b :** the repetition of a sequence of computer instructions a specified number of times or until a condition is met -- compare RECURSION
2 : one execution of a sequence of operations or instructions in an iteration

FIGURE 2-5: Fra Webster Online Dictionary: 'it-er-a-tion'

Spesifisering, analysering, design og evaluering

Hver iterasjon ender opp i en prototyp som så blir gjenstand for en brukbarhetstest. Resultatet av denne testen evalueres med hensyn på de mål som tidligere har blitt satt rundt brukbarhet og brukererfaring. På bakgrunn av evalueringen kan man luke vekk feil ved designen, funksjonaliteten eller arkitekturen, og forbedre mangelfulle eller forvirrende brukergrensesnitt. [Bråten, 2004] [Preece et. al., 2002]

I den neste iterasjonen syklus endres så prototypen med hensyn på denne evalueringen, deretter reevalueres den endrede prototypen og man tenker gjennom hvorvidt disse endringene faktisk har forbedret brukergrensesnittet og interaksjonen. Hver iterasjon medfører en viss progresjon med hensyn på dybden til designen. På denne måten trer designen fram gjennom repeterende design-evaluering-redesign sykluser. [Preece et. al., 2002]

ISO 13407: Menneskesentrert design for interaktive systemer

ISO 13407 definererer en standard for hvordan man utfører en iterativ designprosess. Standarden foreslår at prosessen deles inn i fem faser, hvor de fire siste itereres inntil systemet er fullført: Planlegge den brukersentrerte designprosessen; forstå og spesifisere brukskonteksten; spesifisere bruker- og organisasjonskrav; produsere designløsninger; og evaluere med hensyn på kravene. [Dick, 2001] [Navalkar, Nettside:3, 200?]

I forkant av den siste fasen er det essensielt at man gjennomfører brukbarhetstester slik at man har noe å evaluere.

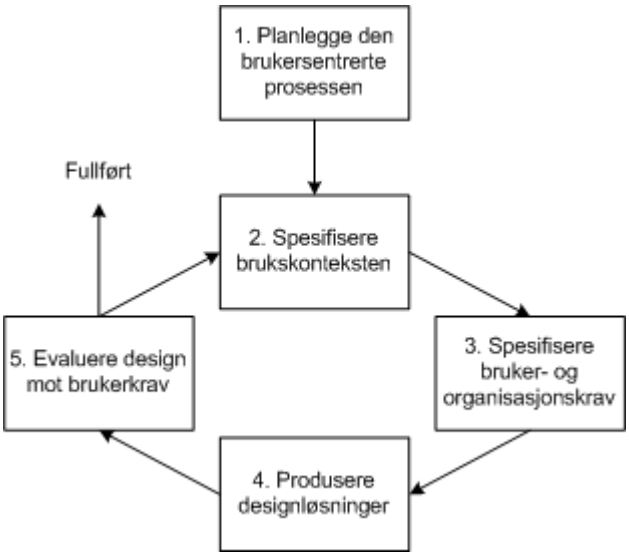


FIGURE 2-6: Skjematisk framstilling av ISO 9241-11 [Navalkar, Nettside:3, 200?]

Brukersentrert design

Brukersentrert design henter mye inspirasjon fra den prosessorienterte utviklingstilnærmingen, der man betrakter både utvikling og bruk av programvare i sammenheng med menneskelig læring, arbeid og kommunikasjon. [Floyd, 1987] Den prosessorienterte tilnærmingen forsøker å kombinere formell dokumentasjon med et iterativt utviklingsforløp og ansvarsdeling mellom alle interessenter i et utviklingsprosjekt. Det faktiske produkt blir da et resultat av de sammenvevde prosesser i analyse, design, implementasjon og evaluering. [Bråten, 2004]

I brukersentrert design involveres brukeren som en likeverdig partner gjennom alle fasene av et prosjekts utviklingsforløp. Slik tidlig og aktiv involvering av brukere skal hjelpe designere å unngå feilslutninger og uheldige designretninger, og dessuten gi grunnlag for en dypere forståelse av de faktiske designproblemene. [Mao et. al., 2005]

Brukere har erfaring om sine egne arbeidsoppgaver og rutiner, og er derfor godt egnet til å forstå og forklare konteksten systemet vil bli brukt i og identifisere subtile aspekter rundt problemene som skal løses. Både designeren og brukeren kan komme med innovative ideer, men designeren har ansvar for å vurdere de sammen med og opp mot andre muligheter som ikke nødvendigvis er kjent for brukeren. [Beaudouin-Lafon et. al., 2003]

Hovedmålet med brukersentrert design er å sørge for at sluttproduktet tilfredsstiller sluttbrukerens krav med hensyn på brukbarhet. [Mao et. al., 2005] For å oppnå et system med høy brukbarhet er det essensielt å ha god kommunikasjon med sluttbruker underveis i utviklingsløpet. Det er også viktig at designfeil oppdages så tidlig som mulig i utviklingsløpet. Brukbarheten til et datasystem bestemmes ut fra følgende mål: Hvor effektivt systemet er; hvor godt det støtter brukerens oppgaver; hvor godt det beskytter brukeren mot uønskede situasjoner; hvor godt systemet

utfører sine oppgaver; hvor lett det er å lære å bruke; og hvor lett det er å huske hvordan systemet skal brukes. [Kruchten, juni 2001]

Abstraksjon kontra konstruksjon

Selve arbeidet med å designe et nytt brukergrensesnitt kan deles inn i to forskjellige måter å angripe problemområdet på; abstraksjon og konstruksjon. Begge disse angrepsmåtene må benyttes vekselvis i designprosessen. Grovt sagt kan man si at ved abstraksjon forsøker man å lage en representasjon hvis hensikt er å forklare den virkelige verden, mens ved konstruksjon forsøker man å bygge en kunstig verden basert på den forenklede beskrivelsen som er kommet fram i abstraksjonsprosessen. [Hartvigsen, 1998]

Brukeren har kunnskap om hva som er ønsket og forventet av et system, og hvordan rutiner og oppgaver utføres i den virkelige verden. Dette er konkret kunnskap som er nødvendig å besitte når et brukergrensesnitt skal designes, og hører dermed til konstruksjonsfasen av prosessen. Designeren sitter derimot som ekspert på nødvendig kunnskap om hvordan man kan generalisere og abstrahere et gitt problemområde til mer lettfattelige modeller. Dette er abstrakt kunnskap som hører til under abstraksjonsfasen.

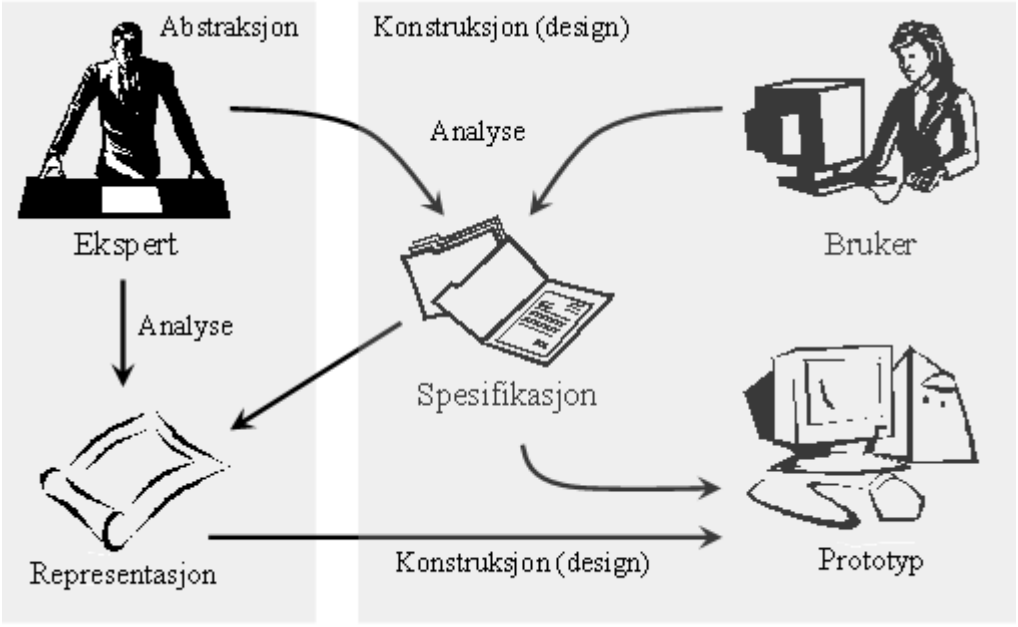


FIGURE 2-7: Sammenhengen mellom abstraksjon og konstruksjon/design

Bruker og designer går først sammen om å analysere det konkrete problemområdet, herunder eksisterende manuelle og automatiske rutiner, brukerens konkrete krav til utføring av oppgaver, systemets ønskede oppførsel, og så videre. På grunnlag av denne analysen produserer de enten en ny eller modifiserer en eksisterende spesifikasjon, som inneholder alle krav man har til brukergrensesnittet med hensyn på hvordan systemet skal utføre sine oppgaver og hvordan oppførse-

len skal være. Spesifikasjonen kan dermed sies å være en konstruksjonsartefakt. Designeren foretar så en analyse av det mer abstrakte problemdomenet, så som kompabilitet med andre systemer, plattformer, osv. Denne analysen danner sammen med de konkrete kravene fra spesifikasjonen grunnlaget for forskjellige designrepresentasjoner i form av tekster, skisser, modeller og utforskende prototyper[1]. Slike representasjoner er således en forenkling av den virkelige verden, og er derfor artefakter som tilhører abstraksjonsfasen..

På grunnlag av spesifikasjonen og de abstrakte designrepresentasjonene kan man så konstruere en endelig prototype. Denne prototypen er så gjenstand for en evaluering som avgjør om den er klar for å utvikles til det endelige systemet eller om den skal danne grunnlag for en ny syklus i designprosessen.

2.3 Systemutvikling kontra kreativ design

Trætteberg hevder i [Trætteberg, 2002:1] at programvareutvikling tradisjonelt sett har vært delt inn i to hovedretninger: En hvor det er lagt vekt på å benytte formelle metoder for å konstruere systemer med høy funksjonalitet, og en hvor uformell eksperimentering med mediumet og høyt innslag av brukersentrering legges til grunn for å designe systemer med høy brukbarhet. Disse to retningene går henholdsvis under betegnelsene ‘tradisjonell systemutvikling’ og ‘kreativ design’.

Generelt kan man hevde at ingeniører og utviklere tilhører den tradisjonelle systemutviklingsretningen, siden de er opplært til å abstrahere vekk tekniske detaljer for å kunne konsentrere seg bedre om essensen av problemer og løsninger. På den andre siden finner vi designere som representanter for kreativ design; de foretrekker ofte å jobbe mot mer generelle løsninger ved å ta i bruk konkrete artefakter som prototyper og skjermbilder. [Trætteberg, 2002:1]

Tradisjonell systemutvikling

Tradisjonell systemutvikling har en ovenfra-og-ned tilnærming, fra problem til løsning. På dette settet kan de formelle beskrivelsene som utledes i hvert steg ses på som en spesifikasjon eller et sett med krav for det neste steget. [Trætteberg, 2002:1] Systemutvikling er en god tilnærming til det å konstruere vellykket funksjonell programvare, men har problemer med å levere gode brukergrensesnitt. Det å designe brukbar programvare er altså ikke det samme som å utvikle sunn og komplett programvare. [Kim, 1995]

Kreativ design

Kreativ design har fokusert på å benytte uformell eksperimentering og brukermedvirkning for å designe systemer med høy brukbarhet, i tett samspill med problemdefinering og problemløsing. Konstruksjonsmediumet som benyttes for å bygge løsningen gir ikke bare potensialet for å løse et problem, men representerer også

1. Utforskende prototyper er raske prototyper, som regel skissert eller tegnet på papir,. Disse blir gjennomgått grundigere i kapittel 4.

en måte å definere problemet på i første omgang. Forståelsen av og eksperimentering med mediumet under utvikling går hånd-i-hånd med forståelsen og spesifiseringen av det originale problemet. [Trætteberg, 2002:1]

Koblingspunktet mellom retningene

Ved å kombinere disse retningene kan man få utviklet systemer med god og stabil funksjonalitet på den ene siden og høy brukbarhet på den andre. Dette kan enten gjøres ved å ta utgangspunkt i tradisjonell systemutvikling og øke fokus på brukeren ved hjelp av en formell, modellbasert vinkling på brukergrensesnittprosessen, eller man kan utnytte den uformelle eksperimenteringen i kreativ design og innføre formalitet i form av modellering og spesifisering av de konkrete artefaktene som produseres i sytemutviklingsprosessen. [Trætteberg, 2002:1]

2.4 Designrepresentasjoner

Kort sagt kan vi si at en en designrepresentasjon er enhver artefakt som via tekst, grafikk eller annet medium, uttrykker relevant kunnskap om brukergrensesnittet i en form forståelig for mennesker. [Trætteberg, 2002:2] Hensikten med en designrepresentasjon kan være å fange opp kunnskap om domenet, støtte designprosessen generelt eller dokumentere et ferdig design. [Trætteberg, 2002:1]

Gjennom designprosessen vil normalt flere forskjellige designrepresentasjoner benyttes, alt ettersom hvilket aspekt ved designen man fokuserer på. [Newman et. al, 2000]

Formelle kontra uformelle representasjoner

Designrepresentasjoner har i hovedsak to forskjellige, komplementære anvendelser, sett fra ståstedet til hvem representasjonene er myntet på: Enten å støtte en konstruksjonsteoretisk programvareutvikling, med fokus på refleksjon, analyse og transisjon mot et kjørende system, eller å stimulere den kreative designprosessen, med fokus på å støtte kommunikasjonen mellom interessentene og gi de store designrom å jobbe i. [Trætteberg, 2002:1]

Ofte er representasjoner som benyttes innenfor konstruksjonsteoretisk programvareutvikling formelle, mens de som benyttes i kreativ design er uformelle.

Formelle representasjoner

Formelle representasjoner har en spesifik hensikt som er veldefinert og uavhengig av konteksten. Dette gjør de i stand til å formidle den objektive *intensjonen* med brukergrensesnittet til andre interessenter. Videre kan viktige egenskaper ved brukergrensesnittet utledes og undersøkes, og forskjellige deler og aspekter ved et system kan, ved hjelp av representasjonen, sammenlignes med hverandre og analyseres under ett. [Trætteberg, 2002:1] Formelle designrepresentasjoner, som for eksempel oppgave-, objekt- og dialogmodeller, kan være nyttige verktøy for å støtte både design og evaluering av brukergrensesnitt.

Komposisjonalitet og konstruktivitet

Innenfor den konstruksjonsteoretiske tilnærmingen til systemutvikling, finner vi begrepene komposisjonalitet og konstruktivitet. Et systems komposisjonalitet sier hvordan hver enkelt av elementene i domenemodellen kan beskrives i form av et sett av formelle erklæringer, uten at det er nødvendig å vurdere hvordan andre elementer vil affektere den. Konstruktiviteten til et system beskriver på den annen side hvordan man kan ta et begrenset sett av karakteristikker og bruke de både til å spesifisere ønskede kvaliteter (properties), og beskrive enkelte domeneelementer. Tilsammen gir dermed disse begrepene oppskriften på hvordan systemets karakteristikker kan utledes ved å se på delkomponentene systemet er bygd opp av og hvordan de er satt sammen. [Trætteberg, 2002:1] Ved å foreta en konseptuell abstraksjon av problem- og brukergrensesnittdomenet, kan dermed et interaktivt system uttrykkes i form av en formell modell.

Uformelle representasjoner

Uformelle representasjoner i form av blant annet prototyper og pappmodeller utnytter friheten som ligger i grove skisser og prototyper, til å stimulere en fruktbar dialog mellom interessentene og gi støtte for et stort designrom. De er ofte grovere enn formelle representasjoner. Dette blir ofte sett på som en fordel siden skisser, i motsetning til mer polerte representasjoner, i større grad klarer å fokusere på helheten i brukergrensesnittet. Dette gjør at den overordnede interaksjonen mellom bruker og system framheves, samtidig som uviktige detaljer skjules. Når det diskuteres over en grov skisse eller prototype, vil designrommet oppleves større for interessentene. Dialogen mellom dem blir dreid mot de viktige, overordnede designspørsmålene og blir mindre hemmet av detaljer. [1] Samtidig fører dette til at intensjonen til en uformell designrepresentasjon er sterkt avhengig av den spesifike konteksten den ble utarbeidet i, altså hvem som utarbeidet den, på hvilket stadium i designprosessen den ble utarbeidet, og så videre. [Kruchten, mars 2001]

Designrepresentasjoner som enkelt skal kommunisere ideer og løsningsforslag til mange interessenter bør relatere tett til nåværende oppgave og designkontekst. Av den grunn bør de også være visuelle, konkrete og uformelle heller enn abstrake og formelle. [Ehn, 1993] [Trætteberg, 2002:1]

Formalisering av en uformell representasjon.

I en prosess hvor en formaliserer en uformell representasjon, vil det kunne oppstå tvetydigheter som må tolkes ut fra konteksten den ble utarbeidet i. Dersom denne konteksten ikke lenger er tilgjengelig vil dette gjøre tolkningen vanskelig. [Trætteberg, 2002:1] For å sikre at en uformell representasjonen tolkes rett, må det enten inkluderes andre representasjoner som tydeliggjør den uformelle representasjonen når den produseres, eller mer eksplisitte og formelle representasjoner må bli lagd.

Representasjoner brukt i konseptuell design

Den ‘konseptuelle designfasen’, det vil si de første iterasjonene i en designprosess, er svært kritisk med hensyn på interaksjonsdesign. I denne fasen spesifiseres brukskontekst, bruker- og bruksmål, samt bruker-, organisasjon- og systemkrav. Det er også i disse første iterasjonene man utforsker designrommet med hensyn på hvordan det endelige brukergrensesnittet i grove trekk vil se ut, hvilken funks-

jonalitet det vil ha og hvordan interaksjonen mellom bruker og system vil foregå. [Trætteberg, Nettside:7, 200?]

Målet med den konseptuelle designfasen er å produsere en endelig designrepresentasjon, som regel en utforskende prototyp, som kan evalueres med hensyn på brukbarheten. Det benyttes imidlertid flere representasjoner og teknikker for å dokumentere, kommunisere og visualisere designen i denne fasen. [Mao et. al., 2005] [Trætteberg, Nettside:7, 200?]

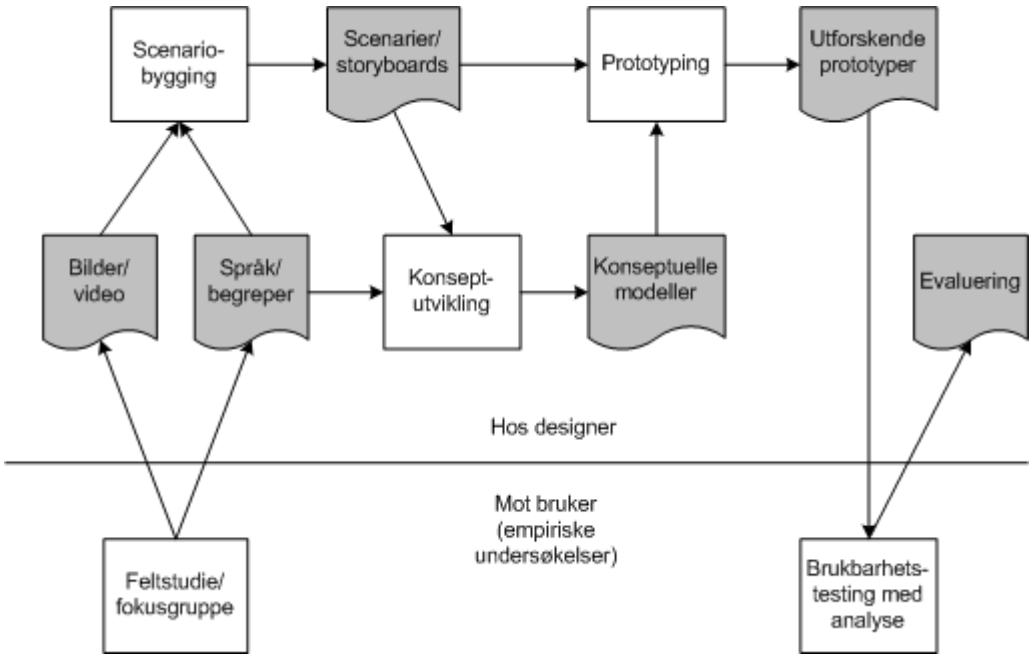


FIGURE 2-8: Oversikt over teknikker brukt i konseptuell design [Trætteberg, Nettside:7, 200?]

Utvikling av problemdomenet

Underveis i prosessen med å lage en representasjon av et design vil problemdomenet utvikle seg etterhvert som man spesifiserer flere og flere detaljer. Ofte vil designløsninger gjennom en slik abstraksjonsprosess vise seg å være ulogiske, lite brukbare eller teknisk vanskelig å implementere. Dette gjelder både dersom man modellerer eller prototyper en designløsning. Slik kan man tidlig i designprosessen luke ut designfeil som ville vært vanskelige og kostbare å rette opp dersom de ble oppdaget senere.

I følge [Beaudouin-Lafon et. al., 2003] argumenterer programmerere ofte for å programmere systemet direkte alt i de tidlige fasene av et prosjekt; de tror at siden de allerede er kjent med et programmeringsspråk, vil det være både raskere og mer nyttig å skrive koden direkte enn å kaste bort tid på å lage prototyper som allikevel kastes etter bruk. I løpet av 20 år med prototyping, både industrielt og forskningsmessig, hevder forfatterne at de enda har til gode å se en situasjon der dette er sant.

Representasjoners forskjellige roller

Forskjellige designrepresentasjoner kan ha forskjellige roller ut fra form og kontekst. Trætteberg presenterer i [Trætteberg, 2002:1] de fem viktigste rollene en designrepresentasjon kan ha:

Semantisk rolle

Designrepresentasjonens semantiske rolle er å fange kunnskap om, eller semantikken til, domenet. Den tekniske programvareutviklingstilnærmingen har tradisjonelt fokusert mye på dette. Representasjonene utvikler seg semantisk ved å gå via en kjede av transformasjoner fra abstrakte krav til konkrete artefakter.

Kommunikativ rolle

Den kommunikative rollen til designrepresentasjonen er å kommunisere eller beskrive den representerte domenekunnskapen til andre interessenter. Dette er svært viktig i kreativ design, som kjennetegnes med tett samspill mellom problemsetting og problemløsning. Dette krever en form for kommunikasjonsmedium som kan støtte dialogen og den asynkrone kommunikasjonen mellom designerne slik at de er i stand til å diskutere hver enkelts forståelse av problemet og foreslåtte løsninger.

Konstruktive representasjoner

Den konstruktive rollen til designrepresentasjonen er å veilede og begrense den videre utviklingen av representasjoner. Problemløsningsprosessen går gjennom mange steg, fra målsetninger til oppgavebeskrivelser, fra oppgaver til dialogstrukturer, og fra dialog til interaksjonsdetaljer. Hvert steg gir økt forståelse, økt konkrethet og økt detaljering og en forståelse for hvordan hvert steg påvirker produktets brukbarhet er viktig. En representasjon av både problemet og den foreslåtte løsningen kan være nyttig både som en kilde til inspirasjon og som en begrensende veiledning.

Analytisk rolle

Den analytiske rollen til designrepresentasjonen er å tolke og evaluere den foreslåtte designløsningen. Designerens problemforståelse krever en analytisk gjennomgang av organisasjonens nåværende praksis. Videre må en designløsning bli evaluert mot både de mål og oppgavestrukturer som er fastsatt og mot erklærte brukbarhetsmål. Generelt kan man si at de løsningene som hver designfase har resultert i, må analyseres og sammenliknes med krav og begrensninger fra tidligere faser.

Kjørende rolle

Den kjørende rollen til designrepresentasjonen er å støtte endelig avslutning og utførelse av løsningen. Realiseringen av det faktiske brukergrensesnittet krever en representasjon som er forståelig for den underliggende plattformen. En ideell designrepresentasjonen bør derfor kunne oversettes direkte til en kjørende versjon, med et minimum av menneskelig anstrengelse.

Sentrale faktorer ved representasjoner

Noen få sentrale faktorer bestemmer hvilken informasjon en representasjon inneholder, og hvordan den uttrykker denne informasjonen. disse faktorene er henholdsvis: perspektiv, abstraheringsgrad, abstraksjonsgrad, dekningsgrad og formalitetsgrad .

Hvor mye informasjon en interessent får ut av en notasjon er knyttet opp mot disse faktorene. Skal en interessent tolke representasjonen rett, må han vite: Hvilken informasjon som er beskrevet og hvilken som er fjernet; hvilken informasjon symbolene i notasjonen uttrykker; og hvordan de forskjellige symbolene påvirker hverandre.

Perspektiv

Perspektiv handler om i hvilken grad designrepresentasjoner er problem- eller løsningsorientert. En designrepresentasjon kan sies å være problemorientert dersom den forsøker å beskrive problemområdet som undersøkes. Dette er for eksempel direkte krav til problemområdet og hvilke mål man ønsker å oppnå med designen. En løsningsorientert representasjon prøver på den annen side å beskrive konkrete aspekter ved brukergrensesnittet som blir designet, som interaksjonsstiler og hvilken kontekst designen vil bli brukt i. Ofte er en designrepresentasjon både problem- og løsningsorientert, siden en representasjon både kan brukes til å representere løsningen på et spesifikt problem, og spesifikasjonen på et problem som må løses senere i designprosessen. [Trætteberg, 2002:1]

Abstraksjonsgrad

Abstraksjonsgrad handler om hvordan informasjonen uttrykkes i en representasjon. Jo mer abstrakt en representasjon er, jo tydeligere utrykkes den rene *ideen* om systemet. I motsatt ende av skalaen har vi de konkrete representasjonene, som beskriver systemet svært sanselig, det vil si at de ser ut som, oppleves eller føles som et reelt system.

Abstraheringsgrad

Abstraheringsgrad beskriver hvor *spesialisert* representasjonens informasjon er i forhold til det helt konkrete systemet, altså hvor detaljert representasjonen er, og kan for eksempel beskrive grovheten i en skisse eller dybdepenetreringen i en dialogmodell. Innenfor hvert perspektiv, vil en representasjon typisk brukes til å modellere objekter med forskjellig abstraheringsgrad. [Trætteberg, 2002:1]

Abstrahering innebærer at man fjerner informasjon som ikke er direkte relevant for det problemområdet man ønsker å undersøke eller beskrive og som derfor vil virke forstyrrende. Abstraheringsgraden til representasjonen vil da øke.

Abstraksjons- og abstraheringsgrad henger ofte tett sammen, ettersom det er lettere å uttrykke en ren ide når uvesentlige detaljer er fjernet fra representasjonen.

Dekningsgrad

Dekningsgrad beskriver hva informasjonen i en representasjon omhandler. Et problemområde har mange aspekter, og dekningsgraden sier hvilke aspekter av et problemområde en representasjon dekker. Hvilke aspekter som kan beskrives til enhver tid er avhengig av domenet. To aspekter som kan beskrives når man skal designe et system er for eksempel statisk struktur og dynamisk oppførsel.

Abstraherings- og dekningsgrad er to uavhengige størrelser. I mange tilfeller er det imidlertid slik at man velger å fokusere på et aspekt av gangen når man abstraherer ned en representasjon.

Formalitetsgrad

Formalitetsgraden til en representasjon angir hvor forståelig en representasjon er for en datamaskin. Spenningsforholdet finner vi i menneskers evne til å forstå, analysere og manipulere en representasjon, hvor selve situasjonen eller konteksten benyttes for å gi mening til representasjonen, kontra et utviklingsverktøy evne til å gjøre det samme. [Trætteberg, 2002:1]

Et dataverktøy kan ikke foreta spranget fra input til output uten å vite reglene som produserer dette resultatet. For mennesker er det imidlertid relativt lett å tenke på denne måten. For eksempel vil teksten “ball” og bilde av en ball konseptuelt bety det samme for et menneske, men en datamaskin må eksplisitt programmeres for å skjønne en slik sammenheng.

Jo mer formell en representasjon er, jo mer kontekstavhengig og eksplisitt er altså kunnskapen. Mens full konsistens og fullstendig analyse er utenfor rekkevidden av vår forståelse, kan vi allikevel bruke formelle representasjoner for veiledning, og mens maskiner ikke klarer å gi mening til uformelle representasjoner, kan de allikevel gi støtte for strukturering, filtrering, visning og navigering av dem. [Trætteberg, 2002:1].

Formalitetsgraden er ofte bundet opp mot abstraherings- og dekningsgraden, ettersom jo mer abstrahert en representasjon er og jo mindre dekningsgrad den har, jo lettere er det å lage en syntaks som beskriver den korrekt. Det er imidlertid ingen automatikk i dette.

2.5 Oppsummering

Interaksjonsdesign handler om å designe en brukeropplevelse, og er mye mer enn bare det visuelle designet. Målet er å lage et system med høy brukbarhet.

Ved å kombinere systemutviklingsretningens formelle tankegang med den kreative designretningens fokus på uformell eksperimentering og utforskning, vil det være lettere å utvikle systemer med god og stabil funksjonalitet på den ene siden og høy brukbarhet på den andre.

En designrepresentasjon er en artefakt som forsøker å uttrykke relevant kunnskap om et design i en form som er forståelig for mennesker. Hensikten med en designrepresentasjon kan være å fange opp kunnskap om domenet, støtte designprosessen generelt eller dokumentere et ferdig design. Vi skiller mellom formelle og uformelle designrepresentasjoner.

For å undersøke hvordan et framtidig system vil samhandle med brukeren, benyttes i hovedsak to forskjellige former fordesignrepresentasjoner; interaksjonsmodeller og interaksjonsprototyper.

Ettersom målet er å lage en hybrid notasjon som ivaretar de beste egenskapene fra hver av disse representasjonsformene, er det nødvendig å undersøke hva som kjennetegner interaksjonsmodeller og - prototyper, hvordan de blir benyttet og hvilke spesifike representasjoner som best egner seg til å lage en hybrid notasjon. Dette skal vi se på i de neste to kapitlene.

Kapittel 3

Interaksjonsmodellering

Dette kapittelet tar for seg hva som egentlig menes med interaksjonsmodellering, hvordan modellering benyttes i forbindelse med interaksjonsdesign, og hva som faktisk kan modelleres. Det beskriver og sammenligner så noen av de mest brukte modellene som finnes innenfor dette fagfeltet. Til slutt fokuserer det på dialogmodellen DiaMODL, og hvordan den modellerer interaksjon ved hjelp av interaktorer og en interaktorstruktur som knytter disse sammen.

3.1 Hva er interaksjonsmodellering

Main Entry: **[1]mod·el**
Pronunciation: 'mä-dᵊl
Function: *noun*
Etymology: Middle French *modelle,* from Old Italian *modello,* from (assumed) Vulgar Latin *modellus,* from Latin *modulus* small measure, from *modus*
1 *obsolete* **:** a set of plans for a building
2 *dialect British* **:** **COPY**, **IMAGE**
3 : structural design <a home on the *model* of an old farmhouse>
4 : a usually miniature representation of something; *also* **:** a pattern of something to be made
5 : an example for imitation or emulation
6 : a person or thing that serves as a pattern for an artist; *especially* **:** one who poses for an artist
7 : **ARCHETYPE**
8 : an organism whose appearance a mimic imitates
9 : one who is employed to display clothes or other merchandise **:** **MANNEQUIN**
10 a : a type or design of clothing **b :** a type or design of product (as a car)
11 : a description or analogy used to help visualize something (as an atom) that cannot be directly observed
12 : a system of postulates, data, and inferences presented as a mathematical description of an entity or state of affairs
13 : **VERSION**

FIGURE 3-1: Fra Webster Online Dictionary: 'mod-el'

Hva er en modell?

En modell er en en abstraksjon av et problemområde, og representerer derfor en forenkling av virkeligheten som fullt ut forsøker å beskrive systemet fra et spesifikt perspektiv. Av denne grunn er det ofte nødvendig med forskjellige modeller for å forstå og finne løsninger på en kompleks problemstilling. Det er essensielt at disse modellene er koordinert for å sikre konsistens og hindre overflødighet. [Bråten, 2004]

En klar identifikasjon av problemområdet gjør det enklere å foreta avveininger og ta hurtige avgjørelser. Ved hjelp av modeller er det mulig å kommunisere ideer og løsninger rundt systemets design utvetydig og dermed øke interessentenes evne til å håndtere programvarekompleksitet. [Bråten, 2004] [Constantine et. al., 1998]

Dersom man direktekoder programvare jobber man i praksis i blinde og har liten kontroll på hvilken retning designen tar eller hvor langt unna det ferdige designet man er. Grundige analyse- og designmodeller forenkler prosessen med å utforske designrommet og finne gode løsninger. Det å vite hva man skal bygge er altså svært viktig når et system skal konstrueres, siden all den tid som går med til å programmere en unødvendig, utilstrekkelig eller direkte feil løsning, er sløsing med tid. [Constantine et. al., 1998]

Kunnskap om hva som faktisk blir bygd gjør dessuten utviklerne i stand til å unngå fallgruver i systemkonstruksjonen, noe som fører til bedre og mer robuste løsninger. [Constantine et. al., 1998] Modellering hjelper til med å finne mange feil i systemkonstruksjonen før den implementeres. Det har vist seg at å finne og reparere feil i programvare er fra 100 til 1000 ganger mer kostbart enn om disse feilene kunne vært identifisert i løpet av utviklingsløpet. [Kruchten, juni 2001]

Modeller er også med på å dokumentere det ferdige systemet.

Modellering av interaksjon

Modeller benyttes innenfor interaksjonsdesign først og fremst for å visualisere, spesifisere, konstruere, dokumentere og vedlikeholde artefakter som inngår i et programvareintensivt system. Både interaksjonsmodellen i seg selv og prosessen rundt det å modellere et system gir gevinst i form av raskere utviklingsprosesser og bedre systemer. [Beaudouin-Lafon et. al., 2003]

Prosessen med å lage en modell av et brukergrensesnitt krever at designeren tenker gjennom sine forslag til designløsninger på nytt. I mange tilfeller vil en interaksjonsmodell kunne avsløre inkonsistens og/eller tvetydighet i brukergrensesnittet. Slik tvinger en modell fram spørsmål om forskjellige tolkninger som designere blir nødt til å besvare.

Hva ønsker vi å modellere?

I forbindelse med interaksjonsdesign er det spesielt interessant å lage modeller som definerer oppførselen til brukeren og det aktuelle systemet og definerer selve samspillet dem imellom.

ISO9241-11 standarden påpeker at det er nødvendig med en forståelse av hvem brukerne er, miljøet de opererer i, hvilke mål de prøver å oppnå og hvilke oppgaver de utfører for å komme dit. For å kunne fange opp og bevare forståelsen av disse aspektene, må man kunne skrive de ned i en passende form, altså forestille kunnskapen i en formalisert beskrivelse eller spesifikasjon. Modeller basert på konseptuelle abstraksjoner av problem- og brukergrensesnittdomenet kan være nyttige verktøy for designere, så sant de gir støtte for de nødvendige rollene i utviklingsprosessen. [Trætteberg, 2002:1]

Kritikk mot en modellbasert design av brukergrensesnitt

Modellbasert brukergrensesnittdesign har vært mye forsket på det siste tiåret, uten at denne tilnærmingen har fått noen særlig akseptanse i programvareindustrien. En av grunnene til dette er at modellbaserte tilnærminger ofte konsentrerer seg om høynivå spesifikasjoner av brukergrensesnittet, noe som medfører at designere mister kontrollen over detaljene på lavere nivå. [Campos et. al., 2004]

Verktøyene som eksisterer fokuserer i for stor grad på formalismen som trengs for å automatisk generere konkrete brukergrensesnitt. Dette hindrer modelleringsverktøy i å gi tilstrekkelig støtte til ideprosessene og designoppgavene som designere må utføre for å lage brukbare og effektive brukergrensesnitt. [Campos et. al., 2004] [Myers, et. al., 2000]

Verktøy basert på den modellbaserte tilnærmingen har også en tendens til å ville løse hele problemet på en gang i stedet for å konsentrere seg om spesifike deler av brukergrensesnittdesignprosessen. Dette gir seg utslag i at verktøyet både får en høy bruksterskel og en relativt lav brukshøyde, altså at verktøyet både er vanskelig å lære, og at mulighetene som verktøy tilbyr er begrensede og ikke vil gi signifikante resultater. [Campos et. al., 2004]

For å styrke markedsakseptansen, trengs en ny generasjon brukersentrerte modelleringsverktøy. [Campos et. al., 2004]

Under konferansen CADUI'02 ble det identifisert flere krav som må innfris dersom en modellbasert tilnærming skal få gjennomslagskraft i industrien: sporbarhet, støtte for delvis (partial) design, kunnskapshåndtering og en glidende overgang fra abstrakte til konkrete modeller. [Campos et. al., 2004] Det siste punktet her er svært relevant for denne oppgaven.

3.2 Modellering av brukergrensesnitt

Innen feltet interaksjonsdesign har det gjennom de siste 25 årene vært brukt flere innfallsvinkler for hvordan man formelt kan beskrive interaksjonen som foregår mellom bruker og system via et brukergrensesnitt. Noen av de mest brukte

innfallsvinklene er oppgavemodellering, arkitekturmodellering og interaksjonsmodellering.

Oppgavemodellering

En oppgavemodell forsøker å fange kunnskap om brukerens miljø, rutiner og arbeidsoppgaver. Hensikten er å beskrive formelt de mål en bruker ønsker å oppnå og hvilke oppgaver som er nødvendige å utføre for å nå disse målene. [Trætteberg, 2002:1]

De kan benyttes både for å evaluere eksisterende brukergrensesnitt eller for å forberede design av et nytt brukergrensesnitt. En oppgavemodell kan i det siste tilfellet være en viktig representasjon å benytte som utgangspunkt når man skal designe selve interaksjonen som skal inngå i systemet.

Kognitiv modellering

En kognitiv modell er en variant av oppgavemodellen som fokuserer på hvilke aksjoner en bruker utfører mot et system. De kjennetegnes ved at de prøver å emulere menneskelig oppførsel, nærmere bestemt de persjepsjonelle, kognitive og/eller de motoriske prosessene mennesker må gjennom for å utføre en oppgave. [Dix et. al., 1993]

Slike representasjoner blir dermed en systematisk oppskrift på hvordan oppgaver skal utføres i brukergrensesnittet. Dette kan føre til at det blir gitt lite spillerom for utforskning av designrommet og eksperimentering med forskjellige forskjellige komponent- og designløsninger. I tillegg gir de ingen informasjon om hvordan det er tenkt at det underliggende systemet skal implementeres for at oppgavene skal utføres rent funksjonelt, altså hvordan interaksjonen innad i brukergrensesnittet, og mellom brukergrensesnitt og funksjonalitetslag, skal foregå.

Modeller som GOMS, KLM og NGOMSL, har en tendens til å overforenkle virkeligheten; de klarer ikke å ta hensyn til variasjonsbredden i verken brukere, arbeidsmåter eller oppgaver, noe som gjør det vanskelig å lage generelle modeller.

Arkitekturmodellering

En arkitekturmodell viser hvordan et brukergrensesnitt kan implementeres mot resten av systemet og hvordan brukerinteraksjonen skal organiseres. Dermed nærmer den seg problemet med hvordan man skal realisere et brukergrensesnitt fra systemsiden. [Nunes et. al., 2000] For en interaksjonsdesigner er imidlertid tradisjonelle arkitekturmodeller som MVC, Arch- eller Seeheimmodellen, ikke mye til hjelp når den konkrete interaksjonen eller brukergrensesnittet skal designes. Til det tar de for seg et alt for overordnet perspektiv. De kan allikevel være nyttige å bruke når brukergrensesnittet skal realiseres mot det funksjonelle laget i systemet.

Wisdom

En relativt ny arkitekturmodell er Wisdom. Den er interessant, også sett fra et interaksjonsdesignperspektiv. Wisdom er en konseptuell arkitektur som ikke tar hensyn til konkrete design eller implementasjonskriterier, men som introduserer et *brukersentrert perspektiv* på analysenivået. [Nunes et. al., 2000] Dette gjøres ved å sørge for at arkitekturen: Bygger på brukergrensesnittkunnskap som fanger

essensen av eksisterende vellykkede og gjennomtestede aerkitekturmodeller; integrerer den eksisterende 'unified analysis' modellen, og søker samarbeid, artefaktutveksling og sporbarhet mellom MMI- og systemutviklingsmodeller; fremmer ansvarsdeling mellom intern funksjonalitet og brukergrensesnittet; og retter seg etter UML-standarden både på semantisk og notasjonelt nivå. [Nunes et. al., 2000]

I Wisdom lages to informasjonsrom som binder den interne arkitekturen sammen med grensesnittarkitekturen og fremmer ansvarsdelingen. [Nunes et. al., 2000]

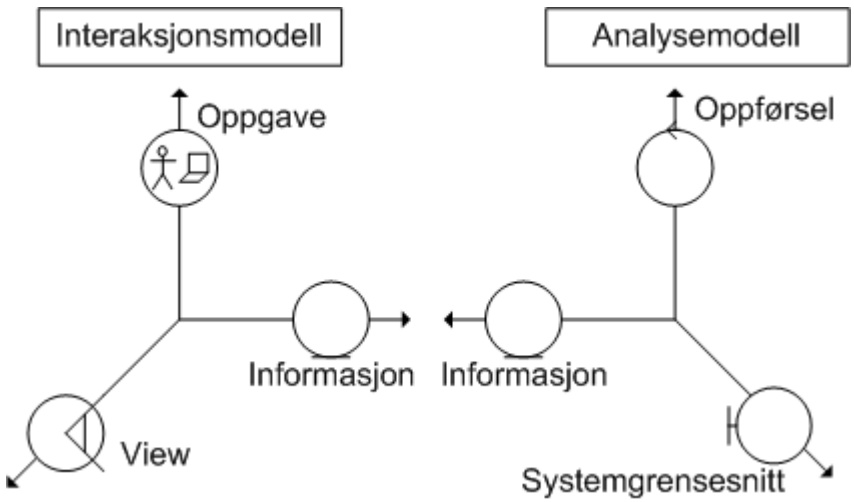


FIGURE 3-2: Wisdomarkitekturen [Nunes et. al., 2000]

Analysemodellen til Wisdom har ansvaret for å håndtere systemets oppførsel. Den tar for seg den indre arkitekturen til systemet; strukturerer de funksjonelle kravene i form av analyseklasser og overlater til etterfølgende design og implementasjonsmodeller å håndtere de ikke-funksjonelle kravene. Analyseklassene er delt inn i 'grenseklasser' (boundrary classes), 'kontrollklasser' (control classes) og 'entitetsklasser' (entity classes). Grenseklassene benyttes for å modellere systemgrensesnittet, dvs ikke-menneskelig interaksjon; kontrollklassene representerer systemets oppførsel, dvs koordinering, sekvensering, transaksjoner og kontroll over andre objekter; og entitsklassene brukes til å modellere varig informasjon og representerer den logiske datastrukturen og binde den indre arkitekturen sammen med brukergrensesnittarkitekturen. [Nunes et. al., 2000]

Interaksjonsmodellen tar for seg den overordnede organiseringen av brukergrensesnittet. Den strukturerer interaksjonen med brukere i interaksjonsklasser og utsetter håndteringen av bruker-grensesnittstiler, teknologi og andre begrensninger til de etterfølgende design- og implementasjonsmodellene. Interaksjonsklassene er delt inn i 'visningsklasser' (view classes) og 'oppgaveklasser' (task classes).

Viewklasser brukes til å modellere interaksjonen mellom systemet og brukeren. De representerer handlingsommet innenfor brukergrensesnittet hvor brukeren sam-

handler med den funksjonalitet som trengs for å utføre sine oppgaver. Viewklasser har ansvaret for den fysiske interaksjonen med brukeren, inkludert definering av systemets oppførsel. [Nunes et. al., 2000]

Oppgaveklasser brukes til å modellere dialogstrukturen mellom bruker og system, og har ansvaret for oppgavenivåsekvensering, konsistens mellom forskjellige brukergrensesnitt og kartleggingen fram og tilbake mellom entitets- og viewklasser. Oppgaveklasser innkapsler ofte kompleks oppførsel som ikke kan relateres til spesifike entitetsklasser. [Nunes et. al., 2000]

Eksempel

En bank ønsker å legge ut en kalkulator som gjør det mulig for kunden å beregne kostnader ved å ta opp et lån. Arkitekturen til en slik applikasjon kan i Wisdom modelleres slik:

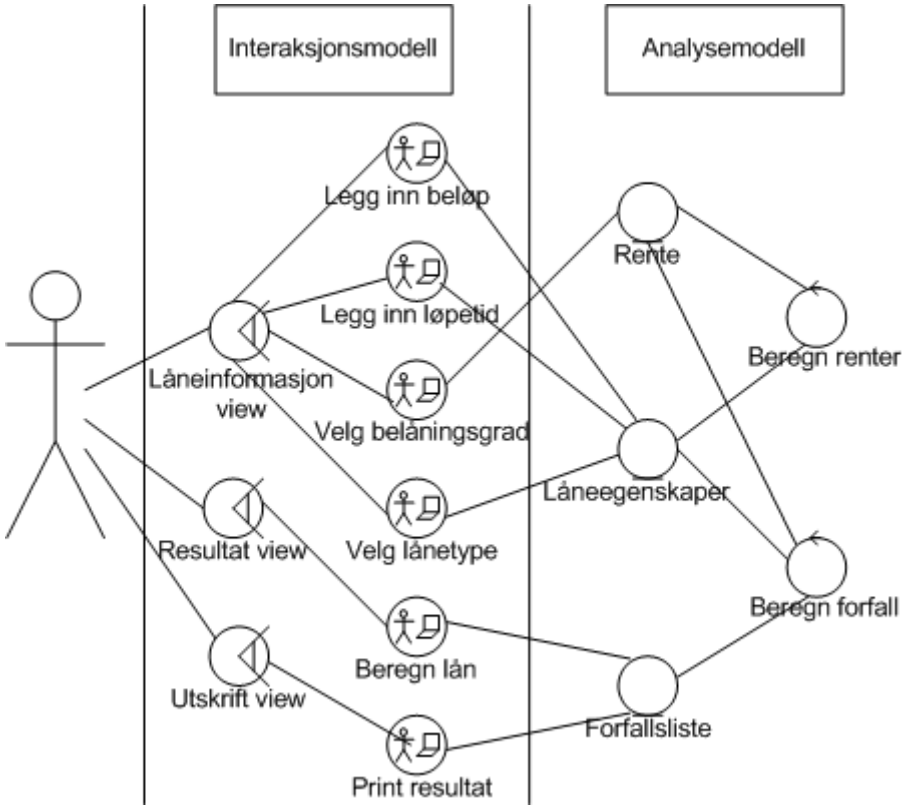


FIGURE 3-3: Wisdom-modell av en enkel lånekalkulator

Kritikk

På interaksjonssiden ligger Wisdom relativt tett opptil Use Case-modellen. Slik forsøker Wisdom å bygge en bro mellom arkitektur-, og interaksjonsmodellen. Dette klarer den ganske bra. Den er i stand til å beskrive både systemets interne arkitektur og brukergrensesnittarkitekturen. Når det gjelder beskrivelsen av interaksjonen som foregår mellom brukeren og systemet lider den imidlertid av de samme svakheter som vi etterhvert skal se finnes i Use Case-modellen. Wisdom egner seg derfor best til å definere grensesnittet mellom systemets logikk- og presentasjonsdel.

3.3 Interaksjonsmodellering

I interaksjonsdesign gjelder det å sørge for at komponentene som tilsammen utgjør brukergrensesnittet til å være koordinerte og arbeide sammen som en helhet. Dette krever en form for organisering; at man skaffer seg oversikten over de komponenter som inngår i brukergrensesnittet, meldingene som går mellom dem og meldingene som går mellom brukeren og det underliggende systemet.

Interaksjonsmodellering ser på hvordan komponenter eller objekter i brukergrensesnittet og systemet kommuniserer med hverandre og med brukeren. Innen forskning og industri har spesielt tre former for interaksjonsmodellering oppnådd stor akseptanse: scenariomodellering, tilstands- og transisjonmodellering, og dialogmodellering.

Scenariomodellering

Et scenario kan defineres som en sekvens av handlinger som beskriver interaksjonen mellom aktører i et system. I UML defineres tre forskjellige representasjoner for å modellere scenarier: Use case-, sekvens- og samarbeidsdiagrammer.

Main Entry: **sce·nar·io**
Pronunciation: s&-'nar-E-"O, -'ner-, *US also and especially British* -'när-
Function: *noun*
Inflected Form(s): *plural* **-i·os**
Etymology: Italian, from Latin *scaenarium* place for erecting stages, from *scaena* stage
1 a : an outline or synopsis of a play; *especially* **:** a plot outline used by actors of the commedia dell'arte **b :** the libretto of an opera
2 a : SCREENPLAY **b :** SHOOTING SCRIPT
3 : a sequence of events especially when imagined; *especially* **:** an account or synopsis of a possible course of action or events
<his *scenario* for a settlement envisages... reunification -- Selig Harrison>

FIGURE 3-4: Fra Webster Online Dictionary: 'sce-nar-io'

Use Case

Et use case-diagram beskriver bestemte brukerscenarier. Mer spesifikt kan man si at det er et sett av scenarier som er bundet sammen av et felles brukermål. Hovedfunksjonen til use case diagrammer er å være et hjelpemiddel når man skal kommunisere systeminteraksjon med brukere; de er ikke ment å brukes som en modell for overordnet design. [Trætteberg, Kompendium:1, 2004?] Ettersom et use case i all hovedsak er en representasjon på hvordan et system kommuniserer med brukeren og hvilke brukerbehov systemet skal oppfylle, kan de sies å være mer systemsentrert enn brukersentrert. [Bråten, 2004] [Nunes et. al., 2000]

I et use case finnes det som regel et hovedscenario der normalløpet for interaksjonen beskrives. I tillegg kan det legges til alternative scenarier som beskriver hva som skjer dersom noe avviker fra normalløpet, for eksempel at en feilmelding kommer opp fordi en bruker har glemt å fylle ut et felt i et skjema. Det er også mulig å legge til tekst som spesifiserer hvilke betingelser som må oppfylles for at scenariet skal være relevant. I use case-diagrammer er det viktig å ikke detaljere for mye. Hovedregelen er at jo større risiko det er for at en interessent kan misforstå et brukerkrav, jo flere detaljer skal use case-diagrammet inneholde. [Trætteberg, Kompendium:1, 2004?]

Eksempel

Banken ønsker at det skal være mulig å benytte lånekalkulatorapplikasjonen til å sende en søknad om lån. En saksbehandler i banken vil så behandle lånesøknaden. Et Use Case diagram for dette vil da kunne se slik ut:

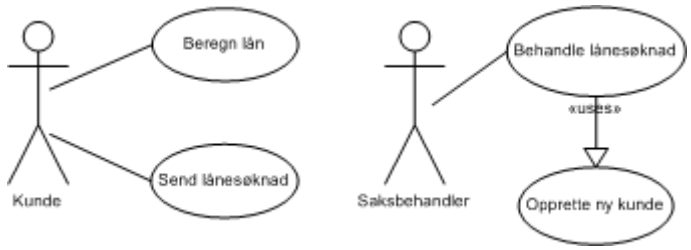


FIGURE 3-5: Use Case for søknad om lån

Kritikk

Use-case kan være et kraftig verktøy for å modellere og dokumentere brukerkrav, men generelt er de ofte for vage i sin definisjon til å være særlig effektive til noe særlig annet. Use-cases blir lett overfladiske og har en tendens til å si mer om hva som blir utført enn hvordan. Det er ikke nok å vise handlinger, ofte må man vite hvilken sekvens handlingene skjer i for at det skal være mulig å implementere effektiv interaksjon. Dette gjelder uansett om interaksjonen skjer mellom mellom bruker og system, eller innad i brukergrensesnittet. [Bråten, 2004] [Trætteberg, Kompendium:1, 2004?]

Sekvens- og samarbeidsdiagrammer

Siden UML i stor grad legger objektorientering til grunn for sine modeller, er det naturlig å tenke på interaksjon som meldinger mellom objekter. Dette blir i UML representert i form av sekvens- og samarbeidsdiagrammer. Hensikten med disse modellene er først og fremst å vise hvordan objekter utveksler informasjon, og siden begge diagrammene er relativt lettleste, kan de være effektive når man skal diskutere løsninger med andre interessenter. Sekvens- og samarbeidsdiagrammer kan sees på som representasjoner av et enkelt scenario, og brukes derfor også ofte sammen med use case for å identifisere hvilke objekter og meldinger som trengs for at objektene i modellen skal samarbeide. [Trætteberg, Kompendium:1, 2004?]

Sekvens- og samarbeidsdiagram viser stort sett det samme, nemlig hvordan interaksjonen mellom de forskjellige objektene som inngår i et system foregår. En

bruker blir i disse diagrammene også redusert til et objekt som sender og mottar meldinger. Forskjellen mellom de to diagrammene ligger hovedsakelig i at sekvensdiagrammene viser den kronologiske rekkefølgen til meldingene, mens samarbeidsdiagrammet legger mer vekt på objektene i systemet og hvordan de kommuniserer med hverandre. [SmartDraw UML Center, Org:3, 200?]

I et sekvensdiagram plasseres objekter langs en horisonal akse, vanligvis ordnet etter hvem som aktiveres først. Meldinger mellom objektene vises som piler. Den vertikale aksen i modellen representerer tiden, slik at en hendelse som vises høyere i diagrammet skjer før en som ligger lavere. [Trætteberg, Kompendium:1, 2004?]

I samarbeidsdiagrammene er objektene representert i form av noder, og meldingene mellom dem representeres i form av nummererte kanter som viser rekkefølgen til meldingene. [Trætteberg, Kompendium:1, 2004?] Samarbeidsdiagrammer viser på denne måten både den statiske strukturen og den dynamiske oppførselen til et system [SmartDraw UML Center, Org:3, 200?]

Sekvens- og samarbeids diagrammer kan gjøres ganske komplekse ved å tegne inn betingelser og betingede hendelser. Siden dette har en tendens til å ødelegge diagrammets kommunikative rolle, er det imidlertid som oftest bedre å lage ett diagram per scenario som skal beskrives. [Trætteberg, Kompendium:1, 2004?]

Eksempel

Prosessen med å behandle en lånesøknad sendt via lånekalkulatorapplikasjonen, kan modelleres ved hjelp av henholdsvis sekvens- og samarbeidsdiagram slik:

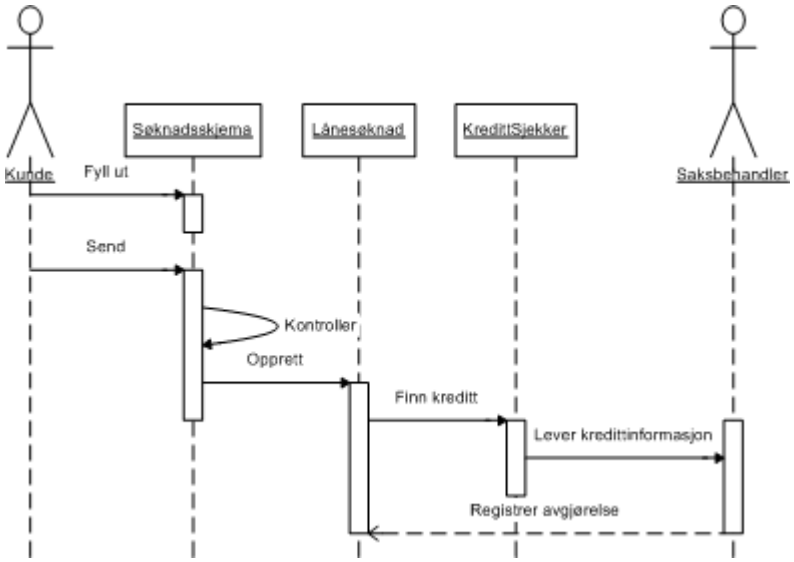


FIGURE 3-6: Sekvensdiagram for behandling av lånesøknad

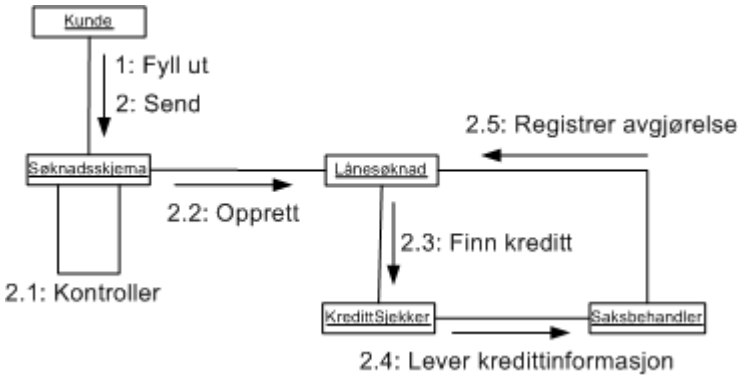


FIGURE 3-7: Samarbeidsdiagram for behandling av lånesøknad

Kritikk

Koblingen til systemets objekter gjør at sekvens- og samarbeidsdiagrammer må regnes som svært systemsentrerte. Det legges mer vekt på hvordan objektene i systemet samarbeider enn hvordan interaksjonen mellom system og bruker skal foregå. Brukt sammen med use cases er de imidlertid effektive når man skal spesifisere brukerkrav og krav til interaksjonen mellom et system og brukergrensesnitt.

Tilstandsmodellering

Når en bruker sender en hendelse inn til systemet via brukergrensesnittet må den korresponderende hendelseshåndtereren være i stand til å bestemme konteksten hendelsen oppstår i slik at rett aksjon kan utføres. [Horrocks, 1999]

Tilstandsmodeller forsøker å beskrive hvordan stadige endringer påvirker tilstanden i systemet. To modeller som har oppnådd stor utbredelse er Petrinett og Statecharts. De har vist seg å være særlig effektive til å modellere reaktive systemer hvor tilstanden stadig endres. [Horrocks, 1999] [van der Aalst, 1998]

Petrinett

Main Entry: **tran·si·tion**
Pronunciation: `tran(t)-'si-sh&n, tran-'zi-, chiefly British tran(t)-'si-zh&n`
Function: *noun*
Etymology: Latin *transition-, transitio,* from *transire*
1 a : passage from one state, stage, subject, or place to another **: CHANGE b :** a movement, development, or evolution from one form, stage, or style to another
2 a : a musical modulation **b :** a musical passage leading from one section of a piece to another
3 : an abrupt change in energy state or level (as of an atomic nucleus or a molecule) usually accompanied by loss or gain of a single quantum of energy
- **tran·si·tion·al** /-'sish-n&l, -'sizh-, -'zish-; -'si-sh&-n^{5}l, -'zi-, -zh&-/ *adjective*
- **tran·si·tion·al·ly** *adverb*

FIGURE 3-8: Fra Webster Online Dictionary: 'tran-si-tion'

Et petrinett modellerer et systems tilstand ved å ta utgangspunkt i dets transisjoner, det vil si hvordan tilstandene til systemet *endrer seg* som respons på stimuli. Dette gjør den i form av et nettverk bestående av 'lokasjoner' (places), 'transisjoner' og 'koblinger' (arcs). [van der Aalst, 1998]

Notasjon

I modellen vises inndatalokasjoner ved at de har koblinger som leder til en transisjon, mens utdatalokasjoner har koblinger som leder ut fra en transisjon. Systemets tilstand er i modellen gitt av antall 'markører' (tokens) som er plassert i petrinettets lokasjoner. Transisjoner viser alle aktiviteter som kan inntreffe (avfyres), og dermed endre systemets tilstand. En transisjon avfyres når alle betingelser for den gitte aktiviteten er oppfylt, det vil si at det er nok markører i transisjonens inndatalokasjoner. Når det skjer, fjernes markører fra disse lokasjonenene og plasseres i alle transisjonens utdatalokasjoner. Antall markører som legges til eller fjernes avhenger av okasjonens kardinalitet. En avfyring kan føre til en kjedereaksjon av avfyringer som forplanter seg gjennom nettverket, et såkalt 'token game'. Ved å flytte på markørene er det på denne mulig å simulere de dynamiske og sammenfallende transisjonene til systemet. [Zimerman, Nettside:8, 200?]

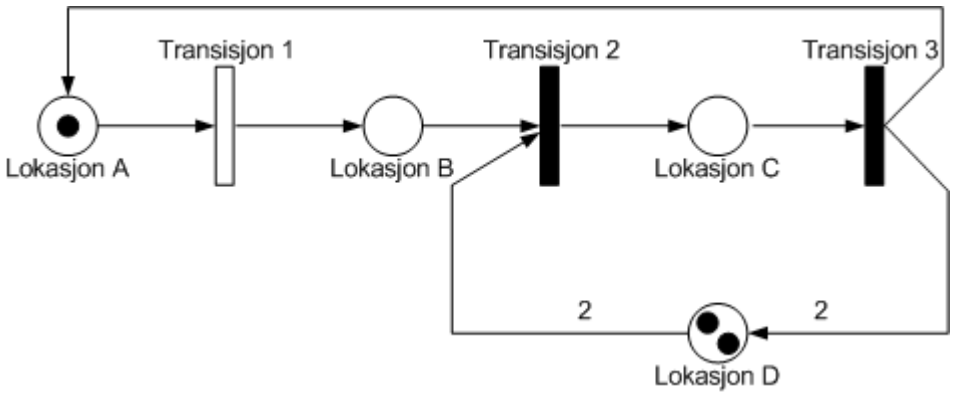


FIGURE 3-9: Notasjonen til Petrinett [van der Aalst, 1998]

I et petrinett er det en markant forskjell på det å muliggjøre en transisjon og det å avfyre den. For å holde orden på dette, benyttes begrepet 'avfyring' (triggering). I [van der Aalst, 1998] beskrives fire av de vanligste måtene en transisjon kan avfyres: 'Automatisk avfyring' skjer i det øyeblikk alle nødvendige markører er på plass i lokasjonen, dvs når den er muliggjort, og benyttes i de tilfeller hvor direkte brukerinteraksjon ikke er nødvendig. 'Brukeravfyring' skjer når systemet er avhengig av en input fra brukeren. 'Beskjedavfyring' skjer når systemet mottar en ekstern beskjed, f.eks svar på et databasekall. Og 'tidsavfyring' skjer når systemet er avhengig av at et gitt tidspunkt skal inntreffe, for eksempel automatisk lagring av et dokument. På denne måten er det også mulig å modellere konkurrerende tilstander. Dette skjer i de tilfeller to eller flere transisjoner er muliggjorte, men bare en kan avfyres. Da vil den transisjonen som utløses først også avfyres først. [van der Aalst, 1998]

Petrinett blir mye brukt for å beskrive og analysere systemer som karakteriseres ved å være sammenfallende, asynkrone, distribuerte eller parallelle. De er spesielt egnet som matematiske verktøy hvor det er mulig å sette opp tilstandslikninger, algebraiske ligninger og andre matematiske modeller som omhandler systemers oppførsel. [Zimerman, Nettside:8, 200?]

Statecharts

Main Entry: **1state**
Pronunciation: 'stAt
Function: *noun*
Usage: *often attributive*
Etymology: Middle English *stat,* from Old French & Latin; Old French *estat,* from Latin *status,* from *stare* to stand -- more at STAND
1 a : mode or condition of being <a *state* of readiness> **b** (1) **:** condition of mind or temperament <in a highly nervous *state*> (2) **:** a condition of abnormal tension or excitement
2 a : a condition or stage in the physical being of something <insects in the larval *state*> <the gaseous *state* of water> **b :** any of various conditions characterized by definite quantities (as of energy, angular momentum, or magnetic moment) in which an atomic system may exist
3 a : social position; *especially* **:** high rank **b** (1) **:** elaborate or luxurious style of living (2) **:** formal dignity **:** POMP -- usually used with *in*
4 a : a body of persons constituting a special class in a society **:** ESTATE 3 **b** *plural* **:** the members or representatives of the governing classes assembled in a legislative body **c** *obsolete* **:** a person of high rank (as a noble)
5 a : a politically organized body of people usually occupying a definite territory; *especially* **:** one that is sovereign **b :** the political organization of such a body of people **c :** a government or politically organized society having a particular character <a police *state*> <the welfare *state*>
6 : the operations or concerns of the government of a country
7 a : one of the constituent units of a nation having a federal government <the fifty *states*> **b** *plural, capitalized* **:** The United States of America
8 : the territory of a state

FIGURE 3-10: Fra Webster Online Dictionary: 'state'

Et statechart er representert som et nettverk bestående av tilstander og hendelser. Det modellerer den dynamiske oppførselen til et system og hvordan dette systemet reagerer på ekstern stimuli, og . Notasjonen er avledet fra tradisjonelle tilstandsdiagrammer og er utviklet spesielt med tanke på å spesifisere interaksjon mellom bruker og system. [Horrocks, 1999]

Notasjon

Selve notasjonen er basert på avrundede rektangler som representerer ‘tilstander’ (states) og og piler som representerer ‘transformasjoner’ (transformations). Pilene angir retningen på transformasjonen ved å peke fra en tilstand til en annen.

En tilstand kan defineres som enhver stabil form av et system. [Horrocks, 1999] Å bestemme hva som skal defineres som en systemtilstand, og hva som skal defineres som en tilstandsendring er derfor en viktig oppgave når man skal lage et statechart. [Horrocks, 1999]

Til en transformasjon hører det alltid til en ‘hendelse’ (events), som beskriver transformasjonen. I tillegg kan det være knyttet en eller flere ‘betingelser’ (conditions) som må være oppfylt for at en transformasjon skal inntreffe. Disse angis i paranteser sammen med hendelsen. Systemets start- og slutt-tilstander representeres som henholdsvis en fylt sirkel og en dobbeltsirkel hvor den innerste er fylt. [Horrocks, 1999]

Det kan finnes flere veier fram til en enkelt tilstand, og det kan være vanskelig å modellere dette presist. For å representere den forrige tilstanden systemet befant seg i, benyttes en åpen sirkel som inneholder tegnet “H” . En pil som peker inn til, eller ut av et slikt symbol indikerer at vi skal returnere til eller ta utgangspunkt i systemets forrige tilstand. [Horrocks, 1999]

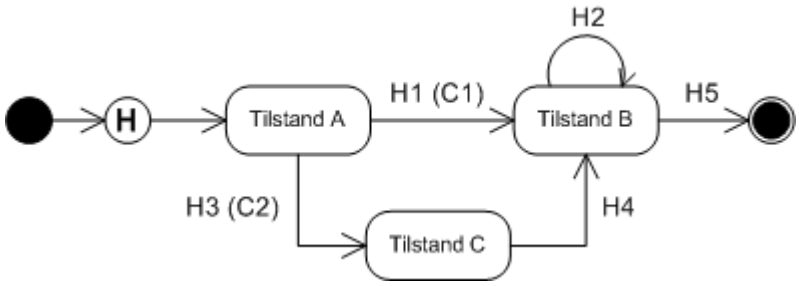


FIGURE 3-11: Notasjonen til statecharts

En tilstand kan ha et sett av subtilstander. Tilstand A kan for eksempel inneholde de nøstede tilstandene A1, A2, A3 og A4.

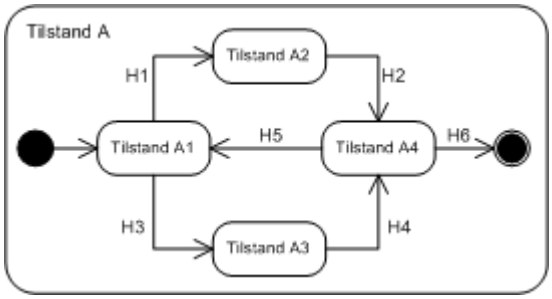


FIGURE 3-12: Subtilstander

Dette er en annen måte å spesifisere at når et objekt befinner seg i tilstand A, vil det også være i en av de fire spesifiserte undertilstandene.

Ofte innebærer en tilstand også en form for aktivitet. For eksempel kan “søk i database” eller “vis film” defineres som tilstander. Tilstand er altså ikke nødvendigvis det samme som stillstand. Som en tilstandsendring er det f.eks vanlig å regne at et nytt vindu kommer opp i brukergrensesnittet, eller at innholdet av en ramme i et brukergrensesnitt byttes ut med noe annet. Endringer som skjer *innenfor* et slikt vindu eller ramme, defineres da ikke til å endre systemets tilstand. [Horrocks, 1999]

Eksempel

Prosessen med å sende inn en lånesøknad og få denne behandlet kan modelleres ved hjelp av et statechart på denne måten:

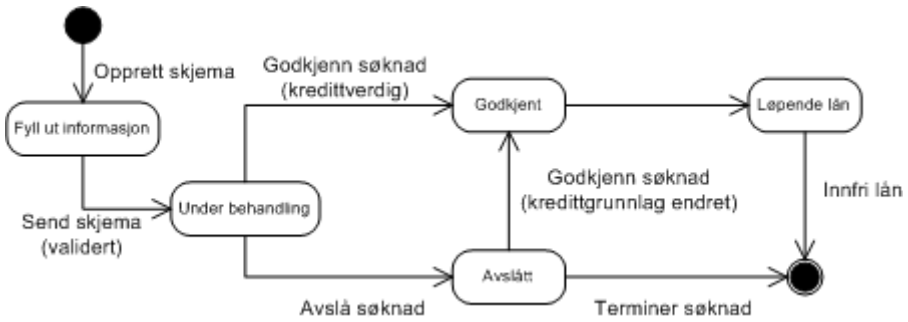


FIGURE 3-13: Statechart for behandling av lånesøknad

Notasjonen tar ikke hensyn til eventuelle aksjoner som blir utført når en transformasjon gjennomføres, men dette er mulig å beskrive i en tilstandstabell som følger statechartet. En slik tilstandstabell inneholder da en beskrivelse av den nåværende tilstanden; hvilke hendelser og evt. betingelser som er knyttet til transformasjonen; hvilken aksjon som skal utføres dersom en transformasjon inntreffer; og systemets neste tilstand. [Horrocks, 1999]

Kritikk

Begge de to tilstandsmodellene vi har beskrevet er systemsentrert, petrinett i enda større grad enn tilstandskart. De er uten tvil kraftige verktøy når man skal spesifisere tilstansendringer i systemet. Tilstandsendringer er imidlertid langt fra det samme som interaksjon. Modellene viser bare interaksjon på et helt overordnet plan, og da bare når systemets tilstand endres. De er verken i stand til å beskrive interaksjonen som skjer innad i et brukergrensesnitt eller mellom system og bruker.

Dialogmodellering

Main Entry: **[1]di·a·logue**
Variant(s): *also* **di·a·log** /'dI-&-"log, -"läg/
Function: *noun*
Etymology: Middle English *dialoge,* from Old French *dialogue,* from Latin *dialogus,* from Greek *dialogos,* from *dialegesthai* to converse, from *dia-* + *legein* to speak -- more at LEGEND
1 : a written composition in which two or more characters are represented as conversing
2 a : a conversation between two or more persons; *also* **:** a similar exchange between a person and something else (as a computer) **b :** an exchange of ideas and opinions **c :** a discussion between representatives of parties to a conflict that is aimed at resolution
3 : the conversational element of literary or dramatic composition
4 : a musical composition for two or more parts suggestive of a conversation

FIGURE 3-14: Fra Webster Online Dictionary: 'di-a-logue'

De interaksjonsmodellene vi har sett på hittil har alle vært relativt systemsentrerte. De fokuserer stort sett på interaksjonen mellom de forskjellige objektene som inngår i et system, og hvordan brukerens handlinger påvirker denne interne informasjonsutvekslingen.

Generelt for dialogmodeller gjelder det at de er representasjoner på brukergrensesnittets interne informasjonsflyt og dets respons på brukerens handlinger. [Trætteberg, 2002:1] De beskriver dialogen, eller meldingsutvekslingen, som foregår mellom en bruker og et system, og blir dermed en analyserbar spesifikasjon på all interaksjon som er relevant, *sett fra brukerens ståsted*. En dialogmodell kan dermed sies å være relativt brukersentrert, i forhold til de andre interaksjonsmodellene vi har sett på.

I prosessen med å designe et nytt brukergrensesnitt kan dialogmodellen brukes som en beskrivelse på hvordan designløsningen må tillate en bruker og et system å samhandle for at oppgavene skal kunne utføres og brukerens mål bli oppnådd.

3.4 DiaMODL

Hva er DiaMODL?

DiaMODL er et dialogmodelleringsspråk som gjør det mulig å representere hvordan informasjonen flyter mellom bruker og system via et brukergrensesnitt. Modellen er i stand til å beskrive både dataflyten mellom interaksjonsobjektene som utgjør brukergrensesnittet, og funksjonaliteten og oppførselen til hver enkelt av disse. [Trætteberg, 2002:1] [Trætteberg, 2003]

Interaktorbegrepet

Ideen bak DiaMODL er å spesifisere interaksjonen i et brukergrensesnitt ved å benytte interaktorer og abstraherte interaksjonsobjekter[1], sammen med en interaksjonsstruktur som binder interaktorene sammen med hverandre og med det underliggende funksjonalitetslag.

En interaktor blir slik en spesifikasjon på den spesifike input og output-funksjonaliteten som er implementert i en konkret, grafisk brukergrensesnittkomponent. Med andre ord kan en interaktor defineres som en generisk overbringer av informasjon fra systemet til brukeren og omvendt, dvs at en interaktor overfører både input fra brukeren til systemet, og output fra systemet til bruker. [Trætteberg, 2002:1]

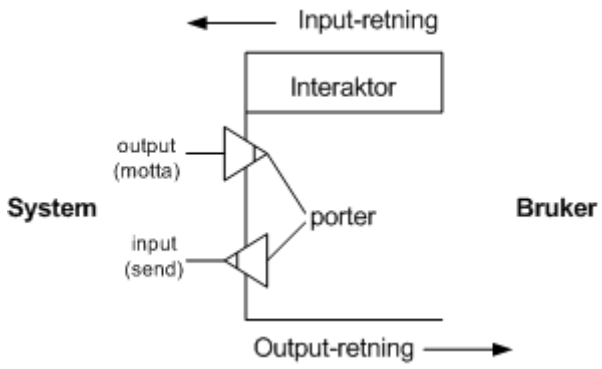


FIGURE 3-15: Interaktorkonseptet [Trætteberg, 2002:1]

En interaktor kan også sees på som en spesifikasjon på et designproblem som skal løses. Interaktorer blir da abstrakte, generiske representasjoner av de konkrete brukergrensesnittkomponentene. Et eksempel på en slik interaktorkomponent er en 'selektor', som rent konkret kan være en rullegardin, en radioknappgruppe, eller lignende. Disse interaktorkomponentene har den samme generiske strukturen, men funksjonaliteten, utseendet og bruksområdet er forskjellig. På denne måten er det mulig å bruke interaktorer som byggesteiner i abstrakt design og analyse av grensesnittstruktur og oppførsel. [Trætteberg, 2002:1]

Tre egenskaper som definerer interaktorers funksjonalitet og oppførsel.

Tre viktige egenskaper definerer en interaktors funksjonalitet og oppførsel: Interaktorer definerer selve informasjonen i seg selv, hvordan den skal overføres og

1. I oppgaven benyttes også begrepet 'interaktorkomponenter'

hvilken retning den flyter; interaktorers oppførsel kontrolleres av et internt tilstandsrom, som inkluderer sending og mottaking av informasjon og aktivering av andre interaktorer; og interaktorer kan settes sammen til nye interaktorer, slik at det er mulig å definere høynivå oppførsel og abstrahere vekk detaljer. [Trætteberg, 2002:1]

Notasjon

Et DiaMODL-diagram består av interaktorer og en struktur som binder disse sammen. Selve informasjonsoverføringen defineres av: "Et sett porter som definerer interaktorenes eksterne grensesnitt til systemet, brukeren og andre interaktorer; et sett koblinger som bærer data mellom portene; og en kontrollstruktur som utløser eller forhindrer dataflyt mellom porter på interaktorens inn- og utside og langs koblinger." [Trætteberg, 2002:1] (s 85 - oversatt)

Interaktorer

Interaktorer er representert i form av en boks, der høyre side er åpen. Dette er for å illustrere at interaktoren er åpen for interaksjon mot brukeren. Boksen har en header hvor interaktoren navngis. På høyre side kommuniserer interaktoren med resten av systemet via et grensesnitt definert av portene. Denne notasjonen gjør at en interaktor i prinsippet er uavhengig av dets implementasjon og blir dermed en spesifikasjon på en gitt *oppførsel*. [Trætteberg, 2002:1]

Verken systemet som interaktoren sender data til eller brukeren den mottar data fra, er eksponert for noe av interaktorens implementasjon. Ved å sette sammen interaktorer og definere dataflyten mellom dem, kan man dermed modellere et system der brukergrensesnittet består av standard brukergrensesnittkomponenter. [Trætteberg, 2002:1] [Mittet, 2006]

Kontrollstruktur

Kontrollstrukturen i notasjonen håndterer utløsing av informasjonsflyt, samt aktivering og deaktivering av av interaktorer. Hver interaktor er egentlig en Statechart supertilstand, som gås inn i når interaktoren aktiveres. [Trætteberg, 2002:1] En interaktor kan dekomponeres i et hierarki av deltilstander, som beskrevet i statechartnotasjonen, hvor hvert nivå i interaktorhierarkiet korresponderer til minst et nivå i statechartet. Når en interaktor aktiveres, tilsvarer det at den går inn i en tilstand og omvendt. I aktiv tilstand er portene klare til å motta og sende verdier og den indre kontrollstrukturen utføres. [Trætteberg, 2002:1]

Porter

En interaktor kommuniserer med omverdenenen via et sett porter. Porter leder informasjonsflyt enten fra system til bruker, eller omvendt, via brukergrensesnittets konkrete interaksjonsobjekter. Konseptet om porter er basert på et underliggende funksjonskonsept som tar et sett argumenter og beregner en utverdi.

En port kan ha kun en inputverdi, og er delt inn i en ‘base’ og en ‘tupp’. Datatypen må være den samme i basen og tuppen, men verdien kan variere. I noen tilfeller er verdien i tuppen utregnet ut fra basens verdi. Hver port er festet til en interaktors venstre side, og basen og tuppen vil dermed befinne seg på forskjellig side av interaktorens grense. [Trætteberg, 2002:1]

FIGURE 3-16: Interaktorkomponent

Objekter og objektsett

Dataverdiene som transporteres inn til eller ut av en interaktor kan komme enten i form av et objekt eller i form av et objektsett. I notasjonen angis dette som hhv en lukket og en overlappende boks der objekttypen og objektsettypen er beskrevet i form av en tekstbeskrivelse. Slike objekter og objektsett regnes som en del av interaktoren.

Ved hjelp av disse elementene er det mulig å beskrive ikke bare hvilken datatype interaktoren benytter i sin kommunikasjon, men også hvilken type interaktorkomponent det er snakk om.

FIGURE 3-17: Basisfunksjoner i DiaMODL

Funksjoner

Strukturen med sammenkoblede porter er istand til å transportere informasjon mellom brukeren og systemet, men tilbyr ingen måte som gjør det mulig å beregne nye verdier. Den generiske funksjonen er en spesialisert variant av en port som benyttes når dette er nødvendig. En funksjon skiller seg fra en port ved at den kan ha flere innverdier og ikke er i stand til å transportere verdier over grensa til en interaktor. [Trætteberg, 2002:1]

Mer kompleks oppførsel enn beregning av en enkelt verdi indikeres ved at tuppen og basen til funksjonssymbolet er atskilt. Dersom resultatet av en funksjon ikke blir brukt lokalt, kan dette indikeres ved at tuppen utelates helt i notasjonen. [Trætteberg, 2002:1]

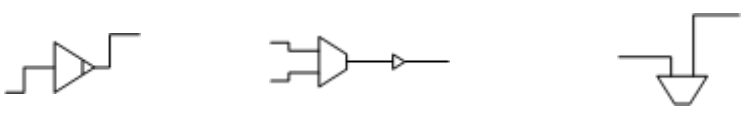


FIGURE 3-18: Basisfunksjoner i DiaMODL

Koblinger

Koblinger sikrer at dataverdier transporteres mellom interaktorene, slik at utdata kan presenteres for brukeren og inndata kan mates til systemet. Ettersom basen og tuppen på porter og funksjoner er innskrenket til kun å ha en datatype, er det mulig å verifisere koblingene for å sikre konsistens på et syntaktisk plan [Trætteberg, 2002:1]

En kobling gir et argument av en bestemt datatype til porten eller funksjonen den leder til. En kobling som leder fra tuppen på porten eller funksjonen tar med en ferdigbehandlet verdi av samme datatype videre.

I enkelte tilfeller er det nødvendig å vise at en funksjon eller metode venter på input fra en bruker før den ferdigbehandlede verdien sendes videre. Dette indikeres ved at en kobling leder inn til siden av funksjonen. Slike koblinger benevnes som ‘avtrekkere’.

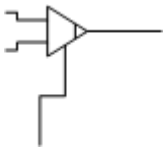


FIGURE 3-19: Funksjon med avtrekker

Metoder

I denne oppgaven utvides notasjonen med en spesialisert funksjon som vi velger å kalle en ‘metode’. En metode er en funksjon som utfører en endring på et objekt eller objektsett. Dette indikeres i notasjonen ved at metodens tupp leder bort til kanten på objektet eller objektsettet.

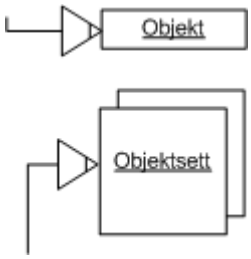


FIGURE 3-20: Metoder

Super- og subinteraktorer

Ofte vil en komme opp i situasjoner der en interaktor består av flere interaktorobjekter. Dersom det er nødvendig å spesifisere funksjonaliteten til disse subinteraktorene plasseres de inne i superinteraktoren.

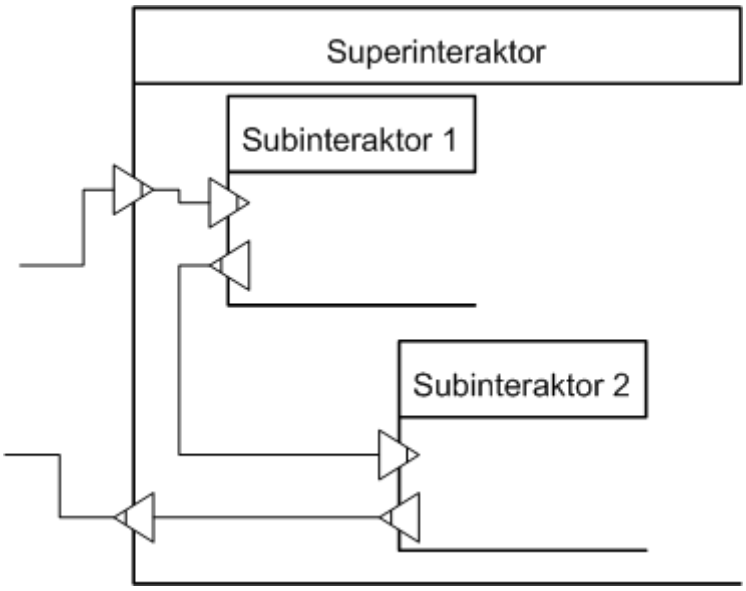


FIGURE 3-21: Super- og subinteraktorer

Noen fundamentale interaktorkomponenter

Når interaksjonen til konkrete brukergrensesnittkomponenter skal abstraheres, vil interaktorene som representerer dem spenne fra de helt enkle til svært komplekse.

Presentere av et objekt

Den aller enkleste interaktoren er en presentasjonsinteraktor. Den har kun en port for å levere data til interaktoren, og denne porten er koblet til et enkelt objekt som skal presenteres i brukergrensesnittet. Dette kan f.eks være en tekst, et bilde eller en videosnutt.

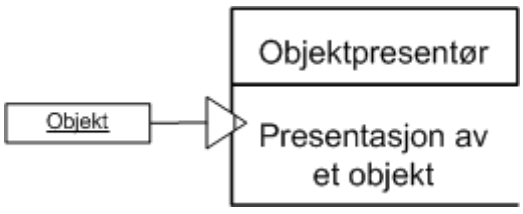


FIGURE 3-22: Presentasjon av et objekt

Manipulere et objekt

En interaktor som i tillegg har en kobling til et objekt via en input/send port, henter inn et objekt og er i stand til å manipulere dette for så å sende det tilbake til systemet. Dette kan f.eks være et tekstfelt eller en avkrysningsboks.

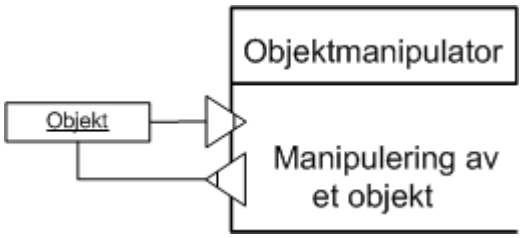


FIGURE 3-23: Presentasjon og manipulasjon av et objekt

Seleksjonsinteraktorer

En interaktor presentere en liste av elementer som en bruker så kan velge ut et enkelt element fra. Eksempler på slike brukergrensesnittkomponent er rullegardiner, sett radioknaper eller lister med lenker.

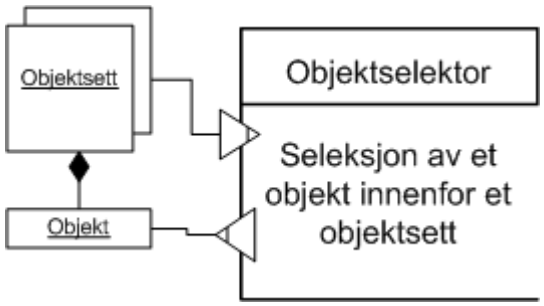


FIGURE 3-24: Seleksjon av et enkelt objekt fra et objektsett

Et annet eksempel er en interaktor som lar brukeren velge flere objekter innenfor et objektsett, altså en multippel seleksjon. Et eksempel på et slikt konkret brukergrensesnittobjekt er en filliste der det er mulig å markere mer enn en fil samtidig.

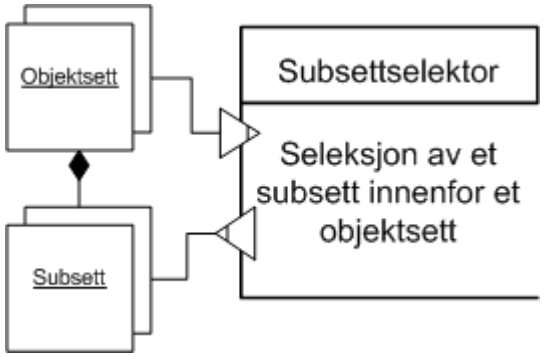


FIGURE 3-25: Seleksjon av et subsett innenfor et objektsett

Ofte har man behov for å beskrive en interaktorkomponent der hvilket objekt som skal presenteres for brukeren avhenger av et annet objekt eller objektsett. En slik situasjon oppstår når brukeren skal velge et objekt fra et sett mens innholdet av dette settet samtidig blir bestemt ut fra et tidligere valg brukeren har foretatt.

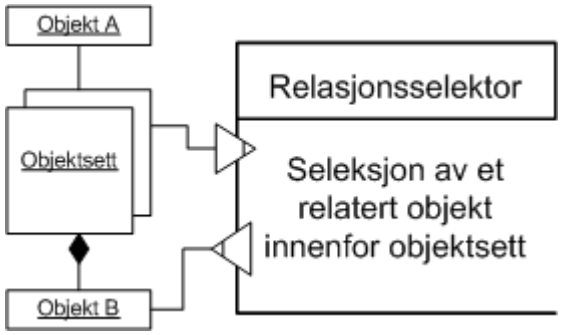


FIGURE 3-26: Seleksjon av et relatert objekt innenfor et objektsett

Et siste eksempel er en hierarkistisk seleksjon, der brukeren må velge et enkelt objekt fra en hierarkistisk struktur. Brukeren blar seg da gjennom en eller flere undernoder for å velge ut et enkelt objekt. Et katalogtre er et konkret eksempel på dette.

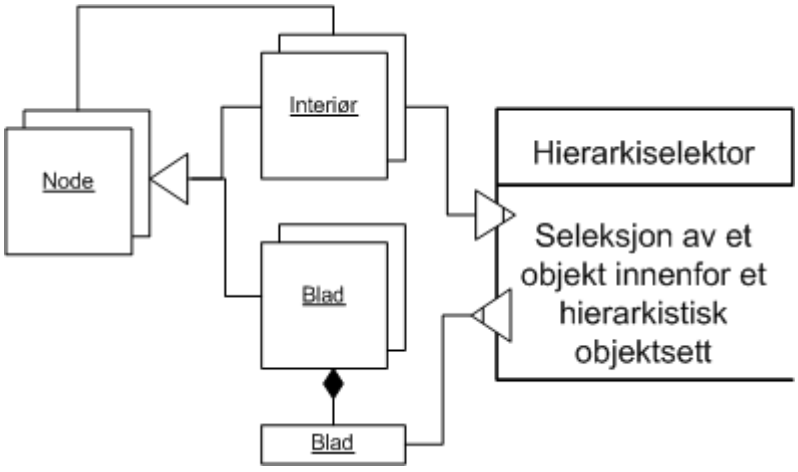


FIGURE 3-27: Seleksjon av et element innenfor et hierarkistiskobjekt sett

Eksempel

En nettleser er en applikasjon som henter et dokument som ligger på en oppgitt url. Dette prinsippet kan i DiaMODL modelleres på denne måten:

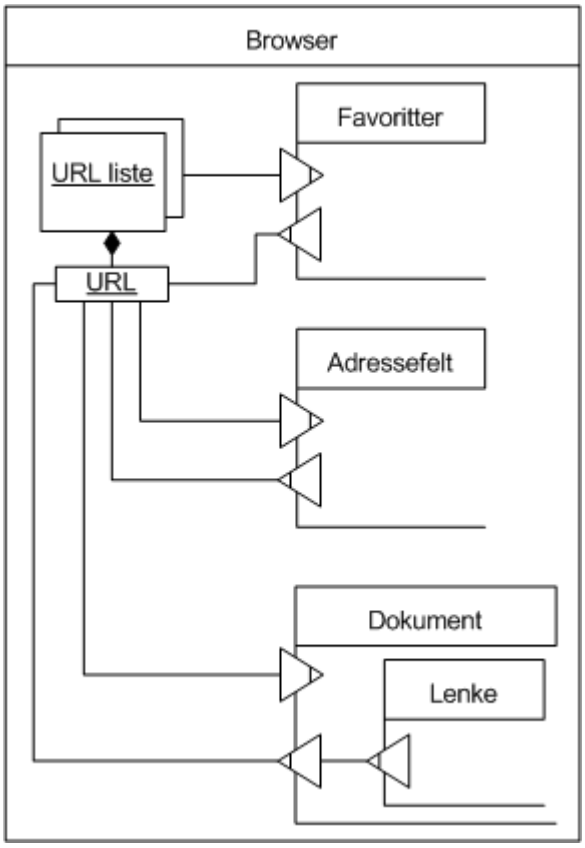


FIGURE 3-28: En enkel nettleser

Brukerens favorittlenker ligger lagret i en liste. Lista presenteres for brukeren, som så plukker ut en enkelt url. Denne urlen sendes til adressefeltet, som oppdateres slik at den gjeldende urlen vises i tekstform, samtidig som det dokumentet som ligger på denne urlen lastes inn i nettleseren og presenteres for brukeren. Når brukeren trykker på en lenke i det gjeldende dokumentet, vil også adressefeltet oppdateres og dokumentet på denne urlen lastes inn. Dersom brukeren skriver inn en url manuelt, vil dette føre til at dokumentet på denne urlen lastes inn i nettleseren.

Bruksterskel kontra brukshøyde

DiaMODL har en relativt høy bruksterskel, men brukshøyden er også potensielt veldig høy. Den abstrakte tilnærmingen gir muligheter for å spesifisere et systems brukergrensesnitt på et høyt nivå. Selv om dette også kan føre til at designeren mister kontroll over detaljene, er ikke dette nødvendigvis et problem i denne konteksten. Den høye formaliteten åpner for utvikling av verktøy som kan gjøre det mulig å generere kode ut fra modellen. Samtidig er modellen såpass spesialisert, at den kan unngå ei felle som har felt flere modellbaserte brukergrensesnittdesignmetoder, nemlig det at de ofte prøver å fange hele domenet på en gang. [Campos et. al., 2004]

Fordeler og ulemper med DiaMODL-notasjonen

DiaMODL sitt største fortrinn er måten den beskriver brukergrensesnittets interne dataflyt og tilstander. Brukeren er ikke redusert til et objekt, men betraktes som en reaktiv faktor som påvirker systemet dynamisk.

Fordelen med å beskrive interaksjon ved hjelp av den detaljerte strukturen som benyttes i DiaMODL er at den utvetydig viser dataflyten mellom interaktorene, både med hensyn på retning, innhold og sammenheng mellom interaktorene. Bakdelen er at interaksjonen raskt kan bli uoversiktelig å lese ut fra diagrammet og at det er vanskelig å beholde oversikten, særlig når man skal beskrive innfløkte brukergrensesnitt. Dette gjør også at DiaMODL-diagrammer kan være vanskelig å tegne opp med penn og papir.

Systemets interaksjon med brukeren er beskrevet på et helt elementært nivå, slik at vi kan se hvilke data som brukeren sender inn og hvilke systemet gir tilbake. I mange tilfeller er det ønskelig å presisere hvordan brukerinteraksjonen skjer, og hvilke komponenter som støtter en slik interaksjon. Å lese slik informasjon ut fra et DiaMODL-digram krever inngående kunnskap om notasjonen og om hvilke kontrollstrukturer som overlapper med hvilke spesifike brukergrensesnittkomponenter.

DiaMODL har relativt streng formalitet, noe som muliggjør utvikling av dataverktøy for å generere brukergrensesnitt eller kode automatisk. Dette gjør imidlertid notasjonen ganske abstrakt, noe som også bidrar til at det er det tungt å sette seg skikkelig inn i modellen.

DiaMODL kan være et nyttig verktøy for å dokumentere og spesifisere funksjonaliteten og strukturen til konkrete brukergrensesnitt. DiaMODL kan nok også fungere bra som et verktøy for å analysere en ferdig designløsning. Tungvint notasjon gjør nok at DiaMODL egner seg lite til å utforske forskjellige designløsninger, og vil derfor ikke være spesielt brukbar i praktisk design.

3.5 Oppsummering

Interaksjonsmodellen skal først og fremst støtte systemutviklingsprosessen, og bør bestrebe og få til en glidende overgang fra en designløsning til noe som kan programmeres.

De fleste interaksjonsmodeller er svært systemsentrerte. Dette vises ved at interaksjonen mellom de forskjellige objektene internt i systemet er i de fleste modellene godt ivaretatt, mens brukeren reduseres til et objekt som sender og mottar meldinger til de andre objektene i systemet. Den dynamiske, reaktive interaksjonen mellom bruker og system blir i liten grad fanget opp. Bare dialogmodellen, her representert ved DiaMODL, har adressert dette problemet og kommet med en løsning som betrakter brukeren som noe mer enn et objekt.

Det er naturlig å benytte DiaMODL-notasjonen som utgangspunkt for en hybrid notasjon av en interaksjonsmodell og -prototyp. Interaktorbegrepet er relativt intuitivt, dataflyten er tydelig og logisk beskrevet og den er i stand til å spesifisere, i form av funksjoner, dynamikken som påvirker brukergrensesnittets innhold og presentasjon. Dette er egenskaper det er viktig å ta med seg videre.

Kapittel 4

Interaksjonsprototyping

Dette kapittelet tar for seg prototyping i forbindelse med interaksjonsdesign; hva som er definisjonen på en interaksjonsprototyp og hva målet med den er. Videre beskriver det hvordan prototyper kan klassifiseres ut fra sin hensikt og analyseres langs fire dimensjoner, før det går inn på temaet papirprototyper. Til slutt fokuserer det på den kanonisk abstrakte prototypen, som både kan betraktes som en konkret designartefakt og en modell av arkitekturen til det brukergrensesnittet som skal designes.

4.1 Hva er en interaksjonsprototyp[1]?

Main Entry: **pro·to·type**
Pronunciation: 'prO-t&-"tIp
Function: *noun*
Etymology: French, from Greek *prOtotypon,* from neuter of *prOtotypos* archetypal, from *prOt-* + *typos* type
1 : an original model on which something is patterned : **ARCHETYPE**
2 : an individual that exhibits the essential features of a later type
3 : a standard or typical example
4 : a first full-scale and usually functional form of a new type or design of a construction (as an airplane)

FIGURE 4-1: Fra Webster Online Dictionary: 'pro-to-type'

En prototyp er et verktøy

En interaksjonsprototyp er et verktøy som brukes for å besvare spørsmål rundt spesifikasjon og design. Et vesentlig poeng i denne sammenhengen er at prototypens form er ganske uvesentlig. Hvilke midler, materialer eller verktøy som benyttes for å lage prototypene har også liten betydning. Det essensielle med prototyper er hvordan de benyttes av interessentene for å utforske eller demonstrere spesifike aspekter ved det fremtidige systemet. [Houde et. al., 1997]

1. Betegnelsen 'interaksjonsprototyp' benyttes vanligvis for å betegne en prototyp som kjører på en datamaskin og har implementert en viss funksjonalitet. I denne oppgaven kommer både denne betegnelsen og betegnelsen 'prototyp' til å bli brukt som en samlebetegnelse om alle varianter av prototyper som brukes for å undersøke eller teste interaksjon. I stedet betegnelsen 'flash-prototyp' om undersøkende, kjørende prototyper..

Når brukes prototyping?

Innen kreativ design har prototyping tradisjonelt blitt brukt for å uttrykke ideer og reflektere over dem. Dette er en temmelig intuitiv tilnærming, orientert mot utforsking av designrommet og generering av nye ideer. [Beaudouin-Lafon et. al., 2003]

Systemutviklere har stort sett benyttet prototyping for å teste og studere gjennomførbarheten til en teknisk prosess. [Beaudouin-Lafon et. al., 2003] Prototyper blir altså i denne situasjonen et verktøy rettet mot analysering og evaluering av et system.

Et viktig bruksområde for interaksjonsprototyper er brukbarhetstesting. Dette er en teknikk for å sikre at et planlagt system møter brukerens krav og forventninger. Etter ISO 13407-standarden er det viktig at hver iterasjon ender i en testbar prototyp slik at designet og prosessen kan evalueres før neste iterasjon tar til.

Ett emne, mange definisjoner

Modeller definerer gjennom abstraksjon de underliggende mekanismer som styrer funksjonaliteten til et system. Prototyper tar derimot sikte på å undersøke konkrete egenskaper ved en designløsning. Selv om abstraksjonsgraden i notasjonen som benyttes kan variere, regnes de allikevel som konkrete representasjoner ettersom målet er å finne konkrete svar om en design gjennom testing og utforskning.

Det finnes antakelig like mange definisjoner på hva en prototyp er, som det finnes artikler som tar for seg temaet. I følge [Houde et. al., 1997] (s369) er en prototyp “en hver representasjon av en designide, uansett medium”. Denne definisjonen favner imidlertid over et alt for stort område til å kunne brukes i denne oppgaven, siden de fleste modeller også vil kunne passe inn under den.

[Beaudouin-Lafon et. al., 2003] (s1007) definerer “en prototyp som en konkret representasjon av en del av eller et helt interaktivt system ... en håndgripelig artefakt, ikke en abstrakt representasjon som trenger å tolkes.” Denne oppgaven vil imidlertid vise at en slik definisjon ikke gjelder alle prototyper. Selv om en prototyp som regel er mer konkret og håndgripelig enn en modell som beskriver det samme domenet, er en prototype svært ofte en forenkling av et ferdig system eller en mer avansert prototyp. En forenkling av noe betyr at prototypen må ha vært gjennom en abstraksjonsprosess. Og en abstrahert representasjon av noe må nødvendigvis tolkes i mer eller mindre grad av adressaten for å kunne forstås.

I [Preece et. al., 2002] (s. 241) defineres en prototyp til å være “en begrenset representasjon av et design som tillater brukere å samhandle med det og utforske dets skikkethet”. Denne definisjonen er mer i henhold til problemstillingen i denne oppgaven, men går litt for langt ettersom det finnes prototyper som er såpass abstrakte at det er vanskelig for brukere å samhandle med dem.

Denne oppgavens definisjon

I denne oppgaven defineres derfor en prototyp som “enhver begrenset representasjon som tillater interessenter å undersøke interaksjonen til en konkret designløsning og måle dens skikkethet gjennom testing eller utforskning”.

Mål med prototyper

Det mest opplagte målet med prototyper er å lede fram til en vellykket design. Klarest trer dette fram i de tilfeller hvor vellykkede programvareprototyper utvikler seg til et ferdig produkt. [Beaudouin-Lafon et. al., 2003] I de tilfeller hvor en prototyp blir kastet etter at dens hensikt er oppfyllt, vil den allikevel ha vært et viktig steg fram mot dette målet. Dette gjelder uansett om prototypen avkreftet eller bekreftet den designløsningen den representerte.

En avslørende representasjon

Prototyper kan avsløre både styrker og svakheter til et design. De er i stand til å besvare spørsmål knyttet til både design og tekniske krav og støtter designere i å velge mellom forskjellige alternativer. De kan brukes til å teste den tekniske gjennomførligheten av en ide, avklare vage krav eller sjekke at designretninger er kompatible med resten av systemutviklingen. [Preece et. al., 2002]

En reflekterende representasjon

Prototyper har den egenskapen at de kan analyseres i lys av systemets egen brukskontekst, i motsetning til rene ideer, abstrakte modeller og andre representasjoner som må analyseres på abstrakt vis med tradisjonell kravanalyse. [Beaudouin-Lafon et. al., 2003] På denne måten kan man se hvordan et system vil bli brukt i en reel situasjon, og prototypen blir da også et hjelpemiddel for designeren når brukerens krav og behov skal analyseres. Også selve prosessen rundt det å prototype et design kan hjelpe designere å reflektere over dette, ettersom de gjennom denne aktiviteten får tenkt gjennom problemstillingene rundt det designet de ønsker å framstille. [Beaudouin-Lafon et. al., 2003]

En kommunikativ representasjon

Et tredje aspekt med prototyper, er den kommunikative rolle de spiller når man diskuterer ideer med andre interessenter i et systemutviklingsprosjekt. Det er som regel lettere å diskutere design over en konkret artefakt enn over abstrakte modeller og tekstlige spesifikasjoner. I tillegg tjener selve prototypeaktiviteten en viktig hensikt ved å fremme refleksjon i designen. [Preece et. al., 2002]

Et vitenskapelig verktøy

Den empiriske metode går ut på at man bare kan motbevise en hypotese, ikke bevise den. Dette er også et av grunnprinsippene i prototyping. I [Preece et. al., 2002] presenteres en påstand om at brukere sjelden er i stand til å fortelle eksakt hva de vil ha, men at når de ser noe, og blir kjent med det, vet de snart hva de *ikke* vil ha.[1] På denne måten kan en bruker motbevise designerens hypotese om hva han mener er et godt design. Prototyping kan derfor i denne sammenhengen sies å være et vitenskapelig verktøy for utforskning av designen til et interaktivt informasjonssystem.

1. Selv om brukere også kan bekrefte et design, er det ikke alltid at dette er det *beste* designet. Brukere har ikke alltid forutsetninger til å forstå hva de vil ha eller hva de egentlig trenger.

4.2 Interaksjonsdesign og prototyping

Prototyping er en svært sentral aktivitet i den iterative designprosessen. ISO 13407 stadfester at hver iterasjon bør ende i en prototyp slik at iterasjonen kan evalueres. [Tajakka, Nettside:6, 2005]

Prototyper som designartefakter

En prototyp kan i likhet med modeller og spesifikasjoner sees på som et resultat av designprosessen: I rollen som designartefakt støtter prototypen kreativitet ved å hjelpe utvikleren å fange og generere ideer. Den letter utforskningen av design-rommet og avdekker relevant informasjon om brukerene og deres arbeidsrutiner. Den oppmuntrer til kommunikasjon ved å gi designere, utviklere, ledelse, kunder og brukere et fokus ved diskusjon rundt designen.

Prototyper gir brukere og andre interessenter muligheten til å samhandle med, og erfare konturene av, et planlagt system. For å få til dette må prototypen til en viss grad være interaktiv, dvs brukeren må kunne få en respons på de aksjoner han eller hun foretar seg mot systemet. I de tidligste utviklingsstadiene kan en slik interaktiv prototyp være laget av primitive materialer som papp og papuir og/eller være relativt abstrakte i formen ved at for eksempel bare rammeverket til en tenkt design er tegnet opp og alle komponenter bare er grovt skissert. Responsen fra systemet kan med slike prototyper simuleres etter Wizard of Oz-metoden.

Prototyper gjør det i tillegg mulig å evaluere et design på et tidlig tidspunkt, ved at den gjennom hele designprosessen tillater forskjellige former for testing, inkludert tradisjonelle brukbarhetstester og uformelle tilbakemeldinger fra brukere. Slik kan designeren på et relativt tidlig stadium i designprosessen få muligheten til å studere bruken av systemet i en realistisk setting, utforske forskjellige bruksområder, og identifisere funksjonelle krav og problemer knyttet til brukbarhet og oppførsel. [Beaudouin-Lafon et. al., 2003] [Preece et. al., 2002]

Etter hvert som designprosessen så skrider framover, og ideer blir klarere og mer detaljerte, benyttes ofte mer polerte artefakter som ligner mer og mer på det endelige produktet. [Preece et. al., 2002]

Domenemodellen utvikler seg

Et viktig poeng ved prototyping er at domenemodellen utvikler seg etter hvert som designrommet innskrenkes. En prototyp er av natur mer løsnings- enn problemorientert. Arbeidet med å lage en konkret representasjon vil tvinge designeren til å tenke gjennom problemstillingen på en annen måte enn hva som er tilfellet med modeller.

Horisontale vs. vertikale prototyper

Interaksjonsprototyper brukes primært til å undersøke brukbarheten til et system. Systemets brukbarhet avhenger i bunn og grunn av hvordan egenskaper knyttet til brukergrensesnittets visuelle design og hvordan systemets funksjonalitet oppleves av en bruker.

Prototyper designet kun for å teste et systems brukergrensesnittegenskaper, dvs besvare spørsmål rundt visuell brukbarhet, interaksjon og generelle designideer, sier vi er undersøkende i horisontal retning. En horisontal prototype ser ut som et ferdig produkt, men mangler funksjonaliteten som støtter selve interaksjonen. [Preece et. al., 2002] Skjermbilder av et system slik vi ofte finner i brukermanualer og kravspesifikasjoner, er eksempler på dette.

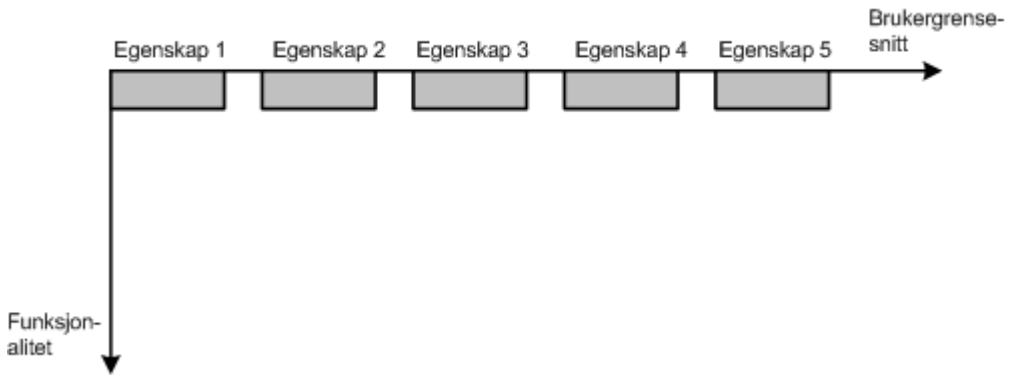


FIGURE 4-2: Skjematisk framstilling av fokuset til en horisontal prototyp

Prototyper designet for å undersøke eller teste den eksakte funksjonaliteten til en enkelt egenskap ved systemet, altså om den planlagte funksjonaliteten lar seg gjennomføre og hvilke konsekvenser dette vil ha for interaksjonen, er undersøkende i vertikal retning. En vertikal prototype har dermed implementert en eller flere deler av systemets funksjonalitet, men brukergrensesnittet til systemet er ikke klart definert. [Preece et. al., 2002] En prototyp som undersøker funksjonaliteten til en pop-up wizard, uten å implementere andre deler av systemet, er et eksempel på en fullstendig vertikal prototyp.

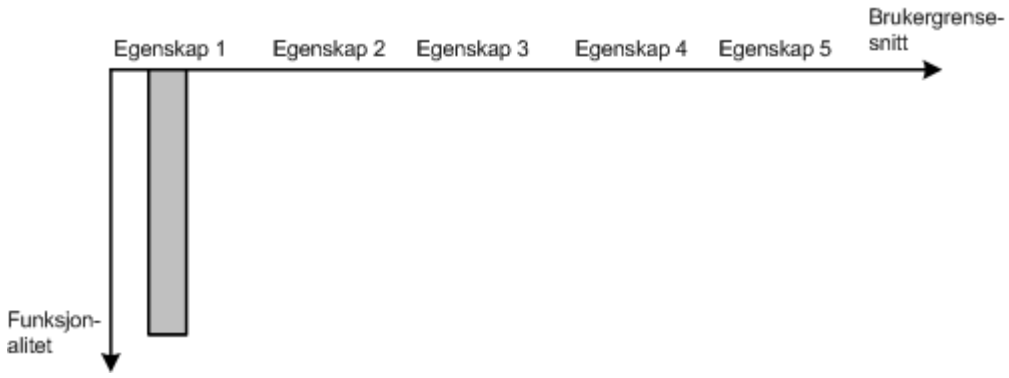


FIGURE 4-3: Skjematisk framstilling av fokuset til en vertikal prototyp

De fleste prototyper er imidlertid en blanding av vertikal og horisontal. Prototyper som lages etter noen iterasjoner i designprosessen vil typisk ha et presist definert utseende på en del områder, mens andre er mer uklare. Bare sjelden vil slike pro-

totyper ha all funksjonalitet ferdigdefinert for noen av egenskapene, men til gjengjeld vil noe funksjonalitet være definert på de fleste områder.

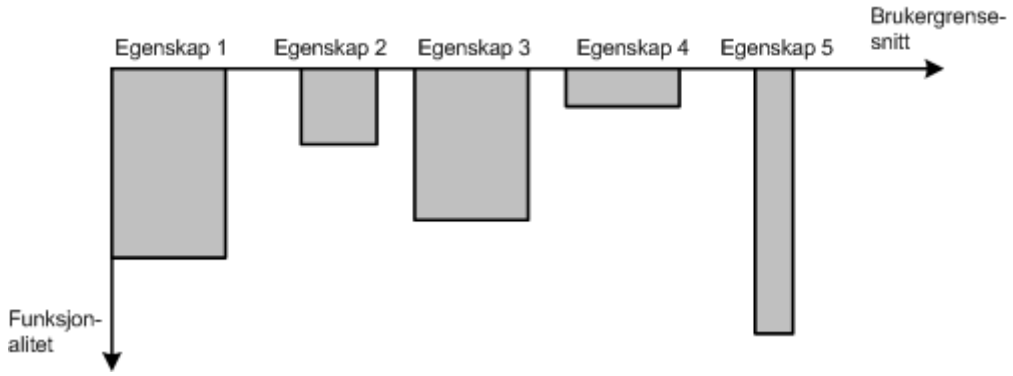


FIGURE 4-4: Skjematisk framstilling av fokuset til en typisk interaksjonsprototyp

Et tidlig utkast til et system vil imidlertid sjelden være særlig godt definert verken i horisontal eller vertikal retning. Funksjonaliteten til systemet er kanskje fortsatt svært uklart, og utseendet består ofte for det meste av hovedvinduer og noen få hovedkomponenter, skissert i relativ størrelse og plassering i forhold til hverandre. I slike prototyper vil da følgelig også interaksjonen være udefinert. En prototyp laget i de siste iterasjonene vil på den annen side ha både utseende og funksjonalitet klart definert på de fleste områder. Følgelig vil da også det meste av interaksjonen være på plass.

4.3 Analysering av prototyper

Presisering av noen begreper benyttet i prototyping

Ofte benyttes begrepene ‘lofi’ og ‘hifi’ som et mål på hvor polert en prototyp er, altså en betegnelse på utseendet til prototype og hva slags materiale den er bygd opp av [Preece et. al., 2002] Men lofi og hifi brukes også mange ganger som en betegnelse hvor nær en prototyp befinner seg det endelige produktet. [Houde et. al., 1997] Som vi skal se senere i dette kapittelet er det ikke nødvendigvis noen sammenheng mellom disse to begrepene. Begrepsforvirringen blir ikke mindre av at hifi og lofi noen ganger også benyttes for å si hvorvidt en prototyp kjører på en datamaskin eller er en statisk representasjon.

I [Beaudouin-Lafon et. al., 2003] velger man å erstatte begrepet fidelity og heller snakke om høy og lav *presisjon*[1]. Dette fordi presisjon referer til innholdet av prototypen i seg selv heller enn dens forhold til et endelig ikke ferdigdefinert system. Denne oppgaven kommer til å følge denne definisjonen. I overenstemmelse med [Houde et. al., 1997] vil lofi og hifi bli brukt for å betegne hvor fjernt eller nær proto-

1. En nærmere definisjon på dette begrepet og hva som menes med det kommer senere i kapittelet.

typen ligger opp til det endelige systemet, mens for å betegne hvor polert prototypen er rent utseendemessig brukes begrepet *resolusjon*.

Utseende kan bedra

Det er altså ikke alltid slik at en designartefakt er i stand til å representere hvor fjernt eller nært prosjektet er ferdigstillelse. En konseptmodell laget tidlig i utviklingsprosessen kan utseendemessig se helt ferdig ut, mens en prototyp laget i sluttfasen for å teste systemets funksjonalitet kan ha blitt designet slik at visuelle detaljer er fjernet for å understreke den overordnede strukturen.

Houde og Hill understreker dette poenget slik: “prototyper er ikke selvforklarende: utseende kan bedra. Å klargjøre hvilke av aspektene til en prototyp som korresponderer til den endelige artefakten - og hvilke som ikke gjør det - er en viktig del av en vellykket prototypingsprosess.” [Houde et. al., 1997] (s 369 - oversatt)

Plassering av prototyper i designprosessen

Prototyping er en iterativ prosess, og alle prototyper gir informasjon om noen aspekter samtidig som de ignorerer andre. For at det skal være mulig å analysere en prototyp må den plasseres på rett sted i designprosessen, ettersom en prototyp defineres ut fra konteksten den er laget i. Designeren må vurdere formålet til protptypen på hvert trinn i designprosessen og velge den representasjonen som passer best til designspørsmålene. [Beaudouin-Lafon et. al., 2003]

Dette gjøres lettest ved å se på hvilke spørsmål prototypen er ment å besvare, altså se på hva den faktisk er ment *å prototype*. Det samme gjelder når man skal lage riktig prototyp for oppgaven. For at en prototyp skal kunne gi muligheten til å undersøke designspørsmål og evaluere løsninger, må man identifisere de viktigste åpne designspørsmålene. Vi sier at man undersøker hva som er en prototypes fokus. [Houde et. al., 1997]

Identifisering av prototypers fokus

Det er et faktum at interaktive systemer ofte er komplekse og at det derfor kan være vanskelig å prototype et helt design i de grunnleggende stadier av et prosjekt. [Houde et. al., 1997] Med et klart definert fokus for hver prototype, kan man bruke prototyper bedre for å tenke gjennom og kommunisere en designløsning.

I [Houde et. al., 1997] presenteres en modell for klassifisering av fokuset til en prototyp. Modellen har form av et triangel, der hvert ytterpunkt representerer en dimensjon basert på hvilke spørsmål en prototyp er ment å besvare. Triangelet er tegnet på skjeive for å understreke at ingen av spørsmålsgruppene er viktigere enn de andre.

Modellens hensikt

Hensikten med modellen er både at en designer skal kunne bruke den til å bestemme hvilken type prototyp som skal bygges, og være et visuelt verktøy for å forklare prototypens hensikt eksplisitt til de interessenter som ikke er med i selve designteamet. Det siste punktet er spesielt viktig, siden “prototyper ikke nødvendigvis er i stand til å forklare sin hensikt ut fra seg selv”. [Houde et. al., 1997] (s 380 - oversatt)

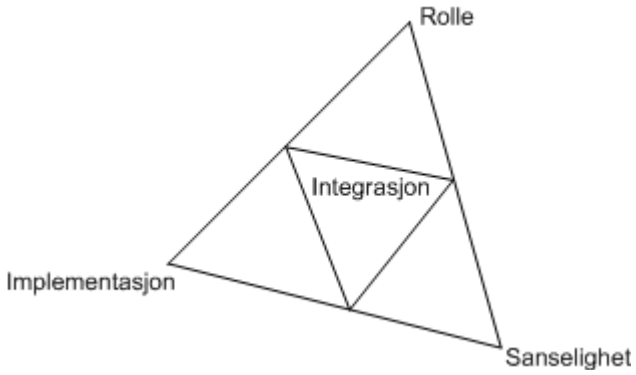


FIGURE 4-5: En modell av prototypers fokus

Rolle

Skal det endelige systemet gi brukere en ny eller utvidet form for funksjonalitet, vil de viktigste designspørsmålene omhandle hvilken *rolle* artefakten skal spille i brukerens liv. En prototyp vil da ha til hensikt å undersøke hvilke karakteristikker artefakten må ha for på best mulig vis støtte denne funksjonaliteten. Dimensjonen 'rolle' refererer altså til den gruppen spørsmål som går på hva et system vil være i stand til å gjøre for brukeren, altså selve nytteverdien til systemet. [Houde et. al., 1997]

Sanselighet

Er derimot artefaktens mål å presentere kjent funksjonalitet på et nytt vis, er det nødvendig at prototypen fokuserer på artefaktens brukbarhet, dvs utseende, opplevelse, osv. Dimensjonen 'sanselighet' refererer dermed til den gruppe spørsmål som går på hvordan en bruker opplever bruken av systemet. [Houde et. al., 1997]

Implementasjon

Skal artefaktens funksjonalitet være basert på ny teknikk, må en prototyp undersøke hvordan dette skal implementeres. Dimensjonen 'implementasjon' refererer slik til den gruppe spørsmål som går på hvilke teknikker og komponenter et system benytter for å utføre sine oppgaver. [Houde et. al., 1997]

Integrasjon

Et fjerde begrep knyttet til denne modellen er 'integrasjon'. En artefakts rolle, sanseslighet og implementasjon kan hver for seg være velkjent, uten at man vet hvordan disse faktorene spiller sammen. Prototyper som har til hensikt å besvare spørsmål som går på artefaktens helhet, vil befinne seg i tyngdepunktet mellom de tre dimensjonene. [Houde et. al., 1997] Dette kan være knyttet til det å balansere og finne løsninger på begrensninger i de forskjellige designretningene, å verifisere at systemet er komplett og sammenhengende, å vise interessenter en nær tilnærming til det endelige systemet, eller å få tilbakemeldinger fra sluttbrukere angående designen.

Prototypers plassering i modellen

Ved å plassere en prototyp i modellen kan man vise hvilket utviklingsstadium prototypen befinner seg i forhold til systemutviklingsprosessen. En prototyp bygget for å simulere hvordan et system kan tenkes å se ut og oppføre seg, og dermed har høy sanselighet men begrenset rolle og implementasjon, kan gi uerfarne inter-

essenter inntrykk av å inneholde mer funksjonalitet enn hva som faktisk er tilfelle. Ved å plassere den nær punktet 'sanselighet' i modellen, viser man at funksjonaliteteten enda ikke er implementert.

Analyse langs fire dimensjoner

Modellen over er i stand til å klassifisere forskjellige prototyper i forhold til deres oppgave, men sier lite om hva en prototype faktisk gjør eller hvordan den ser ut. En prototypes største fortrinn framfor en modell er jo nettopp at den er i stand til å visualisere problemstillingen på et mer konkret vis. I [Beaudouin-Lafon et. al., 2003] identifiseres fire dimensjoner, betegnet som hhv form[1], presisjon, interaktivitet og livsløp (evolusjon), som gjør det mulig å analysere prototyper nettopp med tanke på dette

Form

Formen til en prototyp betegner enten materialet prototypen er laget av eller graden av automatikk som er implementert, altså hvorvidt den er 'offline' eller 'online'. Ofte er det en sammenheng mellom disse to faktorene. Offline prototyper inkluderer alle former for undersøkende prototyper, som papirprototyper, skisser, konkrete konseptmodeller (laget i for eksempel papp, plast eller isopor), storyboards og videoer. Skjermbilder vil også falle inn under her. Online prototyper krever en datamaskin for å kunne brukes. Dette gjelder for eksempel dataanimasjoner, interaktive videorepresentasjoner, flash-prototyper og tidlige kjørbare versjoner av systemet. [Beaudouin-Lafon et. al., 2003]

Generelt kan man si at offline prototyper er billigere og raskere å produsere enn online prototyper. De tillater hurtige iterasjoner, og hjelper designere å utforske designrommet siden de i mindre grad hindrer designerens kreative tenking og det ikke er nødvendig å ta hensyn til de begrensninger som ligger i et ferdigdefinert komponentbibliotek. Den aller største fordelen med offline prototyper er kanskje likevel at de kan produseres av et vidt spekter interessenter, fra utviklere og designere, til ledelse, kunder og brukere. På denne måten tillater de at alle interessenter kan delta på tilnærmet likt grunnlag. [Beaudouin-Lafon et. al., 2003]

Online prototyper er mer krevende å utvikle, samtidig som de legger store begrensninger på hvilke designløsninger som er teknisk gjennomførbare. Slike prototyper egner seg derfor best i de senere iterasjonene i en designprosess, når man ønsker å besvare spørsmål om hvorvidt en design er *teknisk* brukbar, for eksempel teste om de automatiske prosessene støtter brukeren på en bra måte i hennes arbeide og at designen er teknisk gjennomførbar. [Preece et. al., 2002]

Presisjon

En prototyp er som sagt en begrenset designrepresentasjon av et konkret design som hjelper interessenter å visualisere og teste systemet som skal bygges. Selv

1. I [Beaudouin-Lafon et. al., 2003] brukes begrepet 'representasjon'. Siden dette begrepet er mye brukt gjennom denne oppgaven som et samlebegrep på modeller, prototyper og andre artefakter som representerer et brukergrensesnitt, vil denne oppgaven benytte begrepet 'form' i stedet.

om en prototyp ikke er i stand til å beskrive interaksjon formelt, kan den vise interaksjonen i brukergrensesnittet på en måte som er vanskelig å få til i tekstlige beskrivelser og abstrakte diagrammer [Beaudouin-Lafon et. al., 2003], for eksempel ved å demonstrere hvordan systemet lar brukeren kopiere inn tekst eller hvilke resultater det presenterer etter en databasespørring. Dette krever imidlertid en form for detaljering.

En prototyps presisjon beskriver relevansen prototypens detaljer har *i forhold til dens formål*. Dersom man grovskisserer et brukergrensesnitt, vil den relative størrelsen og plasseringen på komponentene ha stor relevans, mens ikoner, titler og spesifike komponenter ikke har det. [Beaudouin-Lafon et. al., 2003] Slike detaljer vil i dette tilfellet kunne ta oppmerksomheten vekk fra prototypens fokus, som er å besvare spørsmål rundt overordnet design.Detaljert er altså ikke nødvendigvis det samme som presist.

Sagt med andre ord definerer presisjon spenningen mellom hva prototypen stadfester og hva den holder åpen. Forholdet mellom relevante og irrelevante detaljer er viktig i denne sammenhengen. De detaljene den stadfester er klare for evaluering, mens de som er åpne må diskuteres nærmere og utforskes nøyere. Dette er en vesentlig karakteristikk ved prototyper. Det er allikevel nødvendig å innlemme upresise deler i prototypens design slik at prototypen kan evalueres og itereres videre. De upresise delene av et design bør derfor også være mindre detaljerte enn de presise for å understreke dette poenget. [Beaudouin-Lafon et. al., 2003]

Vanligvis er det sånn at jo flere iterasjoner som gjennomgås i designprosessen, jo høyere grad av presisjon har prototypen, ettersom flere og flere detaljer kommer på plass gjennom utforskning og testing. [Beaudouin-Lafon et. al., 2003]

Interaktivitet

En av de viktigste oppgavene til en interaksjonsprototyp er å illustrere samspillet mellom en bruker og et system. En prototyps interaktivitet beskriver i hvilken grad brukeren faktisk kan samhandle, påvirke og oppleve interaksjonen i prototypen.

Det å designe en effektiv og brukbar interaksjon i et brukergrensesnitt kan være svært vanskelig. Det finnes en lang tradisjon for hvordan man skal lage en design rent visuelt, men hvordan man designer brukbarheten til interaktive systemer er fortsatt et felt hvor designere har liten erfaring [Beaudouin-Lafon et. al., 2003]. Når man skal designe interaksjon må man ta hensyn til brukskonteksten: En dyp forståelse av sluttbrukerne, og innsikt i deres arbeidsoppgaver og måten de er vant til å utføre disse oppgavene på, er svært viktig for totalkvaliteten til et design. [Beaudouin-Lafon et. al., 2003] [Preece et. al., 2002]

Prototyper er i stand til å teste interaktivitet på forskjellige nivåer. Statiske (fixed) prototyper brukes ofte for å visualisere og teste scenarioer. De illustrere dermed hvordan systemet legger opp til interaksjon med brukeren gjennom sitt brukergrensesnitt. Fastsatt-sti (fixed-path) prototyper støtter svært begrenset interaktivitet.

De visualiserer hvordan interaksjonen vil fungere i gitte situasjoner for interessentene. De kan være effektive når man skal utforske et scenario og brukes også i horisontale og oppgaveorienterte prototyper. Åpne prototyper støtter interaksjon i stor grad. De fungerer som et virkelig system vil gjøre, med visse begrensninger: For eksempel dekker de ofte bare deler av systemet og har begrenset feilhåndtreingsmekanismer. De gjør det mulig å teste en lang rekke eksempler som viser brukerens interaksjon med systemet på en lang rekke områder. [Beaudouin-Lafon et. al., 2003]

Et viktig poeng er at interaktivitet, presisjon og form er gjensidig uavhengige dimensjoner. Man kan lage en offline, grovskissert papirprototyp som er svært interaktiv ved at en designer simulerer systemets planlagte respons på brukerens handlinger, og man kan lage svært presise dataanimasjoner som viser en predefinert brukssituasjon, det vil si hvordan systemet gir respons på definerte brukeraksjoner. [Beaudouin-Lafon et. al., 2003]

Livsløp

En fjerde dimensjon er prototypers forventede levealder, betegnet som ‘livsløp’ (evolusjon). Hovedsakelig kan man dele opp prototyper i tre kategorier: ‘Utforskende prototyper’ kasseres med det samme når deres misjon er utført; ‘iterative prototyper’ skal reflektere over en design og varer over to eller flere designiterasjoner; og ‘evolusjonære prototyper’ designes for å smelte sammen med, eller utvike seg til, det ferdige systemet. [Beaudouin-Lafon et. al., 2003]

Utforskende (bruk-og-kast) prototyper blir som regel benyttet i starten av en designprosess.De tar sikte på å utforske designrommet og dermed finne forskjellige muligheter rundt designen. Deres hensikt er å hjelpe designeren undersøke forskjellige varianter av interaksjon. Dette krever at de er svært raske og billige å utvikle. [Beaudouin-Lafon et. al., 2003] Ofte er utforskende prototyper offline papirprototyper, men de kan også være online prototyper implementert i et enkelt scriptspråk som Macromedia Flash eller Visual Basic.

Iterative (eksperimenterende) prototyper blir ofte brukt når man har vært gjennom den første konseptuelle fasen. De tar sikte på undersøke forskjellige muligheter rundt en design som har blitt etablert, og er mer rettet mot å undersøke brukbarheten til denne designløsningen. Hensikten med iterative prototyper er å la designeren og andre interessenter reflektere over et design, enten ved at presisjonen økes slik at de kan besvare spørsmål man har rundt detaljer, ved at man ser på flere varianter over det samme temaet eller prøver ut alternative løsninger på et kjent designproblem. [Beaudouin-Lafon et. al., 2003] Når iterative prototyper har utført sin hensikt, vil enten denne kunnskapen bli implementert i det som skal bli det endelige systemet og prototypen bli kassert, eller prototypen vil utvikle seg videre til en evolusjonær prototyp. Den siste varianten er alltid online, mens den første varianten kan i noen tilfeller være offline papir- eller eller konseptprototyper.

En evolusjonær prototyp er en iterativ prototyp som er designet med det for øye at den enten skal ende opp som en del av et system eller det ferdige systemet. Det å designe en prototyp som skal ende opp i et ferdig system, kontra det å prototype forskjellige designretninger som det er uklart om vil bli tatt i bruk eller forkastet, er imidlertid en vanskelig balansegang. Det å utvikle en prototyp som på samme tid er en *representasjon* av systemet og systemet i seg selv, gjør det vanskelig å utforske alternative design. Det krever derfor mer planlegging og større erfaring å prototype en evolusjonær prototyp enn en av de andre variantene. [Beaudouin-Lafon et. al., 2003]

4.4 Papirprototyper

En papirprototyp er en underkategori av offline prototyper som i hovedsak har fokus på å undersøke det planlagte brukergrensesnittets komposisjon og interaksjon. Prototypens fokus er å analysere enten systemets rolle eller sanselighet, eller en kombinasjon av disse. Både presisjon og interaktivitet kan variere fra svært lav til svært høy. [Beaudouin-Lafon et. al., 2003] [Houde et. al., 1997] [Preece et. al., 2002]

Papirprototyper vs. flash-prototyper

Både papirprototyper og flash-prototyper er mye brukt i forbindelse med interaksjonsdesign, og begge formene anbefales brukt i den iterative designprosessen. [Beaudouin-Lafon et. al., 2003] [Preece et. al., 2002] Det er imidlertid et poeng at man benytter rett type prototyp alt etter hva man ønsker å få svar på.

Som vi har sett kan forskjellen mellom offline og online prototyper være ganske liten med hensyn på fokus, interaktivitet, presisjon og livsløp. Forskjellen er tydeligst når man ser på prototypens form. Form er av betydning ettersom interaktiviteten i en prototyp oppleves forskjellig for brukeren når den simuleres av en designer, enn når den simuleres av en datamaskin. [Preece et. al., 2002] Dette gjelder selv om utfallet av brukerens handlinger blir det samme.

Den store fordelen med papirprototyper er deres versatilitet. De er billige, raske å designe, krever ikke tilgang til noen form for utstyr, de er intuitive i bruk og de er svært kommunikative. Det to siste punktene er ekstremt viktig når man arbeider i et designteam, hvor medlemmene i teamet ofte kommer fra forskjellige fagfelt og har forskjellig erfaring og kunnskap om datamaskiner og systemer. Med en papirprototyp stiller alle medlemmene i utgangspunktet likt. [Scanlon, Nettside:4, 1998] [Snyder, Nettside:5, 2001] I en flash-prototyp virker ofte designen mye mer definert og ferdig enn om den samme designen er representert på et stykke papir. Det er derfor lettere for medlemmene i gruppa å komme med alternative løsninger på et designproblem når man diskuterer over en upresis papirprototyp enn en kjørende prototyp. [Preece et. al., 2002]

Et annet viktig poeng er at en papirprototyp ikke har noen form for funksjonalitet implementert. Når man analyserer en flash-prototyp kan feil og mangler i funksjonaliteten virke forstyrrende, selv om det man faktisk skal diskutere er mulige designretninger og ønsket interaksjon. Også i forbindelse med brukbarhetstesting er det lettere å få brukeren til å fokusere på interaksjonen når man benytter en papirprototyp. I en flash-prototyp kan man lett overfokusere på overfladiske detaljer i stedet for prototypens innhold og funksjonalitet. [Beaudouin-Lafon et. al., 2003] [Preece et. al., 2002] [Snyder, Nettside:5, 2001]

Papirprototyper er derimot lite brukbare når man vil studere hvordan brukeren faktisk samhandler med et system. Slike faktorer som hvordan brukeren benytter inputenheter for å kommunisere med systemet, hvor brukeren til enhver har fokus, tidsforbruk på å løse oppgaver, frustrasjon over responstid, o.l. kan ikke simuleres ved hjelp av en offline papirprototyp. Dette må undersøkes i en brukbarhetslab på et kjørende system. [Preece et. al., 2002]

Fordeler og ulemper med papirprototyper

Alle interaksjonsprototyper benyttes til syvende og sist som et verktøy for å samle inn data om designen. Papirprototyper er spesielt nyttige når man ønsker å samle inn data vedrørende visuelle og strukturelle interaksjonselementer i et design. Dette kan være om konsepter og terminologier som benyttes i designen faktisk er på brukerens nivå; om navigasjon og arbeidsflyt matcher brukerens forventninger; om innholdet i brukergrensesnittet er tilstrekkelig og rett informasjon gis til rett tid; om layouten er oversiktelig designet; eller om funksjonaliteten støtter de oppgaver brukeren ønsker å utføre. [Snyder, Nettside:5, 2001]

Papirprototyper har imidlertid ikke muligheten til å simulere animerte elementer, lyder og hvordan komponentene i brukergrensesnittet faktisk oppfører seg (scrolling, rullegardiner, osv). Det at en person simulerer datamaskinens oppførsel gjør både at testsituasjonen kan virke kunstig på brukeren og at responstiden ikke er representativ. I tillegg mangler papirprototyper de tekniske begrensninger en kjørende kjørende prototyp har, og kan derfor ikke brukes til å teste teknisk gjennomførbarhet av konsepter og designideer. [Snyder, Nettside:5, 2001]

Fem varianter

Papirprototyper kommer i fem forskjellige varianter, fra de svært konkrete til de helt abstrakte.[1] [Constantine et. al., 2003] De mer abstrakte variantene fokuserer på å undersøke den overordnede organiseringen av brukergrensesnittet, og spørsmål knyttet til navigering og designarkitektur. De mer konkrete prototypene er designet for å hjelpe med å løse detaljerte designspørsmål rundt layout, visuell presentasjon og valg av brukergrensesnittobjekter, samt selve oppførselen og interaksjonen som skal implementeres i brukergrensesnittet. Ved å følge denne rangeringen kan

1. Når det gjelder papirprototyper er det som regel sammenheng mellom hvor konkret designen er og presisjonen i brukergrensesnittet. Også abstraksjons- og abstraheringsgrad er ofte sammenfallende.

man arbeide seg fra upresise og forenklede mot stadig mer realistiske og presise papirprototyper ettersom designen itereres og utvikler seg mot et ferdig system. [Constantine, 2003]

Papirprototyper suppleres ofte med et navigasjonskart som viser mulige stier gjennom interaksjonskontekstene[1] og transisjonene mellom dem. Interaksjonen kan også simuleres eller demonstreres av en designer.

Innholdsmodell

En innholdsmodell med tilhørende navigasjonskart regnes som den mest abstrakte prototypen. Innholdsmodellen er bygd opp rundt en serie interaksjonskontekster (views). Hver av disse er representert på et stykke papir hvor det blir festet post-it lapper som representerer hvilke oppgaver som skal utføres i hvert view (verktøy), og hvilke komponenter som trengs for å støtte denne oppgaven (materialer). I denne representasjonen er det ikke tatt hensyn til verken interaksjonskontekstenes størrelse, plassering, eller layout. [Constantine et. al., 2003]

Rammeskjema

Et rammeskjema viser den relative størrelsen og posisjonen til de forskjellige rammene (og vinduene) i brukergrensesnittet, men viser ingen spesifike brukergrensesnittobjekter. Fargelegging av de forskjellige rammene kan gjøres for å indikere rammenes innhold, eller hvor viktig deres informasjon eller funksjon er. [Constantine et. al., 2003]

Abstrahert layoutdiagram

Abstraherte layoutdiagrammer viser relativ størrelse og posisjon til brukergrensesnittelementene, men ikke deres eksakte plassering eller utseende. De viktigste brukergrensesnittobjektene er skissert opp, og hovedinnholdet i rammene er spesifisert. [Constantine et. al., 2003]

Upresis papirprototyp

En upresis papirprototyp er ofte en grov skisse av hvordan man tenker seg det ferdige brukergrensesnittet. De fleste brukergrensesnittobjektene er representert med omtrentlig plassering og størrelse og innholdet av de forskjellige rammene er bestemt, men utseendet er ikke detaljert i nevneverdig grad. [Constantine et. al., 2003] [Preece et. al., 2002]

Presis papirprototyp

Presise papirprototyper kjennetegnes ved en realistisk og detaljert design som ligner det ferdige produktet utseendemessig. Alle rammer er definert, både innholdsmessig og utseendemessig, og alle brukergrensesnittobjekter er tegnet opp med eksakt størrelse og plassering. [Constantine et. al., 2003] [Preece et. al., 2002] Også skjermbilder av kjørende systemer hører til innunder denne varianten av papirprototyper. Interaksjonen er da ofte beskrevet ved hjelp av tekstlige beskrivelser av hvordan brukeren kan utføre forskjellige oppgaver ved å utføre definerte aksjoner, og hva systemets repons på disse aksjonene er.

1. De rammer, vinduer eller dialoger et brukergrensesnittet er bygd opp av

4.5 Kanonisk abstrakte prototyper

Main Entry: **ca·non·i·cal**
Pronunciation: -ni-k&l
Function: *adjective*
1 : of, relating to, or forming a canon
2 : conforming to a general rule or acceptable procedure : ORTHODOX
3 : of or relating to a clergyman who is a canon
4 : reduced to the canonical form <a *canonical* matrix>
- **ca·non·i·cal·ly** /-k(&-)lE/ *adverb*

Main Entry: **canonical form**
Function: *noun*
: the simplest form of something; *specifically* **:** the form of a square matrix that has zero elements everywhere except along the principal diagonal

FIGURE 4-6: Fra Webster Online Dictionary: 'canonical' og 'canonical form'

Den kanonisk abstrakte prototypen er en spesialisering av det abstraherte layout-diagrammet. Den er en offline, delvis interaktiv og utforskende prototyp med middels til lav presisjon og dekningsgrad. Ved hjelp av denne prototypen kan designeren bygge opp innholdet og beskrive den overordnede organiseringen av av et brukergrensesnitt samt komponentenes grunnleggende funksjonalitet ut fra predefinerte 'kanoniske abstrakte komponenter'. En kanonisk abstrakt prototype kan derfor både betraktes som en konkret designartefakt og som en modell av arkitekturen til det brukergrensesnittet som skal designes. [Constantine et. al., 2003] [Constantine, 2003]

Denne prototypnotasjonen ble laget med tanke på å støtte en jevnere overgang fra den abstrakte oppgavemodellen til implementasjonsmodellen, og dermed gjøre det enklere for designeren å foreta spranget fra brukerkrav til konkret design [Constantine, 2003] I sandwich-modellen legger derfor den kanonisk abstrakte prototypen seg mellom det konkrete brukergrensesnittet og DiaMODL-diagrammet.

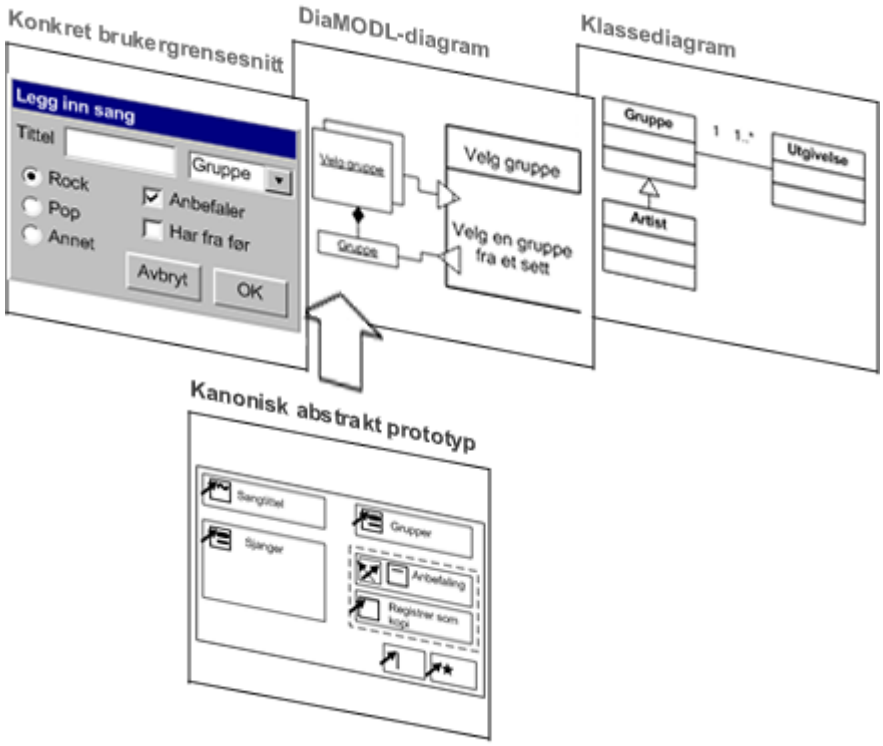


FIGURE 4-7: Kanonisk abstrakte prototypers plassering i sandwich-modellen

Ved å innlemme et standard sett av verktøy og materialer er ideen å gjøre det enklere for uøvde designere å bygge opp en design, ettersom valgmulighetene innskrenkes. For mer avanserte designere innebærer dette imidlertid en forenkling av selve *designprosessen*, slik at man kan få frigjort mer ressurser til å konsentrere seg om å utforske designrommet og utvikle kreative løsninger. En annen tanke er også at standardiserte abstrakte komponenter bør gjøre det lettere å kjenne igjen og beskrive de designmønstre (patterns) som gjennom erfaring har vist seg å fungere bra i interaksjonsdesignsammenheng. [Constantine et. al., 2003]

Kanoniske abstrakte komponenter

Notasjonen baserer seg på bruk av tre generiske abstrakte komponenter beskrevet i kanonisk form: ‘beholder’ (container), ‘aksjon/operasjon’ (action/operation) og ‘aktivt materiale’ (active material). Ut fra disse generiske komponentene er det laget et utvalg spesialiserte komponenter som beskriver mer nøyaktig *formålet* med komponenten. Disse komponentene deles opp i tre komponentgrupper: ‘materialer’ (materials), ‘verktøy’ (tools) og ‘aktive materialer’ (active materials).

Ettersom alle materialer, verktøy og aktive materialer effektivt er spesialiseringer av henholdsvis den generiske beholderen (representert som en kvadratisk boks), den generiske aksjonen/operasjonen (representert som en pil) og det generiske aktivet materialet (representert som en kvadratisk boks med en pil inni), vil det si at generiske komponenter kan brukes for ethvert formål. På denne måten sørger kan-

oniske abstrakte komponenter for et standard verktøysett av generiske komponenter som kan benyttes for å prototype den generelle layouten til et design, innholdet i de forskjellige rammene og den spesifike interaksjonen til hvert enkelt brukergrensesnittobjekt. [Campos et. al., 2004] [Constantine et. al., 2003]

Rent semantisk blir materialer og verktøy identifisert i prototypen ved hjelp av en tekst som beskriver det faktiske brukergrensesnittobjektet (materialer) eller aksjonen som skal utføres på et slikt objekt (verktøy). Hver komponent har også et ikon som visualiserer komponenten; hvilken funksjon den har eller hva den gjør.

En kanonisk abstrakt komponent kan i realiteten sees på som en interaktor, slik den er definert i DiaMODL. Også kanonisk abstrakte komponenter overbringer informasjon fra bruker til systemet og omvendt.

Materialer

Materialer brukes for å representere de komponenter i brukergrensesnittet som presenterer et spesifikt innhold eller informasjon, som for eksempel et bilde, et lydklipp, en tekst eller en elementsamling. De brukes også for å representere systemets status og eventuelle feilmeldinger eller beskjeder.

Når man skal navngi materialer er konvensjonen at man benytter navnet på innholdet som representeres (for eksempel “Navn”, “Filmklipp”). For samlinger benyttes enten flertall for å antyde at materialet innholder flere av noe for eksempel (“Adresser“) eller man beskriver hvilken type samling det er snakk om (for eksempel “Personlig adresseliste”). [Constantine et. al., 2003]

Table 1 – Summary of Canonical Abstract Materials

		MATERIALS
SYMBOL	INTERACTIVE FUNCTION	EXAMPLES
	container*	Configuration holder, Employee history
	element	Customer ID, Product thumbnail image
	collection	Personal addresses, Electrical Components
!	notification	Email delivery failure, Controller status

FIGURE 4-8: Oversikt over kanonisk abstrakte materialer [Constantine et. al., 2003]

Verktøy

Verktøy blir brukt for å representere at materialer kan endres eller manipuleres. Innenfor denne komponentgruppa finner vi både operasjoner og aksjoner. Operasjoner opererer direkte på materialer, mens aksjoner utfører en oppgave. [Constantine et. al., 2003] [Constantine, 2003]

Når man skal navngi verktøy spesifiserer man ganske enkelt den aktuelle oppgaven som skal utføres (for eksempel start, stopp, gå tilbake). Operatorer bør navngi materialene de opererer på (kopier fil, velg element, endre tekst). [Constantine et. al., 2003]

Table 2 – Summary of Canonical Abstract Tools

	TOOLS
SYMBOL INTERACTIVE FUNCTION	EXAMPLES
action/operation*	Print symbol table, Color selected shape
start/go/to	Begin consistency check, Confirm purchase
stop/end/complete	Finish inspection session, Interrupt test
select	Group member picker, Object selector
create	New customer, Blank slide
delete, erase	Break connection line, Clear form
modify	Change shipping address, Edit client details
move	Put into address list, Move up/down
duplicate	Copy address, Duplicate slide
perform (& return)	Object formatting, Set print layout
toggle	Bold on/off, Encrypted mode
view	Show file details, Switch to summary

FIGURE 4-9: Oversikt over kanonisk abstrakte verktøy [Constantine et. al., 2003]

Aktive materialer

Aktive materialer er en kombinasjon av verktøy og materialer, og brukes for å representere et objekt som har karakteristikker fra begge komponentklassene, for eksempel et editerbart tekstfelt eller en rullegardin. [Campos et. al., 2004] Slike komponenter kan bli sett på som enten en beholder som tar imot data fra brukeren, eller som et verktøy som opererer på et materiale. [Constantine et. al., 2003]

Når man skal navngi aktive materialer spesifiserer man materialet, og evt. operasjonen som hører til det aktive materialet.

Table 3 – Summary of Canonical Abstract Active Materials

	ACTIVE MATERIALS
SYMBOL INTERACTIVE FUNCTION	EXAMPLES
active material*	Expandable thumbnail, Resizable chart
input/accepter	Accept search terms, User name entry
editable element	Patient name, Next appointment date
editable collection	Patient details, Text object properties
selectable collection	Performance choices, Font selection
selectable action set	Go to page, Zoom scale selection
selectable view set	Choose patient document, Set display mode

FIGURE 4-10: Oversikt over kanonisk abstrakte aktive materialer [Constantine et. al., 2003]

Flere symboler som benyttes i notasjonen

I notasjonen er det også tatt med tre spesialsymboler: En trippel vinkel indikerer en repeterende komponent eller komponentgruppe. En oppstreket ramme indikerer at komponentene har en konseptuell tilhørighet til hverandre. Kommentarer i krøllparanteser brukes for å avklare komponenters oppførsel eller spesielle karakteristikker i designen. [Constantine, 2003]

Den ferdige representasjonen ligner i grove trekk det ferdige brukergrensesnittet, men ettersom alle detaljer er skjult er designen fortsatt åpen for kreative ikke-standardiserte designløsninger. [Constantine et. al., 2003]

Eksempel

Med utgangspunkt i modellene fra forrige kapittel, kan en kanonisk abstrakt prototyp av en lånekalkulator se slik ut:

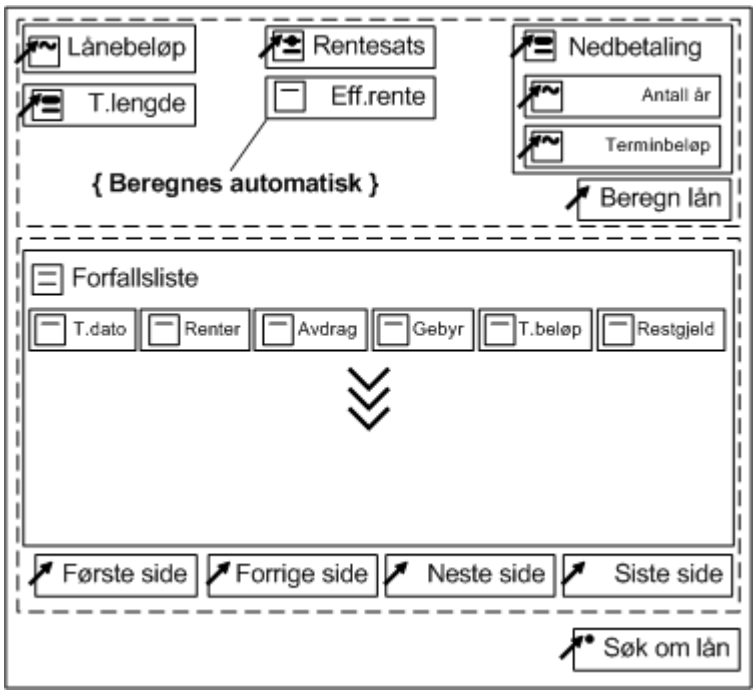


FIGURE 4-11: Kanonisk abstrakt prototyp av enkel lånekalkulator

Realisering av kanonisk abstrakte prototyper

Når en kanonisk abstrakt prototyp skal realiseres i en design med større presisjon, involverer dette både visuell design og interaksjonsdesign. Designeren må bruke erfaring eller prøve seg fram for å finne en konkret brukergrensesnittkomponent som passer med den kanonisk abstrakte komponenten i den kanonisk abstrakte prototypen

Prosessen med å designe et brukergrensesnitt ut fra en kanonisk abstrakt prototyp har grovt sett disse fasene: Først blir de enkelte abstrakte komponentene, de *nøstede* komponentene og komponenter med konseptuell tilhørighet identifisert. Så, for hver slik gruppe eller kombinasjon bør man:Tenke ut både konvensjonelle og kreative løsninger som kan realisere den spesifiserte interaksjobnen; velge ut noen lovende kombinasjoner; og foredle designen [Constantine et. al., 2003]

Selv om notasjonen til den kanonisk abstrakte prototypen mangler en presis formalisme og semantikk som kreves for å gi verktøystøtte og automatisk generering av brukergrensesnitt, er notasjonen såpass uttrykksfull at man på denne måten kan generere konkrete brukergrensesnitt fra kanonisk abstrakte prototyper. [Campos et. al., 2004]

Bruksterksel kontra brukshøyde

Den kanonisk abstrakte prototypen har relativt lav bruksterskel, men også relativt lav brukshøyde. Bruken av den er begrenset til å lage abstrakte prototyper av komponentbaserte systemer. Systemer med høyt innslag av kompleks interaktivitet (for eksempel spill) vil kanskje bli vanskelig å prototype ved hjelp av denne notasjonen.

Fordeler og ulemper med kanonisk abstrakte prototyper

Interaksjon beskrevet ved hjelp av kanonisk abstrakte komponenter er ganske enkel å sette seg inn i, og sammenhengen mellom dem og deres respektive brukergrensesnittkomponenter er relativt tydelig. Kanonisk abstrakte prototyper er også enkle å skissere med på papir, og kan derfor benyttes for eksempel i forbindelse med brainstorming.

På den annen side beskriver ikke en kanonisk abstrakt komponent dataflyten mellom interaktorene entydig, noe som gjør at den kanonisk abstrakte prototypen må regnes som mindre formell enn DiaMODL. Ettersom dataflyten ikke er beskrevet og strukturen med blant annet funksjoner som vi finner i DiaMODL mangler, vil en kanonisk abstrakt prototype ofte gi rom for mange tolkninger når man forsøker å forstå interaksjonen i systemet den representerer.

4.6 Oppsummering

Interaksjonsprototyper benyttes i hovedsak for å støtte den kreative designprosessen, det vil si arbeidet med å utforme og undersøke gode designløsninger.

Prototyper kan enten kastes etter bruk, eller utvikle seg til mer avanserte prototyper eller til et ferdig system. Uansett livsløp, og uansett form, er dokumentasjon av interaksjonen til prototypen et problem. Prototypen er ofte i seg selv den eneste dokumentasjonen som finnes på hvilken respons et system skal gi på en gitt brukerhandling. Hvilke mekanismer som styrer denne interaksjonen må man ofte bare gjette seg til.

En av de få prototypene som har forsøkt å løse dette problemet er den kanonisk abstrakte prototypen. De kanonisk abstrakte komponentene er først og fremst en beskrivelse av hva et brukergrensesnitt inneholder av generiske komponenter. Det som gjør de kanonisk abstrakte komponentene spesielle er imidlertid at de også er i stand til å dokuementere ikke bare hva slags respons et system gir på en gitt brukerhandling, men også *hvordan* systemet utfører denne resonsen.

Når vi skal lage en hybrid representasjon av en interaksjonsmodell og en interaksjonsprototyp, vil derfor de kanonisk abstrakte komponentene både kunne brukes for å representere selve innholdet i brukergrensesnittet og hvordan systemet responderer på brukerens handlinger.

Kapittel 5

ProtoMODL - En hybrid mellom en abstrakt prototyp og dialogmodell

Dette kapittelet diskuterer likheten og forskjellen mellom DiaMODL og den kanoniske prototypen og hvordan man kan kombinere de beste egenskapene til disse representasjonene for å lage en ny hybrid notasjon. Denne hybride notasjonen, ProtoMODL, er i stand til å kombinere interaksjonsmodellens evne til å kapsle inn kompleks og abstrakt informasjon med interaksjonsprototypens evne til å formidle denne informasjonen til en interessent.

5.1 Interaksjonsrepresentasjoners informasjonsinnhold

Vi har sett på to tilnærminger som adresserer problemet med hvordan man kan designe interaksjon på en strukturert og kontrollert måte slik at man unngår ad-hoc løsninger. Både interaksjonsmodellering og -prototyping er egnede metoder til dette formålet når de brukes i en gjennomført og veldokumentert utviklingsprosess, som for eksempel ISO 13407. Begge disse tilnærmingene kjennetegnes ved at en interaksjonsrepresentasjon, enten i form av en modell eller en prototyp, står sentralt i designprosessen.

Interaksjonsmodell kontra interaksjonsprototyrp.

Den store forskjellen mellom modeller og prototyper som representasjonsartefakter for interaktive systemer, ligger i hva som er den grunnleggende *hensikten* med den. Ettersom en prototyp er en konkret representasjon gir den mer rom for direkte interaksjon med en bruker og støtte for eksperimentell designutvikling, mens en modell er abstrakt i formen og definerer gjennom nettopp abstraksjon de underliggende mekanismer og beskriver dermed funksjonaliteten til det interaktive systemet på et bedre sett enn hva prototyper klarer å formidle.

Interaksjonsmodeller har lav deknings- og høy abstraheringsgrad: Informasjonen er fokusert mot enkelte aspekter og mye urelevant informasjon er abstrahert vekk. Dette, sammen med høy abstraksjonsgraden, gir modellen mulighet til å uttrykke skjult informasjon, som for eksempel hvordan dataflyten mellom systemkomponenter foregår. Imidlertid er det vanskelig for en modell å uttrykke resultatet av en handling på en intuitiv måte, for eksempel kan den ikke vise den konkrete responsen et system vil gi på en input fra brukeren. For at en mottaker skal kunne forstå og

benytte seg av informasjonen en modell representerer, må også den notasjonen en modell benytter for å beskrive de aspekter den er fokusert mot være godt kjent.

Prototyper er ofte er laget for å vise fysiske egenskaper eller konkrete virkemåter til et tenkt systemet. [Preece et. al., 2002] En prototyp som er laget med tanke på å undersøke en fysisk egenskap, slik som form, størrelse, vekt eller utseende, fokuserer som regel kun på dette aspektet ved å abstrahere vekk annen informasjon. En prototyp laget for å undersøke en konkret virkemåte abstraherer vekk urelevante eller ikke-definerte aspekter ved systemets funksjonalitet. Slik kan prototyper ha høy abstraheringsgrad, men lav dekningsgrad, akkurat som en modell.

Den store forskjellen på en modell og en prototyp er altså at en prototyp er mer konkret, og dermed mer intuitive og lettere å forstå for mennesker. Det er sjelden nødvendig å sette seg inn i en spesiell notasjon for å forstå prototypen. På den annen side har ofte prototypen en lav formalitetsgrad. Man kan spesifisere hvilken informasjon som skal mates inn i et system, og hvilken informasjon som systemet spytter ut, men det er vanskelig å beskrive hvordan systemet kommer fram til dette resultatet. Det er derfor ofte nødvendig å ha mye kunnskap om den konteksten prototypen opererer i. For eksempel vil en trekloss som skal vise fasong og størrelse til en PDA bare være nyttig så lenge man vet hva den forestiller og hvilken funksjon PDA-en skal ha. [Preece et. al., 2002]

5.2 Analyse av tre interaksjonsrepresentasjoner

Hensikten med å benytte designrepresentasjoner er å gjøre det enklere å identifisere problemområder, finne løsninger og utforske det mulige designrommet, dokumentere designet og designprosessen, og være et verktøy som stimulerer og sikrer kommunikasjon mellom medlemmene i designteamet. [Beaudouin-Lafon et. al., 2003] [Nunes et. al., 2000] [Trætteberg, 2002:1] Noen representasjoner er imidlertid bedre egnet til enkelte oppgaver enn andre. Det er derfor nødvendig at man raskt kan lage en ny representasjon, som egner seg bedre til å uttrykke det aspektet man ønsker å fokusere på, basert på en annen representasjon. I denne sammenhengen er det spesielt interessant å undersøke i hvilken grad de tre interaksjonsrepresentasjonene skjermbilde, kanonisk abstrakt prototyp og DiaMODL-diagram er i stand til å løse disse oppgave.

Utforskning, spesifikasjon, dokumentasjon, kommunikasjon

Gode representasjoner av et brukergrensesnitt og dets interaksjonsegenskaper gjør det lettere å utforske et problemområde. De kan la interessenter fokusere på de enkelte aspekter innenfor problemdomenet som er uavklarte, mens aspekter som enten er uvesentlige eller allerede er veldefinerte kan abstraheres vekk. Når så designen skal implementeres er det nødvendig å ha en representasjon som kan spesifisere hva som skal lages, hvordan det skal lages og hvilken funksjonalitet det skal ha.

Dokumentasjon av brukergrensesnittet og hvordan interaksjonen foregår er nødvendig i mange tilfeller. Ved ferdigstillelse må man kunne dokumentere hva som er gjort og hvordan systemet er bygd opp, for eksempel med tanke på vedlikehold, videre utvidelser av systemet eller som en del av en brukermanual. Også underveis i designprosessen kan det være viktig å dokumentere interaksjon; dersom det for eksempel skjer utskiftninger eller utvidelser av designteamet, må nye medlemmer raskt og presist få innsikt i prosessen og kunnskap om hva som allerede er bygd, og hva som gjenstår og hvordan dette skal implementeres i forhold til spesifikasjonen.

Å gjøre det enklere å kommunisere ideer og konkretisere forslag til løsninger er også en viktig oppgave til en representasjon. For at en representasjon skal kunne benyttes til dette formålet bør alle interessenter ha inngående kjennskap til den notasjon som benyttes, slik at de både er i stand til å forstå de representasjonene som andre legger fram og uttrykke sine egne ideer for andre.

Skjermbilder

Skjermbilder beskriver svært konkret et brukergrensesnitts utseende og komposisjon. Dersom det benyttes standardkomponenter som alle interessenter er kjent med fra før, gir de også et visst innblikk i hvordan interaksjonen skjer. Et problem med å benytte skjermbilder som et verktøy i en designprosess er at interessenter kan få inntrykk av at systemet er nærmere ferdigstillelse enn hva som faktisk er tilfelle. I tillegg kan valg av spesifike komponenter tidlig i designprosessen stenge store deler av designrommet for videre utforskning, slik at man risikerer at gode løsninger aldri blir oppdaget. [Beaudouin-Lafon et. al., 2003]

Skjermbildet er en veldig konkret representasjon. Brukergrensesnittets enkeltkomponeter og layout er klart spesifisert, men interaksjonen må forstås helt intuitivt. Interaksjonen er altså beskrevet uformelt.

Abstraksjons- og abstraheringsgraden er lav, mens dekningsgraden er relativt høy. Et skjermbilde har i seg selv relativt lav formalitet, selv om formaliteten til et brukergrensesnitt laget med en editor som benytter et standardisert vverktøysett kan være ganske høy.

Interaksjonen i et ferdig brukergrensesnitt dokumenteres ofte i form av skjermbilder og en tekstlig forklaring. Brukergrensesnitt blir imidlertid ofte innfløkte, noe som krever at man må bruke mye tekst for å forklare hva som egentlig skjer. Ofte kan også et skjermbilde gi opphav til flere tolkninger. Det er vanskelig å dekke alle eventualiteter og skrive presise og entydige beskrivelser. En slik representasjon kan altså bli tvetydig, og passer derfor dårlig inn i en spesifikasjon som danner grunnlaget for hvordan brukergrensesnittet skal konstrueres.

Når man ønsker å formidle sine tanker om et design til andre, er det enkelt å vise ved hjelp av skjermbilder *hva* man ønsker et konkret design skal inneholde. Å vise *hvordan* det støtter brukeren i de oppgavene hun ønsker å utføre er derimot ikke

like enkelt. Man blir da igjen nødt til enten å lage detaljerte scenarier, skrive lange tekstlige forklaringer eller forklare muntlig hvordan man ser for seg at interaksjonen vil foregå. Også i kommunikasjonsøyemed er altså risikoen stor for at misforståelser oppstår og at interessentene har ulike tolkninger av interaksjonen.

Kanonisk abstrakte prototyper

En kanonisk abstrakt prototyp representerer i teorien brukergrensesnittet på en mer kompakt og utvetydig måte enn hva et skjermbilde er i stand til. Den er først og fremst ment å være et hjelpemiddel som skal hindre at designrommet innskrenkes ved at spesifike brukergrensesnittkomponenter låser designet i en tidlig fase, og hjelpe kommunikasjonen mellom designere i et team til å uttrykke sine konkrete designideer. [Constantine et. al., 2003]

Den kanonisk abstrakte prototypen virker ikke spesielt velegnet til å spesifisere og dokumentere interaksjonen i et brukergrensesnitt. En kanonisk abstrakt komponent beskriver bare hva den tilbyr brukeren av data og funksjonalitet, ikke hvordan den utveksler data med andre komponenter. En kanonisk abstrakt prototyp sier altså svært lite om hvordan interaksjonen innad i brukergrensesnittet foregår. Den vil derfor, som et skjermbilde, kreve en god del tilleggsinformasjon for at en interessent skal få god innsikt i et planlagt brukergrensesnitts interaksjon.

Denne representasjonen fokuserer først og fremst på brukergrensesnittets *layout* og funksjonaliteten til komponentene som inngår. Enkeltkomponentene er abstrahert ned slik at de utelukkende beskriver hva den enkelte komponent presenterer for brukeren og hvordan de reagerer på en brukerhandling. Interaksjonen er beskrevet relativt abstrakt, ved hjelp av symboler. Denne representasjonen har dermed en høyere abstraksjons- og abstraheringsgrad enn et skjermbilde, men lavere dekningsgrad. Selve interaksjonen beskrives på et relativt formelt vis.

DiaMODL-diagrammer

DiaMODL er en formell notasjon som brukes for å beskrive hvilke data som presenteres for brukeren og hvilke som sendes tilbake til systemet for videre prosessering, og hvordan denne utvekslingen foregår. [Trætteberg, 2002:1] Dette gjør DiaMODL til et kraftig verktøy for å spesifisere og dokumentere systeminteraksjon.

Et DiaMODL-diagram er i stand til å vise antall komponenter som eksisterer i et brukergrensesnitt og deres konkrete innhold, uten å beskrive brukergrensesnittets layout, utseende eller brukerens spesifike handlinger mot systemet nærmere. Kompliserte brukergrensesnitt beskrevet i denne notasjonen kan imidlertid bli uoversiktelige, og datastrømmene derfor bli vanskelige å følge. I tillegg er notasjonen lite komprimert, noe som bidrar til at DiaMODL-diagrammer ikke lett lar seg integrere i en spesifikasjon. Notasjonen er også relativt krevende å sette seg inn i. Alt dette kan gjøre det vanskelig å benytte slike representasjoner til å kommunisere ideer og løsninger i en initiell designfase som ofte involverer personer fra mange fagfelt og interessesfærer.

DiaMODL-diagrammet beskriver kun brukergrensesnittets interaktorer og interaksjonen mellom dem. Notasjonen er svært abstrakt og formelt beskrevet; det er ideen om systemets interaksjon som presenteres og ikke dets konkrete respons på brukerens handlinger. Når det gjelder dekningsgraden er den lav, ettersom det fokuseres på den rene interaksjonen i brukergrensesnittet. Abstraheringsgraden er høy, ettersom all uviktig informasjon om designets utseende abstraheres vekk.

Som vi ser er det ingen av disse representasjonene som er i stand til å både utforske, spesifisere, dokumentere og kommunisere interaksjonen i et brukergrensesnitt. Det er derfor rimelig å anta at en representasjon som er i stand til dette potensielt kan være et viktig hjelpemiddel i en designprosess.

Overføring av informasjon mellom representasjonene

Ettersom enhver representasjon av et system fokuserer på noen aspekter og utelater andre, er det nødvendig å kunne lage nye representasjoner som dekker de aspektene man fokuserer på, samtidig som man tar vare på den relevante informasjonen som finnes i de representasjonene man allerede har produsert. Dette vil for eksempel være et tema når man skifter fokus fra design av brukergrensesnittets interaksjon til design av brukergrensesnitets komposisjon.

Settes de tre representasjonene vi har analysert i et diagram med hensyn på deres deknings- og formalitetsgrad, ser vi at dekningsgraden til et DiaMODL-diagram er lav i forhold til skjermbildet og den kanonisk abstrakte prototypen, mens formalitetsgraden er høy. Abstraksjons- og abstraheringsgraden til de tre representasjonene følger dekningsgraden relativt tett. De to prototypene ligger i dette diagrammet ganske tett sammen, mens det er et gap ned til DiaMODL-diagrammet.

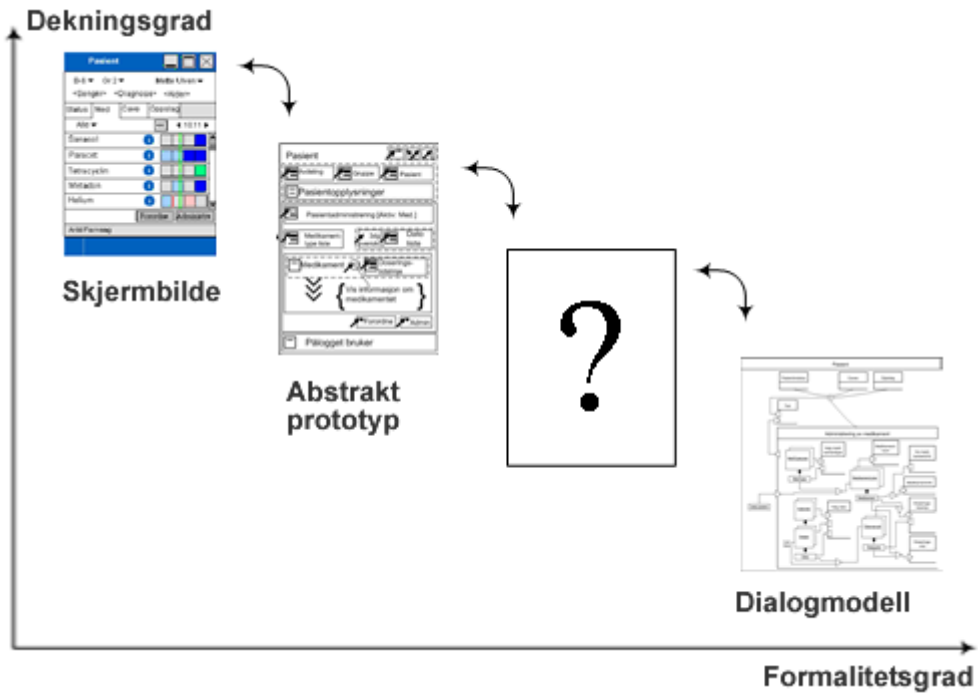


FIGURE 5-1: Overføring av informasjon mellom designrepresentasjoner

Dette tilsier at å overføre informasjonen i et skjermbilde til en kanonisk abstrakt prototyp og omvendt er relativt enkelt; de kanonisk abstrakte komponentene korresponderer i stor grad til de brukergrensesnittkomponenten i skjermbildet.

Å overføre relevant informasjon fra en kanonisk abstrakt prototyp til et DiaMODL-diagram, eller omvendt, virker imidlertid en god del vanskeligere. En kanonisk abstrakt komponent har mange likhetstrekk med en interaktor, ettersom begge to representerer en konkret brukergrensesnittkomponent (eller en gruppe av slike). Det er imidlertid vanskelig å oversette informasjonen som finnes i de kanonisk abstrakte komponentene til DiaMODLs kontrollstruktur med objekter, objektsett og funksjoner. I tillegg er ikke koblingene mellom de kanonisk abstrakte komponentene klart spesifisert. Skal man gå andre veien er abstraksjonsgraden til DiaMODL et problem. Det krever stor innsikt i notasjonen og en opparbeidet kunnskapsbase om hvilke mønstre (patterns) i notasjonen som tilsvarer hvilke kanonisk abstrakte komponenter. Det kreves derfor mye arbeid, inngående kjennskap til begge de to representasjonenes notasjon og et velkjent problemdomen dersom man ønsker å overføre informasjon mellom disse representasjonene.

Det kan av den grunn hevdes at det trengs en representasjon som ved å vise både interaktorenes konkrete innhold og funksjonalitet, og koblingene mellom dem i samme diagram, kan gjøre denne transgresjonen enklere. En slik representasjon må nødvendigvis være mer formell og ha mindre dekningsgrad enn en kanonisk abstrakt prototyp, og være mindre formell og ha høyere dekningsgrad enn et DiaMODL-diagram.

En utstrakt hånd

Som nevnt tidligere er DiaMODL en notasjon som har som målsetning å ivareta formalismen som ligger bak det konkrete brukergrensesnittet, samtidig som det skal være et praktisk designverktøy. [Trætteberg, 2002:1] Den kanonisk abstrakte prototypen er en representasjon som skal vise det konkrete brukergrensesnittet i abstrakt form, og kan derfor anses å være være en modell av arkitekturen til brukergrensesnittet som skal designes. [Constantine et. al., 2003] Begge disse representasjonene gjør dermed et forsøk på å komme den andre tilnærmingen i møte: Mens den kanonisk abstrakte prototypens målsetning er å være et bindeledd mellom det konkrete brukergrensesnittet og den underliggende implementasjonsmodellen, er hovedmålet til DiaMODL å være en analyserbar spesifikasjon på all relevant interaksjon, sett fra brukerens ståsted, og nærmer seg dermed konkret design fra den formelle modelleringssiden. Dette bør gjøre det lettere å finne sammenfallende områder som kan utnyttes til å lage en representasjon som har egenskaper fra både DiaMODL og den kanonisk abstrakte prototypen.

5.3 En hybrid representasjon

Utfordringen er altså å lage en representasjon som både kan brukes til å utforske, spesifisere, dokumentere og kommunisere interaksjonen i et brukergrensesnitt. Det er ønskelig at interaksjonen beskrives ut fra interaktorer som er i stand til å beskrive datautvekslingen mellom bruker og system og hvordan denne utvekslingen i prinsippet skal foregå. I tillegg bør representasjonen benytte koblinger til å spesifisere hvordan data utveksles mellom interaktorene. Slik er det mulig å bevare den eksplisitte informasjonen i både den kanonisk abstrakte prototypen og DiaMODL-modellen.

Ønskede bruksområder

Ved å kombinere DiaMODL sin evne til å kapsle inn kompleks og abstrakt informasjon i et enkelt diagram med den kanonisk abstrakte prototypens evne til å formidle denne informasjonen til en interessent, bør en hybrid notasjon dekke alle de oppgavene vi ønsker å få utført. Tabell 5-1 viser hvilke representasjoner som egner seg til hvilke bruksområder.

Tabell: 5-1: DiaMODLs og den kanonisk abstrakte prototypens bruksområder

	Design	Spesifikasjon	Dokumentasjon	Kommunikasjon
Kan. abstrakt prototyp	X			X
DiaMODL		X	X	
Hybrid repr.	X?	X?	X?	X?

Fellesnevneren

Fellesnevneren til DiaMODL og kanonisk abstrakte prototyper er interaktorer. En interaktor er definert som en komponent som er i stand til å overbringe informasjon fra systemet til brukeren og omvendt.

DiaMODL sin notasjon er direkte basert på interaktorbegrepet, mens i kanonisk abstrakte prototyper kan vi finne de igjen i form av de kanonisk abstrakte komponentene, dvs materialer, verktøy og aktive materialer.

Hvis vi minsker dekningsgraden til den kanonisk abstrakte prototypen, det vil si kun fokuserer på komponentene og fjerner fokuset på layout, har prototypen og DiaMODL omtrent like stor abstraheringsgrad. Begge notasjonene fjerner de konkrete brukergrensesnittkomponentene og erstatter de med generiske interaktorer. Men hvor DiaMODL benytter koblinger mellom porter, funksjoner og objektsett for å beskrive selve interaksjonen, bruker den kanonisk abstrakte prototypen en rekke ferdigdefinerte symboler som representerer forskjellige interaksjonsformer. Vi ser at det altså er formen som er forskjellig i de to representasjonene, ikke innholdet.

De to representasjonene har også overlappende dekningsgrad. Fokuset for begge representasjonene sin del er interaksjonen i brukergrensesnittet. Forskjellen her ligger i at et DiaMODL-diagram sitt fokus er sentrert mot systemsiden av interaksjonen, mens fokuset til en kanonisk abstrakt prototyp er sentrert på interaksjonen med brukeren.

Ettersom begge disse notasjonene baserer seg på interaktorkomponenter bør det være mulig å lage en representasjon som både evner å fange opp dataflyten mellom komponentene i et brukergrensesnitt slik som vi finner den i et DiaMODL-diagram, og det visuelle uttrykket og interaksjonen mellom brukeren og systemet slik vi ser den i en kanonisk abstrakt prototyp.

Den hybride representasjonens plassering i Sandwichmodellen

Innledningsvis ble sandwich-modellen brukt for å vise hvordan denne oppgaven er rettet mot hvordan man kan benytte modellering i praktisk design. Diagrammet som viser informasjonsoverføringen mellom de tre designrepresentasjonene som er analysert tilsvarer en slik sandwich-modell. En hybrid representasjon basert på notasjonen til DiaMODL og den kanonisk abstrakte prototypen vil da ta plass i modellen slik:

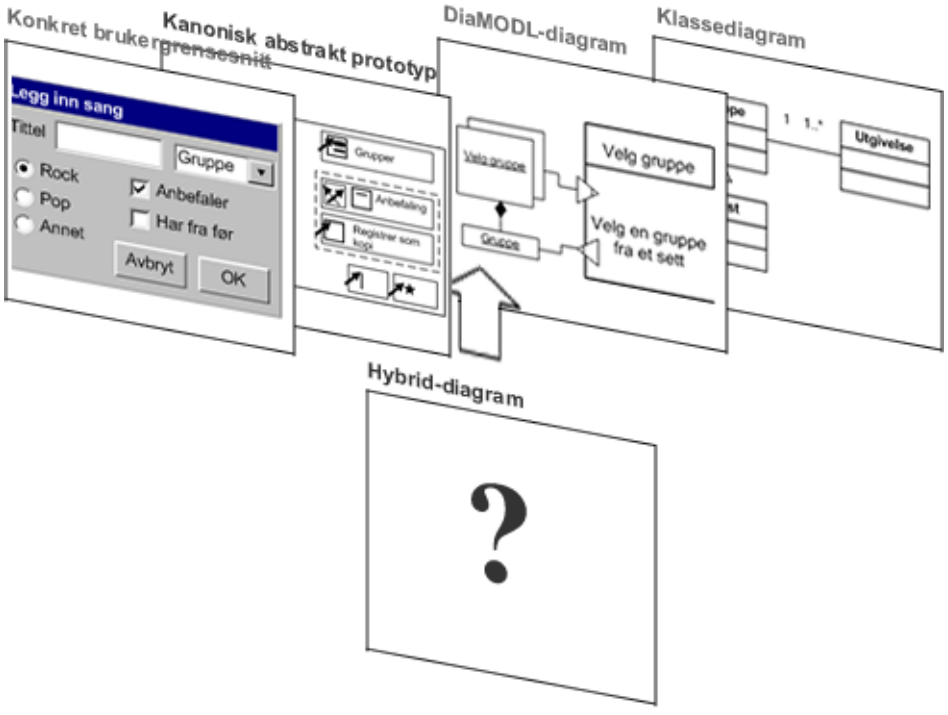


FIGURE 5-2: Hybrid representasjon satt inn i sandwichmodellen

Hypotese

I innledningen ble det lansert en hypotese. Den er et resultat av de forutsetningene som ble omtalt i innledningskapittelet, teorien som er presentert i de tre foregående kapitlene, og de tankene som er gjort hittil i dette kapittelet. Før

notasjonen til den hybride representasjonen presenteres, er det greit å minne om den:

Ved å kombinere notasjonen til den kanonisk abstrakte prototypen med notasjonen til DiaMODL, kan man lage en designrepresentasjon som viser både interaksjon og generiske komponenter i samme diagram. En slik representasjon vil kunne spesifisere den underliggende interaksjonen i et brukergrensesnitt på en slik måte at man fjerner eller reduserer tvetydigheter, og dermed gjøre det lettere å dokumentere denne interaksjonen, utforske alternative designløsninger, samt kommunisere ideer rundt dette til andre interessenter.

5.1 ProtoMODL: En hybrid mellom prototyp og dialogmodell

Notasjonen som har skal utvikles har fått betegnelsen 'ProtoMODL' for å understreke at det er en hybrid av en (kanonisk abstrakt) prototyp og DiaMODL

Krav til notasjonen

I kapittel 3 og 4 identifiserte vi en del egenskaper som var viktig å implementere i en hybrid representasjon. Fra notasjonen til den kanonisk abstrakte prototypen skal vi bruke de kanonisk abstrakte komponentene, ettersom de beskriver systemets oppførsel i forhold til brukerens aksjoner ved hjelp av symboler som er relativt intuitive og enkle å forstå for en interessent. Fra DiaMODL sin notasjon anses interaktorbegrepet med tilhørende koblinger og funksjoner som viktig å ha med videre, ettersom man ved hjelp av disse elementene kan beskrive både retningen og innholdet til den interaksjon som skjer innad i brukergrensesnittet. Slik blir ProtoMODL utviklet med tanke på å bevare den kanonisk abstrakte prototypens relativt konkrete uttrykksform samtidig som dataflyten slik den er identifisert i et DiaMODL-diagrammet blir bevart.

Det er vesentlig at man kan tegne opp diagrammer direkte på papir, for eksempel i forbindelse med idegenerering og utforskning av et designrom. Notasjonen må derfor være relativt komprimert, slik at komplekse brukergrensesnitt kan tegnes på en relativt liten flate. Selve elementene i notasjonen må nødvendigvis baseres i stor grad på de to notasjonene den er inspirert av.

Notasjonen bør videre være såpass formell at det er mulig å utvikle dataverktøy som støtter notasjonen. Det er imidlertid ikke denne oppgavens oppgave å bevise at dette er mulig.

Notasjon

Notasjonen er, som DiaMODL, basert på interaktorer med tilhørende koblinger og funksjoner for å beskrive dataflyten. Det finnes imidlertid flere viktige forskjeller.

Interaktor

Først og fremst er selve interaktoren forenklet. I ProtoMODL defineres en interaktor som en boks hvor høyre vegg er fjernet. Slik videreføres tanken fra DiaMODL om at interaktorens høyre side skal være åpen for interaksjon mot brukeren.

For å beskrive interaktorens oppførsel og innhold benyttes et 'interaktorsymbol' sammen med en 'interaktorbeskrivelse'. Dette erstatter strukturen bygd opp rundt objekter og objektsett slik vi finner den i DiaMODL.

Interaktorsymbol

Interaktorsymbolene som benyttes er de kanoniske abstrakte komponentene, slik de er definert i notasjonen til den kanonisk abstrakte prototypen. Interaktorsymboler har i oppgave å beskrive hvilken type interaksjon det er snakk om. Dette kan for eksempel være å velge et enkelt objekt fra et sett, endre et objekt, sette i gang eller avslutte en handling, osv.

Interaktorbeskrivelse

Interaktorbeskrivelsen forteller hvilke data som presenteres for brukeren, og hvilke data som interaktoren sender tilbake til systemet som respons på en gitt brukerhandling. Konvensjonen er slik at en beskrivelse er delt opp i to deler; en del som angir brukerens handling på interaktoren, og en del som angir hvilken type data interaktoren skal håndtere. Det skal også gå fram av beskrivelsen om det er snakk om enkeltobjekter eller sett av objekter.

Noen slike interaktorbeskrivelser kan for eksempel være "Velg artist" eller "Skriv inn navn", der den første beskrivelsen indikerer at man skal velge et enkeltobjekt fra et sett, mens den andre indikerer endring av et objekt. Interaktorbeskrivelsen støtter på dette viset oppunder tolkningen av interaktorsymbolet, og presiserer samtidig brukerens interaksjon med det faktiske brukergrensesnittobjektet. I de tilfeller der en interaktor bare har i oppgave å presentere data for brukeren, vil man i beskrivelsen kun angi hvilke data det er snakk om. En slik interaktor kan for eksempel gis beskrivelsen "Gjeldende sangtittel".

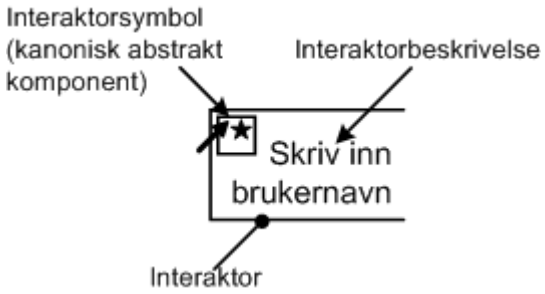


FIGURE 5-3: Interaktorbegrepet i ProtoMODL

Koblinger

I ProtoMODL er konseptet med porter fjernet, slik at koblingene er festet direkte i interaktoren. En kobling kan som i DiaMODL bare transportere data i form av et enkelt objekt eller et objektsett. Koblinger som leder til interaktorens venstre side

transporterer data *inn til* interaktoren, koblinger som leder til interaktorens over- eller underside, transporterer data *fra* interaktoren.

En stiplet kobling markerer at dataene som transporteres kan være ‘null’.

Funksjoner

Alle koblinger leder i ProtoMoDL fra en interaktor til en annen. En kobling må imidlertid passere en funksjon. Denne funksjonen angir systeminteraksjonen, dvs hvordan systemet manipulerer dataene og kontrollerer datastrømmene. Hva funksjonen gjør, angis i en kort beskrivelse. Dette kan for eksempel være “Beregn verdi” eller “Presenter data”. En funksjon understreker samtidig datastrømmens retning, ettersom data som stammer fra en interaktor alltid ender i basen på en funksjon, og data som skal inn til en interaktor alltid kommer fra tippen på en funksjon.

På samme måte som i DiaMODL, kan en mer kompleks oppførsel enn beregning av en enkelt verdi indikeres ved at tuppen og basen til funksjonssymbolet er atskilt. Dersom resultatet av en funksjon ikke blir brukt lokalt i systemet, kan dette indikeres ved at tuppen utelates helt i notasjonen.

I enkelte tilfeller er det nødvendig å vise at en funksjon eller metode venter på input fra en bruker før den ferdigbehandlede verdien sendes videre. Dette indikeres ved at en kobling leder fra en interaktor og inn til siden av funksjonen. Slike koblinger benevnes som ‘avtrekkere’.

Data som ikke er påkrevd for at en funksjon skal kunne utføre sin oppgave, angis ved at koblingen mellom kildeinteraktoren og funksjonens base er stiplet. Slike data kan for eksempel stamme fra felter i et kunderegistreringsskjema som ikke er nødvendige for at registreringen skal kunne gjennomføres.

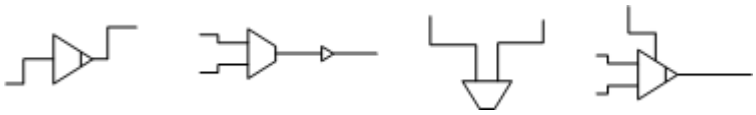


FIGURE 5-4: Basisfunksjoner i ProtoMODL

Metoder

En metode er en funksjon som endrer eller virker direkte inn på det objektet eller objektsettet som målinteraktoren representerer. Dette vises ved at koblingen som leder fra metoden ender i en svart pilspiss som peker inn mot målinteraktorens venstre side.

FIGURE 5-5: Metodekonseptet i ProtoMODL

Diagramkonstruksjon

Når man skal lage et ProtoMODL-diagram, anbefales det å ta for seg de mest sentrale interaktorene først. Man må da spørre seg hvilke data eller objekter som er viktigst for interaksjonen med brukeren, og hvordan disse kan representeres som interaktorer. Deretter må man spesifisere interaksjonen mellom brukeren og interaktoren. Dette angis i form av en interaktorbeskrivelse sammen med et interaktorsymbol. Man spesifiserer så hvordan de forskjellige interaktorene skal samhandle og hvordan de utveksler data. Datautvekslingen skjer alltid via funksjoner eller metoder, og det er viktig å identifisere hvilke data som inngår som parametre til de forskjellige funksjonene og metodene.

ProtoMODL som en generisk abstrakt prototyp

ProtoMODL-interaktorer kan benyttes til å bygge opp en abstrakt prototyp. Ettersom hver interaktor representerer en brukergrensesnittkomponent, eller en gruppe av slike komponenter, er det mulig å bruke de til å identifisere hvilke komponenter som må være med i ei brukergrensesnitt og den innbyrdes plasseringen. En ProtoMODL prototyp likner til forveksling en kanonisk abstrakt prototyp, men interaktorbeskrivelsen angir brukerens aksjon mot brukergrensesnittet i tillegg til elementet som presenteres for brukeren. De tre spesialsymbolene som angir hhv. repetisjon, konseptuell tilhørighet og oppklarende tekst er ikke tatt med i notasjonen til ProtoMODL.

Eksempel

Det skal lages en applikasjon som gjør det mulig å opprette et nytt sangobjekt, eller endre et eksisterende sangobjekt dersom det allerede finnes. Et sangobjekt skal inneholde attributtene ‘sangtittel’, ‘gruppe/artist’, ‘sjanger’ og ‘anbefales’. Det er ikke angitt om systemet består av flere brukergrensesnitt enn dette. Det er heller ikke spesifisert hvilke initielle data som skal presenteres for brukeren.

FIGURE 5-6: Skjermbildeprototyp

ProtoMODL prototyp

Ut fra skjermbildeprototypen kan det lages en abstrakt ProtoMODL prototyp. Denne representasjonen angir brukerens handlinger mot brukergrensenittet, og abstraherer de spesifiserte komponentene til interaktorer. I prinsippet er dette en kanonisk abstrakt prototyp, men med en større dekningsgrad ettersom denne prototypen også spesifiserer brukerens interaksjon på de enkelte komponentene.

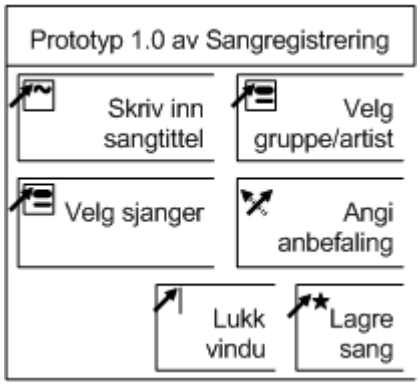


FIGURE 5-7: Eksempel på en abstrakt ProtoMODL prototyp

Alt på dette stadiet må designerne tenke gjennom hvordan systemet skal samhandle med brukeren; hvordan systemet skal respondere på bestemte handlinger og hvilke data som skal presenteres for brukeren. Layouten som er spesifisert i skjermbildeprototypen er bevart.

ProtoMODL diagram

Selve samspillet mellom de forskjellige interaktorene spesifiseres så i et ProtoMODL diagram. Interaktorenes innbyrdes plassering er nå fristilt i forhold til ProtoMODL prototypen og skjermbildeprototypen. Dekningsgraden er den samme, men fokuset er flyttet fra layout til dataflyt. Også abstraksjonsgraden øker på grunn av dette skiftet i fokus. Diagrammet kan nå klassifiseres både som en papirprototyp av typen ‘innholdsmodell’ og som en dialogmodell med lav abstraksjonsgrad.

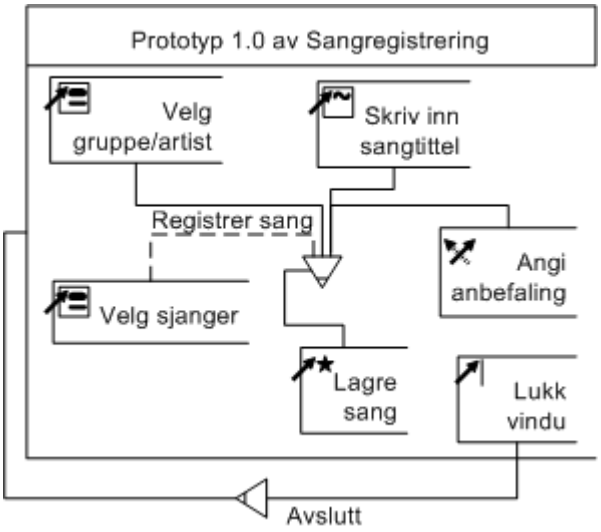


FIGURE 5-8: Eksempel på et simpelt ProtoMODL diagram

I dette diagrammet er det spesifisert hvilke abstrakte komponenter som skal inngå i brukergrensesnittet, hvordan brukeren samhandler med komponentene og hvordan samspillet mellom de forskjellige komponentene foregår, mens brukergrensesnittets layout er abstrahert vekk.

Forklaring til diagrammet

For å opprette eller endre et sangobjekt, kreves det at brukeren har angitt en sangtittel i interaktoren ‘Skriv inn sangtittel’. Interaktorsymbolet angir at innholdet i denne interaktoren skal kunne redigeres. Videre kreves det at sangobjektet knyttes opp mot en gruppe eller artist. Dette skal gjøres via interaktoren ‘Velg gruppe’ som presenterer et sett og lar brukeren velge et enkelt element fra settet. Sangobjektet kan også knyttes opp mot en bestemt sjanger. Dette gjøres i interaktoren ‘Velg sjanger’. Den stiplete koblingen som leder fra interaktoren viser at dette ikke er et påkrevd attributt for sangobjektet. Brukeren kan i tillegg angi sin personlige anbefaling ved å “slå på” interaktoren ‘Angi anbefaling’.

Ved å aktivere interaktoren ‘Lagre sang’, avfyres funksjonen ‘Registrer sang’. Funksjonen har tre påkrevde parametre, pluss en parameter som kan være null. Ved å aktivere interaktoren ‘Lukk vindu’, avsluttes registreringen og vinduet lukkes.

I dette eksempelet påvirker ingen av interaktorene innholdet til noen av de andre interaktorene. Alle interaktorene har derfor bare koblinger som leder ut av interaktoren.

Beskrivelsen av diagrammet tilsvarer omtrent den mengde tekst som behøves for å forklare interaksjonen i en skjermbildeprototyp på en entydig måte. Dette eksempelet demonstrerer at selv et svært enkelt brukergrensesnitt krever mye tekst dersom man skal beskrive interaksjonen nøyaktig. Et ProtoMODL-diagram kan altså brukes til å vise det samme på en mer kompakt og utvetydig måte.

Nøstede interaktorer

Ofte vil man få bruk for å benytte nøstede interaktorer. En interaktor som er plassert inne i en annen kalles en ‘subinteraktor’. Den utenforliggende interaktoren er da en ‘superinteraktor’. Subinteraktorer kan fritt hente data fra sin superinteraktor, og koblinger som leder fra en super- til en subnteraktor kan derfor utelates fra diagrammet. Skal en subinteraktor sende data ut til sin superinteraktor, eller en annen interaktor, må dette angis med koblinger og funksjoner på vanlig måte.

Neste diagram viser hva som kreves av et brukergrensesnitt dersom man skal bestille time hos en frisør. Brukeren velger en dato fra et sett av datoer. Dette aktiverer en liste som inneholder alle timeavtalene for denne datoen, og hvor de ledige timene er markert. Hver enkelt timeavtale er bygd opp av en time, en frisør, og en oppgave. Når brukeren har valgt en time, vil en liste over frisører komme opp og tilgjengelige frisører på dette tidspunktet markeres. Når så en frisør er valgt, vil eventuelle oppgaver for denne frisøren på det aktuelle tidspunktet komme opp i form av et editerbart element. Diagrammet spesifiserer på denne måten at det også skal være mulig å hente fram timeavtaler som allerede er registrert og endre disse. Til slutt registeres/endres den gjeldende timeavtalen ved å aktivere interaktoren ‘Registrer time’. Metoden ‘registrer timeavtale’ viser at denne aktiveringen skal føre til en endring på det gjeldende elementet i interaktoren ‘Timeavtaler’.

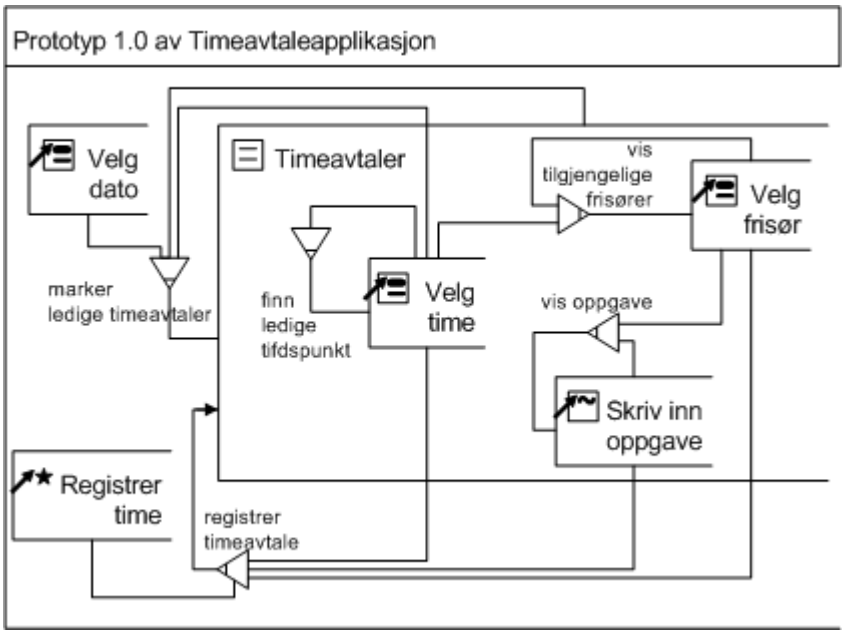


FIGURE 5-9: Nøstede interaktorer i ProtoMODL

Systeminteraktorer

Interaktorer som ikke er en del av selve brukergrensesnittet, men som allikevel påvirker hvordan andre interaktorer håndterer eller presenterer data, kalles systeminteraktorer. Systeminteraktorer kan ikke samhandle med brukeren, men representerer for eksempel forskjellige filtre, betingelser som gjør at komponenter er inaktive, og annet som er spesifisert i selve systemet. Systeminteraktorer angis i et ProtoMODL-diagram ved å bruke DiaMODL-notasjonen til å tegne interaktorene.

FIGURE 5-10: Systeminteraktor

5.2 En brobygger mellom representasjonene

Det kan være vanskelig å overføre informasjonen i en abstrakt prototyp til et DiaMODL-diagram, eller omvendt. Ettersom ProtoMODL er en hybrid av disse to representasjonene, kan den fungere som en brobygger mellom dem. Vi får da fire overganger mellom de forskjellige representasjonen.

ProtoMODLs plassering i sandwichmodellen

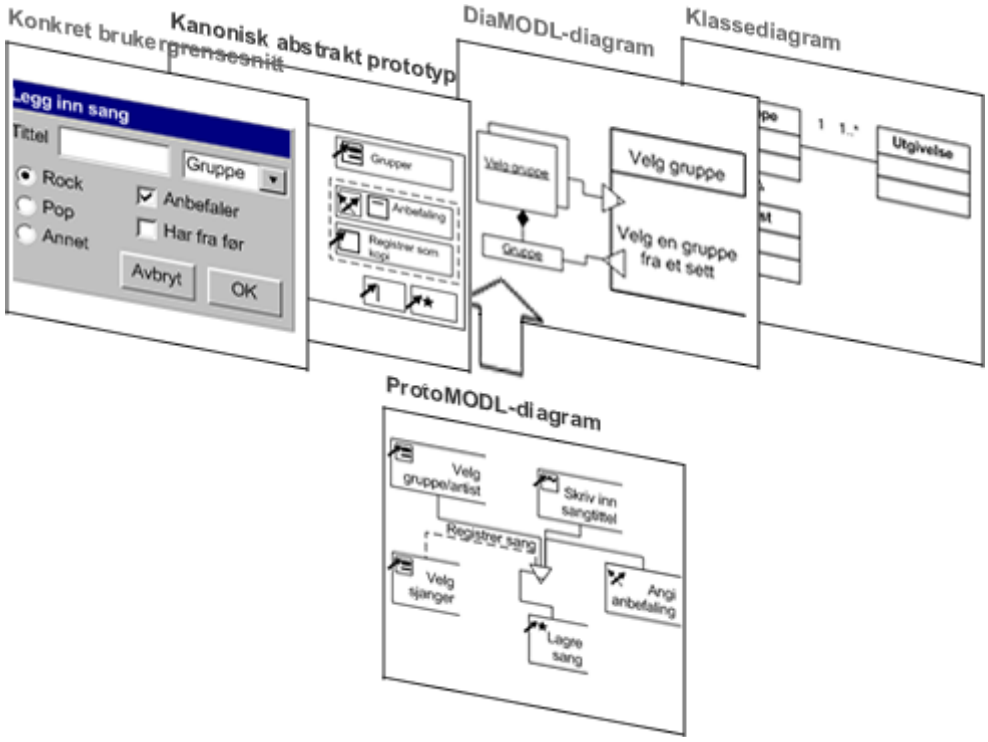


FIGURE 5-11: ProtoMODLs plassering i sandwichmodellen

Fra abstrakt prototyp til ProtoMODL-diagram

I et tidligere eksempel ble det vist hvordan man kan overføre informasjonen i en skjermbildeprototyp, via en abstrakt ProtoMODL prototyp, til et ProtoMODL-diagram. Dette er en relativt enkel transisjon, ettersom en brukergrensesnittkomponent som regel kan abstraheres ned til en mer generisk abstrakt komponent, som altså tilsvarer en enkelt interaktor. Dersom interaksjonen er spesifisert for eksempel i form av en tekstlig beskrivelse, er det relativt enkelt å lage koblinger mellom de resulterende interaktorene. Hvilke data som skal presenteres for brukeren, og hvilke handlinger brukeren utfører på interaktoren er også relativt opplagt. Den vanskeligste operasjonen er å identifisere datastrømmene mellom interaktorene, og hvilke funksjoner og metoder som kreves for å beregne og presentere gyldig data til brukeren.

Fra ProtoMODL-diagram til abstrakt prototyp

ProtoMODL skal kunne brukes som et designverktøy, det vil si at det skal være mulig å lage et ProtoMODL-diagram som spesifiserer den ønskede interaksjonen for så å generere en prototyp ut fra diagrammet. Den vanskelige biten her er å konstruere selve ProtoMODL-diagrammet. Overgangen fra ProtoMODL-diagram til ProtoMODL prototyp er derimot relativt enkel: Ved å fjerne alle koblinger, funksjoner og metoder som inngår i diagrammet, sitter man igjen med et skall og de interaktorer som skal inngå i brukergrensesnittet. Ettersom hver interaktor tilsvarer en enkelt abstrakt komponent eller komponentgruppe, kan man flytte interaktorene rundt til man finner en layout som oppleves bra. Den resulterende abstrakte prototypen kan så testes direkte på brukere, eller man kan lage en presis papirprototyp basert på den abstrakte prototypen.

Tar man utgangspunkt i ‘Timeavtaleapplikasjonen’ over, kan man tenke seg en layout som kan representeres slik:

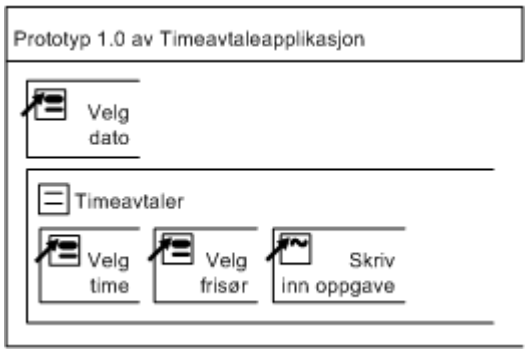


FIGURE 5-12: Abstrakt ProtoMODL prototyp basert på ProtoMODL-diagram

Her vil interaktoren ‘Velg dato’ typisk representere en rullegardin. Men man kan også tenke seg at interaktoren representerer en link eller knapp som åpner en kalenderkomponent. Interaktorsymbolet til interaktoren ‘Timeavtaler’ viser at innholdet av denne interaktoren skal være en samling av elementer. Hvert element er så definert av de tre subinteraktorene ‘Velg ledig time’, ‘Velg frisør’ og ‘Skriv inn oppgave’. ‘Velg ledig time’ kan for eksempel representere en rullegardin hvor bare de ledige tidspunktene presenteres, eller en liste av lenker der for eksempel fargekode angir hvilke tidspunkt som er ledige. ‘Velg frisør’ vil typisk være en rullegardin eller en liste, men man kan også tenke seg at en liste over ledige frisører presenteres i form av et pop-up vindu. ‘Skriv inn oppgave’ vil som regel representere et tekstfelt eller tekstareal.

En skjermbildeprototyp basert på den abstrakte ProtoMODL prototypen, kan for eksempel bli seende ut slik:

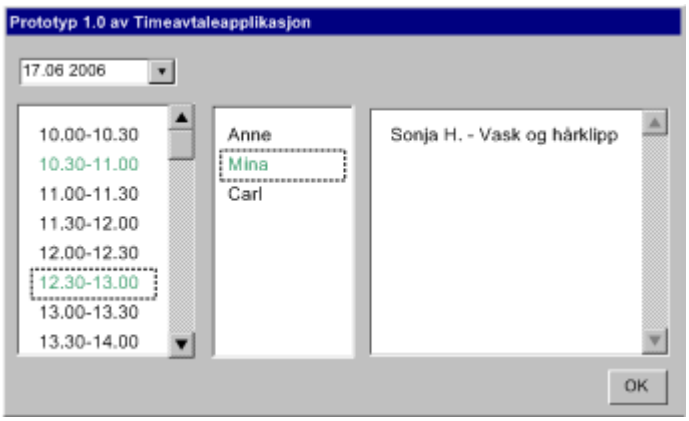


FIGURE 5-13: Skjermbilde basert på abstrakt ProtoMODL prototyp

Her er interaktoren ‘Velg dato’ implementert som en rullegardin, ‘Velg time’ og ‘Velg frisør’ er implementert som lister av klikkbare elementer, der tilgjengelige tidspunkt er markert med grønn tekstfarge. Når man velger et element, markeres dette med en stiplet boks rundt det valgte elementet. Interaktoren ‘Skriv inn oppgave’ er implementert som et tekstareal. Vi ser at interaktoren ‘Timeavtaler’ på denne måten er realisert ut fra de gjeldende elementene i de tre subinteraktorene.

Fra ProtoMODL-diagram til DiaMODL-diagram.

Noen ganger er det ønskelig å presisere dataflyten mellom interaktorene og de objektene som skal presenteres og/eller manipuleres. Denne informasjonen er i ProtoMODL spesifisert i interaktorsymbolene, som er en mindre formell notasjon enn strukturen med objekter og objektsett som man finner i DiaMODL.

Når man skal lage et DiaMODL-diagram basert på et ProtoMODL-diagram, vil hver ProtoMODL-interaktor representere en enkelt DiaMODL-interaktor.

I ProtoMODL sier interaktorbeskrivelsen hvordan brukeren samhandler med dataene representert via interaktoren, mens i DiaMODL angir interaktorens navn kun hvilke data interaktoren inneholder.

Funksjonen og metodene, samt koblingene som leder til og fra dem stemmer også overens med notasjonen i DiaMODL. Informasjonen som ligger i interaktorsymbolene og interaktorbeskrivelsen må imidlertid oversettes til objekter og -sett, og til strukturen mellom disse og mellom dem og interaktorene. En kobling som leder til en ProtoMODL-interaktor, vil dermed ikke nødvendigvis lede til en interaktor i DiaMODL, men kan også lede til et objekt eller objektsett.

Tar vi utgangspunkt i ProtoMODL-diagrammet over som spesifiserer hvordan interaksjonen skal foregå i applikasjonen “Sangregistrering”, vil tilsvarende DiaMODL-diagram bli seende ut som dette:

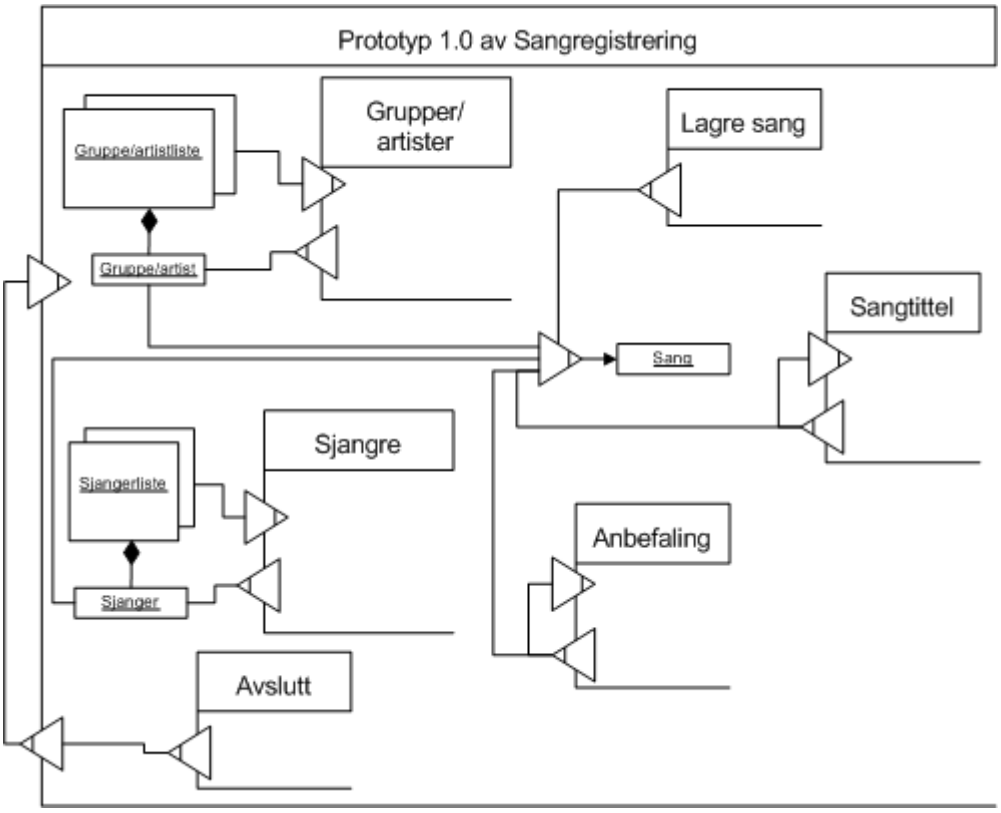


FIGURE 5-14: DiaMODL-diagram basert på ProtoMODL-diagram.

Fra DiaMODL-diagram til ProtoMODL-diagram

I kapittel 3.4 ble det gjennomgått et eksempel som tok for seg hvordan en enkel nettleser kan modelleres. Denne modellen forklarte datastrømmene mellom interaktorene, men sa lite om hvordan brukeren samhandler med systemet.

Ønsker man å lage et ProtoMODL-diagram basert på et DiaMODL-diagram, vil vær DiaMODL-interaktor tilsvare en ProtoMODL-interaktor. Objektsett og enkeltobjekter oversettes til passende kanonisk abstrakte komponenter, og legges inn i den interaktoren de tilhører. Når man så skal koble interaktorene sammen, vil de interaktorene som er direkte sammenkoblet i DiaMODL-diagrammet også være det i ProtoMODL-diagrammet.

Når det gjelder koblinger som i DiaMODL-diagrammet leder til eller fra objekter eller objektsett, må man se på interaksjonen som formidles. I DiaMODL vil ofte en kobling gå fra en interaktor via et objekt eller -sett til en annen interaktor. Disse må identifiseres og trekkes opp i ProtoMODL-diagrammet.

Til slutt oppretter man en funksjon eller en metode til hver kobling, De funksjoner som er spesifisert i DiaMODL-diagrammet vil som regel også kunne overføres direkte i ProtoMODL-diagrammet.

ProtoMODL-diagrammet av interaksjonen i en enkel browser vil da bli seende ut slik som dette:

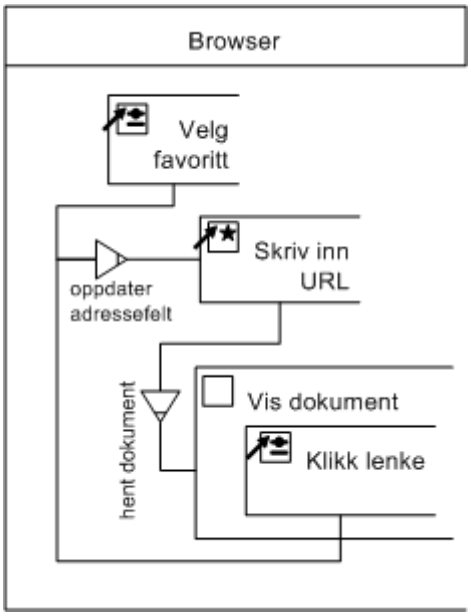


FIGURE 5-15: ProtoMODL-diagram basert på DiaMODL-diagram

5.3 Oppsummering

ProtoMODL er en hybrid notasjon som plasserer seg mellom den kanonisk abstrakte prototypen og DiaMODL med hensyn på dekningsgrad og formalitet. Den kan både benyttes som en ren interaksjonsmodell, en abstrakt prototyp eller en kombinasjon av disse to.

Hypotetisk skal denne representasjonen gjøre det enklere å beskrive interaksjon mellom bruker og system, og mellom de forskjellige komponentene som inngår i brukergrensesnittet, på en mer entydig måte. Slik skal det bli enklere å utforske, dokumentere og kommunisere interaksjonen i brukergrensesnittet.

For å se om dette stemmer, er det nødvendig å utføre empiriske eksperimenter som kan teste denne hypotesen. Et slikt eksperiment beskrives i neste kapittel.

Kapittel 6

Empiri

Dette kapittelet beskriver oppgavens empiriske grunnlag; et kvalitativt eksperiment som undersøker i hvilken grad skjermbilder, abstrakte prototyper, ProtoMODL- og DiaMODL-diagrammer gjør det mulig å tolke interaksjonen i et brukergrensesnitt. Slik finner vi først og fremst ut hvilke av disse representasjonene som er i stand til å avklare uklarheter og fjerne tvetydigheter i brukergrensesnittets design, men resultatet gjør oss også i stand til å avdekke styrker og svakheter i notasjonen til ProtoMODL.

6.1 Et kvalitativt eksperiment

Main Entry: **[1]ex·per·i·ment**
Pronunciation: ik-'sper-&-m&nt *also* -'spir-
Function: *noun*
Etymology: Middle English, from Middle French, from Latin *experimentum,* from *experiri*
1 a : TEST, TRIAL <make another *experiment* of his suspicion - - Shakespeare> **b :** a tentative procedure or policy **c :** an operation carried out under controlled conditions in order to discover an unknown effect or law, to test or establish a hypothesis, or to illustrate a known law
2 *obsolete* **:** EXPERIENCE
3 : the process of testing **:** EXPERIMENTATION

FIGURE 6-1: Fra Webster Online Dictionary: 'ex-per-i-ment'

Hensikten med eksperimentet

Under utviklingen av ProtoMODL dukket det opp flere usikkerhetsmomenter knyttet til selve notasjonen. Spesielt fire av disse er viktig å undersøke nærmere, nemlig: Hvor oversiktlig fremstår et ProtoMODL-diagram for en person med middels eller bedre kompetanse innen prototyping og/eller modellering av brukergrensesnitt; hvor tydelig fremstår retningen til datastrømmene til og fra interaktorene framstår som tydelige nok; hvor stor betydning har de abstrakte komponentene for forståelsen av et diagram skrevet i denne notasjonen; hvilken betydning har navngivningen av interaktorene for leserens forståelse av brukerense interaksjon mot systemet.

Siden ProtoMODL bygger på både den kanonisk abstrakte prototypen og DiaMODL, er det også ønskelig å se om det er noen vesentlig forskjell på hvor godt ProtoMODL fungerer som design-, analyse- og dokumentasjonsverktøy, i forhold til disse representasjonene.

Hensikten med eksperimentet er altså både å undersøke brukskvaliteten til ProtoMODL sammenlignet med andre representasjoner av interaksjonen i et brukergrensesnitt og å avdekke styrker og svakheter ved ProtoMODL-notasjonen.

6.2 Eksperimentets form og innhold

Grunnlaget for selve eksperimentet er et konstruert case. Det gjør det mulig å legge inn kontrollerte uklarheter og tvetydigheter i brukergrensesnittet. Flere case tatt fra virkeligheten ble i forkant av denne beslutningen undersøkt, men ingen av dem klarte å oppfylle kravene som ble satt til kontroll av eksperimentet.

Testdeltakere

Siden det er metoden og ikke verktøyet som skal testes, er det essensielt at man får tak i testdeltakere til gjennomføringen av eksperimentet som er kjent med notasjonen i den kanonisk abstrakte prototypen og DiaMODL, og som har praktisk erfaring i å tegne diagrammer ved hjelp av disse notasjonene.

Når det gjelder selve utførelsen av eksperimentet ble det foretatt avveininger om datagrunnlaget ville bli mest nøyaktig om en og en testdeltaker utførte oppgavene med å tolke og modellere interaksjonen hver for seg, eller om det var best at to og to personer gjorde dette sammen og diskuterte løsningene i fellesskap. Fordelen med å bare benytte en testdeltaker av gangen er at man da kan få gjennomført dobbelt så mange tester og dermed øke datagrunnlaget. Ved å benytte to testdeltagere vil det på den annen side være mye lettere å dokumentere deres tankegang rundt abstrahering og tolkning av de forskjellige representasjonene på et mer naturlig sett, noe som det ble antatt vil øke datagrunnlagets *kvalitet*. I tillegg antas det at sjansen er større for å avdekke andre interessante momenter ved enten representasjonene som blir testet eller selve eksperimentet når man kan studere hvordan flere personer løser oppgavene i fellesskap, enn når testdeltakerene arbeider hver for seg. Disse momentene er såpass tungtveiende at det ble besluttet å utføre eksperimentet med testdeltakere satt sammen i par.

Valg av testdeltakere

Mulige kandidater ble gått gjennom sammen med veileder. Av fire potensielle kandidater som hadde de nødvendige kvalifikasjonene, var det kun to som både var villige og hadde anledning til å stille opp. Selv om bare to testdeltakere vil gi et snevert datagrunnlag, er det fortsatt såpass mange momenter som kan observeres at det fortsatt er interessant å gjennomføre eksperimentet.

Gjennomføring

Når det gjelder selve gjennomføringen av eksperimentet, ble det først vurdert å la et par med testdeltakere lage en modell av interaksjonen i et design, og så la et annet par med testdeltakere tolke disse modellene for å se om deres oppfatning av interaksjonen stemmer overens med interaksjonen i det faktiske brukergrensesnittet. Begrensninger i antall personer som kan tenkes å både ha nok kompetanse til å kvalifisere som testdeltakere og ha tid til å stille opp, samt usikkerhetsmomenter knyttet til nøyaktigheten av de dataene som ville bli innsamlet, gjør derimot at et eksperiment på denne formen antas å være uegnet.

Eksperimentet vil derfor gå ut på at testdeltakere får i oppgave å tolke fire forskjellige representasjoner av et brukergrensesnitt med hensyn på brukergrensesnittets interaksjon. Deretter skal de selv modellere en utvidelse av interaksjonen ved hjelp av en valgfri notasjon. De representasjonene som skal tolkes er henholdsvis en skjermbildeprototyp, en kanonisk abstrakt prototyp, et ProtoMODL-diagram og et DiaMODL-diagram. Etter at disse oppgavene er utført vil det bli gjennomført en debriefing, der testdeltakerene blir stilt en del spørsmål av testleder for å bekrefte eller avkrefte de observasjoner som er gjort underveis.

Forberedelser

Det er viktig at testdeltakerene får en eksperimentsbeskrivelse hvor eksperimentets hensikt og form forklares, notasjonen til CAP og DiaMODL repeteres og notasjonen til ProtoMODL gjennomgås. Til eksperimentets første del trengs en skjermbildeprototyp, en kanonisk abstrakt prototyp, et ProtoMODL-diagram og et DiaMODL-diagram som representerer det samme brukergrensesnittet. Til eksperimentets andre del trengs en skjermbildeprototyp som viser hvordan brukergrensesnittet vil bli seende ut etter en utvidelse av funksjonaliteten til den representerte applikasjonen.

Oppstart

Før eksperimentet tar vil det opplyses om at en del informasjon om interaksjonen er ubeskrevet, ettersom en viktig side av eksperimentet går ut på å tolke interaksjonen som er beskrevet i de forskjellige representasjonene av brukergrensesnittet.

Når eksperimentet først har startet, skal testleder så langt det er mulig være en helt passiv observatør. Dette innebærer å bare svare på spørsmål som går direkte på notasjonene og bruken av dem, og kun gripe inn i de tilfeller hvor det er fare for breakdown. Dette skal også testdeltakerene opplyses om.

6.3 Antakelser om datagrunnlag, observasjoner og funn

Datagrunnlaget

Datagrunnlaget til eksperimentet skal dokumenteres i form av: Testleders notater rundt de observasjoner som blir gjort av hvordan testdeltakerene arbeider med oppgavene og hvordan de bruker de forskjellige representasjonene; video som tas opp under utførelsen av eksperimentet og den etterfølgende debrifingen; og de

notatene, skissene og modellene som blir produsert av testdeltakerene underveis i eksperimentet.

De innsamlede dataene bør først og fremst gi grunnlag for å si noe om designrepresentasjonenes evne til å spesifisere og dokumentere interaksjon, ved å studere hvilke representasjoner som er enklest å tolke for testdeltakerene og hvilke de velger å benytte når de skal beskrive interaksjonen selv. I tillegg bør de gjøre det mulig å beskrive de forskjellige representasjonenes evne til å støtte oppunder kommunikasjonen mellom testdeltakerene.

Når det gjelder ProtoMODL, er det lagt opp til at datagrunnlaget skal være såpass bra at man kan trekke slutninger om kvaliteten til notasjonen, det vil si om den er i stand til å beskrive interaksjonen i brukergrensesnittet.

Mulige problemer knyttet til datagrunnlaget

Flere momenter rundt datagrunnlaget og innsamlingen kan ha innvirkning på resultatet av eksperimentet. Blant disse er de viktigste: Om selve datagrunnlaget vil bli solid nok, dvs. om det vil bli samlet inn nok materiale; om det vil bli brukt tilstrekkelig antall testdeltakere til å kunne si at man har et reelt empirisk grunnlag; og om testdeltakerene har god nok kompetanse til at eksperimentets målsetninger kan bli oppfylt.

Eksperimentets første del: Tolkning av interaksjon

Testdeltakerenes første oppgave i eksperimentet vil være å tolke interaksjonen i et brukergrensesnitt, representert ved hjelp av ulike designrepresentasjoner. Ut fra en kvalifisert gjetning om at *interaksjonen trer fram tydeligere jo mer abstrakt representasjonen er,* er det bestemt at representasjonene blir presentert for testdeltakerene i denne rekkefølgen: Skjermbildeprototyp uten tekstforklaring, kanonisk abstrakt prototyp, ProtoMODL-diagram og DiaMODL-diagram.

Trer interaksjonen klarere fram etter hvert?

Det mest interessante med den første delen er å å observere om interaksjonen trer fram klarere for testdeltakerene etter hvert som dekningsgraden faller og formalitets-, abstraksjons- og abstraheringsgraden øker. Med andre ord: Om representasjonene formidler en gradvis økning av kvalitativ og kvantitativ informasjon om brukergrensesnittets interaksjon til testdeltakerenedeltakerene.

Dersom det finnes tvetydigheter, og en representasjon gjør at testdeltakerene tolker interaksjonen til grensesnittet slik som designeren har ment den skal være, er dette en god indikasjon på at en slik representasjon er i stand til å fjerne tvetydigheter knyttet til interaksjonen i brukergrensenittet.

Dette vil i sin tur kunne indikere hvilken evne representasjonene har til å avdekke tvetydigheter og tydeliggjøre uklarheter. Noe som igjen kan gjøre det mulig å si noe om representasjonenes nytteverdi i forhold til å kommunisere interaksjon til andre personer, og om nytteverdien i forhold til å spesifisere og dokumentere interaksjon ved hjelp av disse representasjonene.

Dersom representasjonene ikke klarer å øke testdeltakerenes forståelse av interaksjonen slik som beskrevet over, vil eksperimentets første del måtte sies å være uten empirisk verdi på dette området, siden et av eksperimentets grunnpremisser var tilbakebevist.

I hvilken grad gir de forskjellige representasjonene rom for tolkning?

Det er også ønskelig å undersøke om det er mulig, ut fra observasjonerne, å slå fast om noen av representasjonene gir rom for tolkning når man ønsker å vurdere interaksjonen som ligger bak. Dersom en eller flere av testdeltakerene, etter å ha blitt presentert for en representasjon, enten tolker en komponent, eller funksjonaliteten til en komponent, forskjellig fra designerens intensjon eller finner flere mulige tolkninger av funksjonaliteten til en komponent, vil dette vise over enhver tvil at denne representasjonen av brukergrensesnittet er tvetydig.

Er ProtoMODL en oppklarende notasjon?

Når det gjelder ProtoMODL, er det flere usikkerhetsmomenter rundt hvordan dataflyten, symbolbruken og navngivingen i notasjonen er i stand til å formidle interaksjonen i et brukergrensesnitt. Et annet usikkerhetsmoment er hvor oversiktelig notasjonen virker på testdeltakerene. Dersom det oppstår forvirring rundt hvordan man skal tolke interaksjonen i ProtoMODL-diagrammet, er dette en klar indikasjon på at notasjonen må endres.

Eksperimentets andre del: Modellering av interaksjon

Testdeltakerenes andre oppgave vil være å modellere den endringen i interaksjon som oppstår etter at brukergrensesnittet blir utvidet. Utvidelsen skal presenteres i form av et nytt skjermbilde, der brukergrensesnittkomponenter er lagt til, endret og fjernet. Noen nærmere beskrivelse av interaksjonen vil ikke bli utlevert.

Hvilken representasjon foretrekkes av testdeltakerene?

For å få en indikasjon på hvilken representasjon testdeltakerene foretrekker eller mener er best egnet, skal de få lov til å bestemme selv hvilken representasjon de ønsker å benytte. Hvilken notasjon testdeltakerene velger å bruke vil gi indikasjoner på hvilken representasjon som framstår som mest brukervennlig. Dersom de velger å benytte en annen representasjon enn de to dialogmodellene å modellere i kan dette indikere at testdeltakerene anser verken DiaMODL eller ProtoMODL som egnet til denne oppgaven.

Er det en kvalitativ forskjell mellom ProtoMODL og DiaMODL?

Uansett hvilken representasjon testdeltakerene velger å benytte i første omgang, er det viktig at de modellerer interaksjonen i utvidelsen ved hjelp av både ProtoMODL og DiaMODL. Slik er det mulig å undersøke hvordan de to representasjonene fungerer som design- og utviklingsverktøy.

Ettersom problemdomenet vil være kjent når interaksjonen modelleres for andre gang, er det imidlertid ikke mulig å foreta en direkte kvalitativ eller kvantitativ sammenligning av de to representasjonene. Derimot antas det at observasjoner av eventuelle forskjeller eller likheter i måten testdeltakerene bruker de to modelleringsspråkene på, kan si noe om forholdet mellom dem.

Er notasjonene vanskelig å bruke?

Et annet moment som kan oppstå, er at testdetakerne får problemer med å lage representasjoner som beskriver interaksjonen rett. Dette kan da være en indikasjon på at notasjonene er vanskelig å bruke i praksis.

Påvirker modelleringsarbeidet testdeltakerenes forståelse av interaksjonen?

Aller mest interessant i denne delen av eksperimentet vil det imidlertid bli å se i hvilken grad selve modelleringsarbeidet påvirker testdeltakerenes forståelse av interaksjonen som framstår i skjermbildeprototypen.

Dersom forståelsen av interaksjonen øker under modelleringen, er dette en indikasjon på at notasjonen(e) tydeliggjør interaksjonen og tvinger brukerne til å tenke gjennom problemdomenet grundigere enn hva som er tilfellet med et rent skjermbilde.

Er protoMODL en god notasjon å spesifisere eller dokuemntere interaksjon med?

For ProtoMODL sin del er det interessant å undersøke om observasjonene rundt det å benytte notasjonen til å modellere interaksjon er sammenfallende med de observasjoner som blir gjort når testdeltakerene tolker interaksjon ut fra den.

Dersom testdeltakerene har vanskeligheter med å modellere interaksjon med denne notasjonen, vil dette også være en indikasjon på at notasjonen bør endres.

Falsifisering av hypotese

Ut fra eksperimentets datagrunnlag bør det til slutt være mulig å gjøre seg en mening om eksperimentet har avkreftet hypotesen som ligger i bunn av denne oppgaven eller ikke.

Dersom eksperimentet skulle vise at ProtoMODL ikke er i stand til verken å hjelpe en tilstrekkelig kompetent person, her definert som en person med basiskunnskap om og erfaring i bruk av både interaksjonsmodeller og abstrakte prototyper, med å forstå interaksjonen i et brukergrensesnitt, eller avklare identifiserte tvetydigheter i et brukergrensesnitt, må man slutte at hypotesen er falsifisert. Det må i så fall undersøkes om dette skyldes momenter ved reprersentasjonens notasjon, eller om det er noe feil ved selve grunnideen til ProtoMODL.

6.4 En prototyp av applikasjonen Fotoeditor

Beskrivelse av prototypen

Skjermbildeprototypen forestiller et førsteutkast til en applikasjons brukergrensesnitt. Applikasjonen skal la en bruker endre navnet til et bilde, legge til attributter til et bilde eller søke etter bilder med et gitt attributt. I tillegg kan det gjeldende bildet slettes eller vendes.

Selve brukergrensesnittet er delt inn i forskjellige arealer: Et hvor brukeren velger hvilken oppgave som applikasjonen skal utføre, et hvor brukeren kan tilordne verdier til et bildeattributt (hvor bildet er tatt, hvem det er tatt bilde av, hvilken situasjon bildet er tatt i, osv), et hvor man utfører grafiske endringer på det gjeldende bildet,

et som viser brukerens mappetre, et som viser bildefilene i den gjeldende mappa, og et som viser det gjeldende bildet.

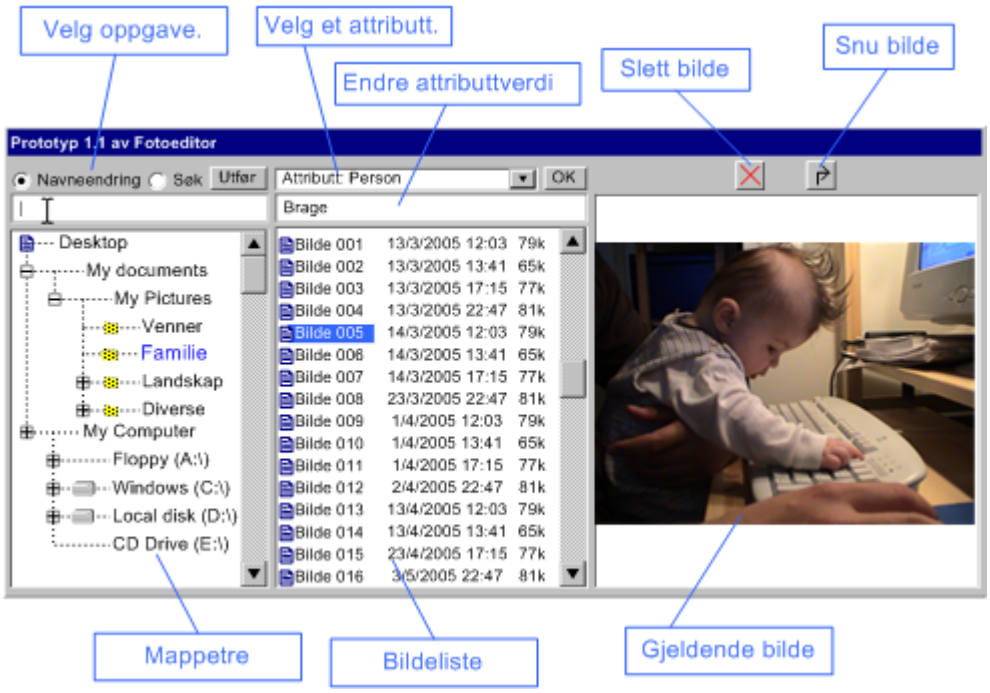


FIGURE 6-2: Skjermbildeprototyp av applikasjonen "Fotoeditor"

Når trykknappen 'Utfør' aktiveres, og radioknappen 'Navneendring' er markert, vil det gjeldende bildets navn tilordnes verdien som er satt i tekstfeltet under disse komponentene. Dersom i stedet radioknappen 'Søk' er valgt, vil det som er skrevet i tekstfeltet sammenlignes med verdiene av et bildes attributter. Det første bildet i lista som har en matchende attributtverdi vil bli markert og vist i bilderammen.

Når trykknappen 'OK' aktiveres, vil den attributten som er valgt i nedtrekksboksen få verdien som er satt i tekstfeltet under. Flere verdier til samme attributt skilles med komma.

Når trykknappen som er markert med et rødt kryss aktiveres, slettes det gjeldende bildet fra mappa. Når trykknappen markert med høyrevridd pil aktiveres, blir det gjeldende bilet vridd 90 grader mot høyre. Tre trykk på knappen vrir dermed bildet 90 grader til venstre.

Når en mappe markeres, vises alle undermapper til den valgte mappa i mappetreet, og alle filer i den markerte mappa identifisert som bilder vises i fillista til høyre. Når applikasjonen startes opp, er det øverste elementet i mappetreet markert. Brukeren kan benytte både mus og tastatur til å velge ei mappe. Ved å trykke 'pil opp' eller 'pil ned' på tastaturet, kan henholdsvis foregående eller neste mappe markeres.

Når en bildefil blir markert, vises det gjeldende bildet i bilderamma til høyre og navnet på bildefila kommer opp i tekstfeltet over fillista. Når en ny mappe markeres, markeres det første bilde i denne mappa dersom denne mappa inneholder minst en bildefil

Tvetydigheter og uklarheter

Selv til dette brukergrensesnittet, som inneholder relativt få komponenter, kreves det en forholdsvis lang tekst for å beskrive interaksjonen.[1]

Denne skjermbildeprototypen inneholder flere tvetydigheter og uklarheter i brukergrensesnittet som må beskrives eksplisitt gjennom tekst eller et annet medium. Noen av disse har utspring i selve designen eller komponentenes natur, noen er med vilje konstruerte for å utheve dette aspektet ved eksperimentet.

Følgende tvetydigheter og uklarheter er identifisert i skjermbildeprototypen:
- Skal bare bildefiler eller skal alle filer i en mappe vises i lista?
- Hvordan håndterer applikasjonen multiple treff i et søk?
- Kommer det nåværende navnet til en bildefil opp i navnefeltet når denne bildefila blir markert?
- Kan man legge til flere attributter til et bilde?
- Hvordan håndterer applikasjonen et søk med flere argumenter?
- Ved å velge ‘Søk’, hva søker man egentlig etter (navn eller andre attributter)?
- Hvilken vei blir bildet vendt?

Testdeltakerene vil imidlertid ikke få utlevert noe annet enn den rene skjermbildeprototypen, og vil derfor måtte utføre kvalifiserte gjetninger om hvordan interaksjonen foregår.

6.5 Tolkning av interaksjon i et brukergrensesnitt

Hensikten med den første delen er å teste om mer abstrakte representasjoner av et brukergrensesnitt øker forståelsen av den underliggende interaksjonen, samt å la testdeltakerene bli fortrolige med representasjonsformene.

I denne delen av eksperimentet får testdeltakerene etter tur fire artefakter som på forskjellige måter representerer interaksjonen i et brukergrensesnitt. Disse artefaktene blir presentert etter hvor høy dekningsgrad representasjonene har. Testdeltakerenes oppgave blir å diskutere med hverandre og beskrive hvordan de tolker interaksjonen i brukergrensesnittet ut fra disse representasjonene. De blir oppfordret til å se etter uklarheter og tvetydigheter som kan gi opphav til flere tolkninger, og elementer og komponenter som virker avklarende på interaksjonsforståelsen.

1. En utfyllende tekst er ingen garanti mot at det oppstår uklarheter og tvetydigheter i forhold til funksjonaliteten til de enkelte komponentene, eller interaksjonen mellom brukeren og systemet eller internt i brukergrensesnittet.

For alle representasjonene gjelder det at dersom det oppstår situasjoner hvor testdeltagerne er usikre på hvordan de skal tolke interaksjonen, skal de legge vekt på den tolkningen de selv mener er den beste eller mest sannsynlige.

Skjermbildeprototyp

Den første representasjonen som presenteres for testdeltakerene er en skjermbildeprototyp. Her er det viktig at testdeltakerene tar seg god tid, og gjør seg umake med å identifisere de forskjellige elementene og komponentene i brukergrensesnittet.

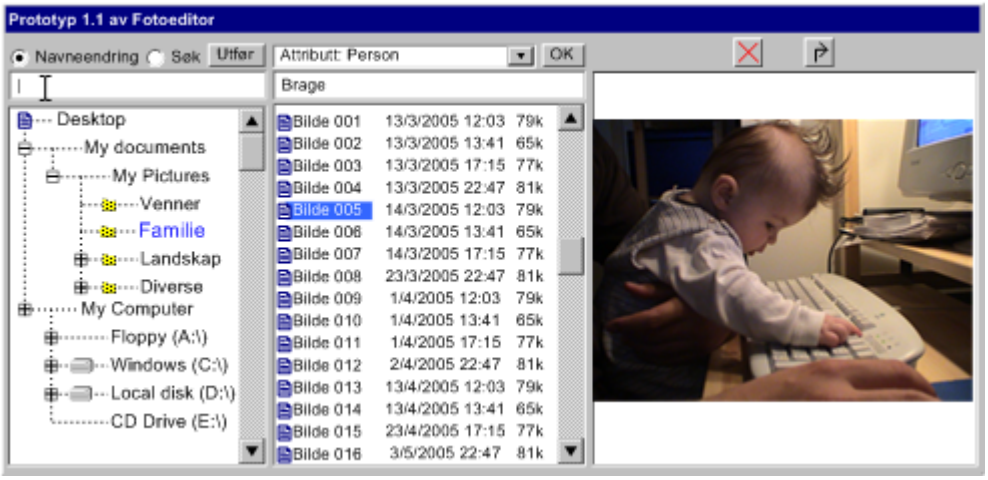


FIGURE 6-3: Utdelt skjermbildeprototyp

Sammen med representasjonen følger denne teksten: “Dette skjermbildet representerer en tidlig prototyp på en applikasjon som har til hensikt å vise bilder, endre filnavn på bilder og legge inn attributter til bilder. Bilder kan også slettes og vendes.”

Testdeltakerene blir her bedt om å diskutere sammen, tolke interaksjonen som skjer i brukergrensesnittet og notere ned hvordan de mener interaksjonen foregår.

Observasjoner

Selve identifiseringen av de beskrevne komponentene skjedde i denne rekkefølgen: Først de to editeringsknappene (slett, vend); deretter ramma hvor det gjeldende bildet vises, filliste og mappetreet; så radioknappene som styrer hvilken oppgave som utføres og nedtrekksboksen som bestemmer hvilket attributt som er valgt. Tilslutt ble de to tekstfeltene som hører til henholdsvis radioknappene og nedtrekksboksen identifisert. Testdeltakerene var usikre på hvilket tekstfelt som hørte til nedtrekksboksen, ellers ble komponentene identifisert raskt.

Testdeltakerene kommenterte følgende tvetydigheter/uklarheter:
- Hvordan velger man aktivt bilde?
- Hvilket tekstfelt brukes for å legge inn attributter?
- Kan man legge til flere attributter?

- Er det meningen man skal kunne søke på enten bildenavn eller bildeattributter, eller kan man søke på begge deler?

Kanonisk abstrakt prototyp

Den neste representasjonen testdeltakerene får utlevert er den kanoniske abstrakte prototypen. Her er det interessant å se om denne representasjonsformen er i stand til å utvide forståelsen av brukergrensesnittet eller tydeliggjøre interaksjonen og avklare noen av tvetydighetene som blir identifisert av testdeltakerene ut fra skjermbildeprototypen.

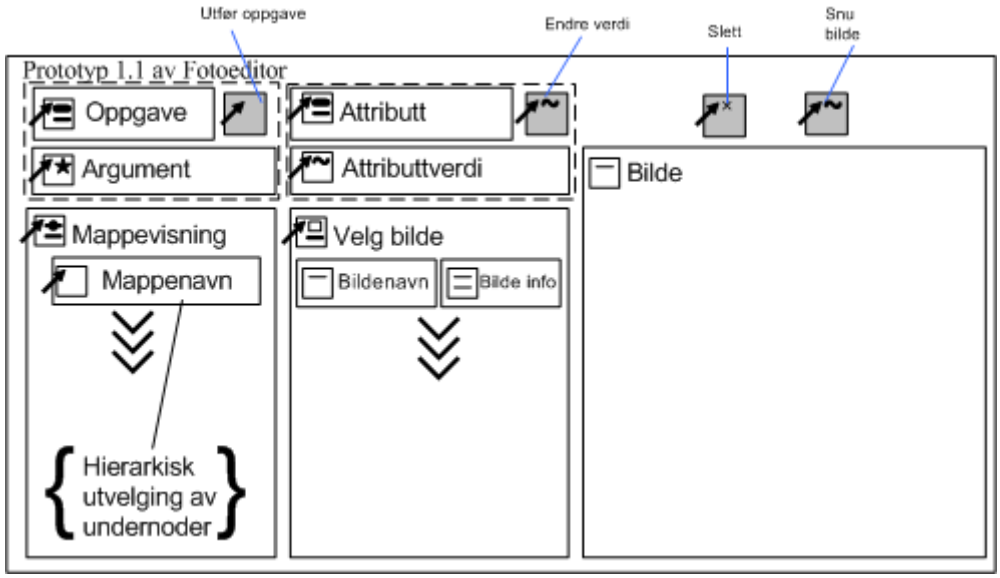


FIGURE 6-4: Utdelt kanonisk abstrakt prototyp

I den kanonisk abstrakte prototypen vil de kanonisk abstrakte komponentene presisere en del momenter om hvordan interaksjonen i brukergrensesnittet foregår: (1) ‘Oppgave’ velges fra et sett av oppgaver, ‘Argument’ gir input til systemet og ‘Utfør oppgave’ trigger en aksjon. Hvilken aksjon som trigges, og hvordan ‘Attributt’ relaterer til ’Oppgave’ gis det derimot ingen informasjon om. (2) ‘Attributt’ velges fra et sett av attributter, ‘Attributtverdi’ angir en komponent som er editerbar og ‘Endre verdi’ fører til en endring når den aktiveres. (Men det sies ingenting om hva denne endringen går ut på.) (3) ‘Slett’ trigger en aksjon som fjerner et element, mens ‘Snu bilde’ trigger en aksjon som endrer et element. (4) ‘Mappevisning’ er en komponent hvor man velger et element fra et sett, og trigger en aksjon. Kommentaren presiserer at dette gjelder en hierarkistisk utvelgelse av undernoder. ‘Mappenavn’ viser at det trigges en aksjon når komponenten aktiveres, men klarer ikke å presisere hva aksjonen faktisk går ut på. (5) ‘Velg bilde’ er en komponent hvor man velger et element fra et sett, og presenterer dette elementet for brukeren. Settet består av et element (‘Bildenavn’) og et sett (Bilde info’). (6) ‘Bilde’ er et statisk element.

Sammen med den abstrakte prototypen følger denne teksten: “Dette diagrammet viser en abstrakt prototyp av skjermbildet vist tidligere.”

Testdeltakerene blir nå bedt om å diskutere og notere ned dersom tolkningen av interaksjonen endres i forhold til skjermbildeprototypen.

Observasjoner

Testdeltakerene kommenterte at den konseptuelle sammenhengen ble tydeligere i dette diagrammet. Spesielt kommentarene i krøllparantesene hjalp på forståelsen. De mente videre at tekstforklaringene til komponentene forklarte interaksjonen bedre enn symbolene, og at selv om oppbygningen av brukergrensesnittet var enklere å se i dette diagrammet, var det vanskeligere å se hva komponentene faktisk ble brukt til.

Videre syntes de komponenten som angir et valg av oppgave ble for generell og at komponenten som angir en valgbar mappe ble for uklar. Komponenten som angir bildeinformasjon var også uklar, men her mente de presiseringen i symbolbruken virket avklarende.

ProtoMODL-diagram

Testdeltakerene får så utlevert en ProtoMODL-prototyp som identifiserer interaktorene i brukergrensesnittet og ett ProtoMODL-diagram som beskriver interaksjonen. Spørsmålet denne gang er om tolkningen av disse diagrammene fører til en økning av forståelsen for interaksjonen ytterligere, både i forhold til det originale skjermbildet og i forhold til den kanoniske abstrakte prototypen?

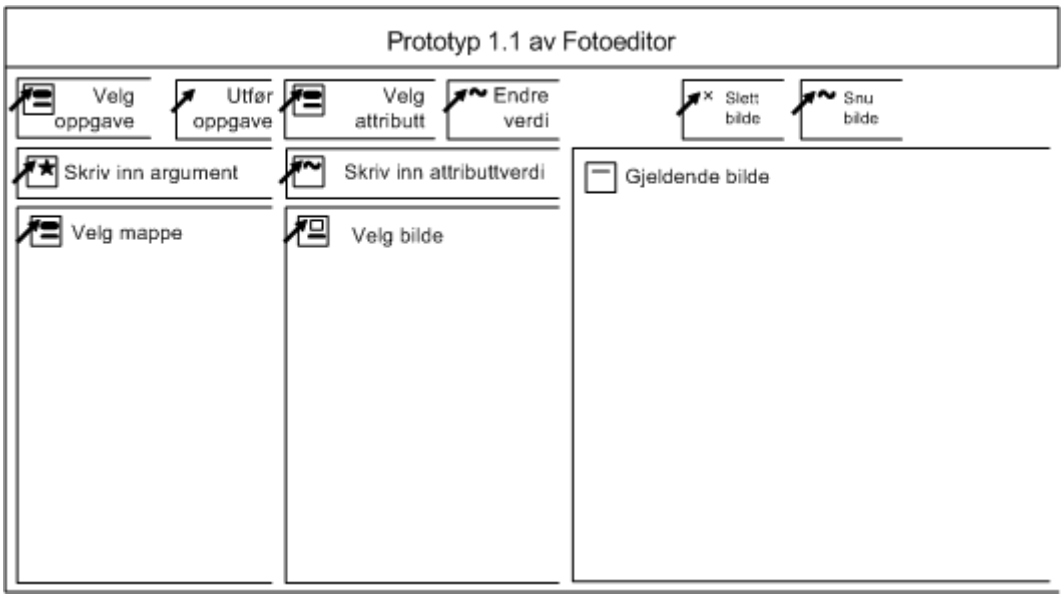


FIGURE 6-5: ProtoMODL-prototyp (interaktorer i brukergrensesnittet)

Det første diagrammet er essensielt det samme som den kanonisk abstrakte prototypen. Den største forskjellen er at alle komponentene nå vises som ProtoMODL-interaktorer, slik at interaktorbeskrivelsen spesifiserer brukerens handlinger på komponentene og ikke innholdet.

I tillegg er ‘Mappevisning’-komponenten abstrahert ned til en enkel interaktor som viser at man velger en mappe fra et sett av mapper og bildeinformasjonen som lå nøstet inn under ‘Velg bilde’ er abstrahert ned til en enkelt ‘selectable action set’-komponent.

Det andre diagrammet viser dataflyten innad i brukergrensesnittet.

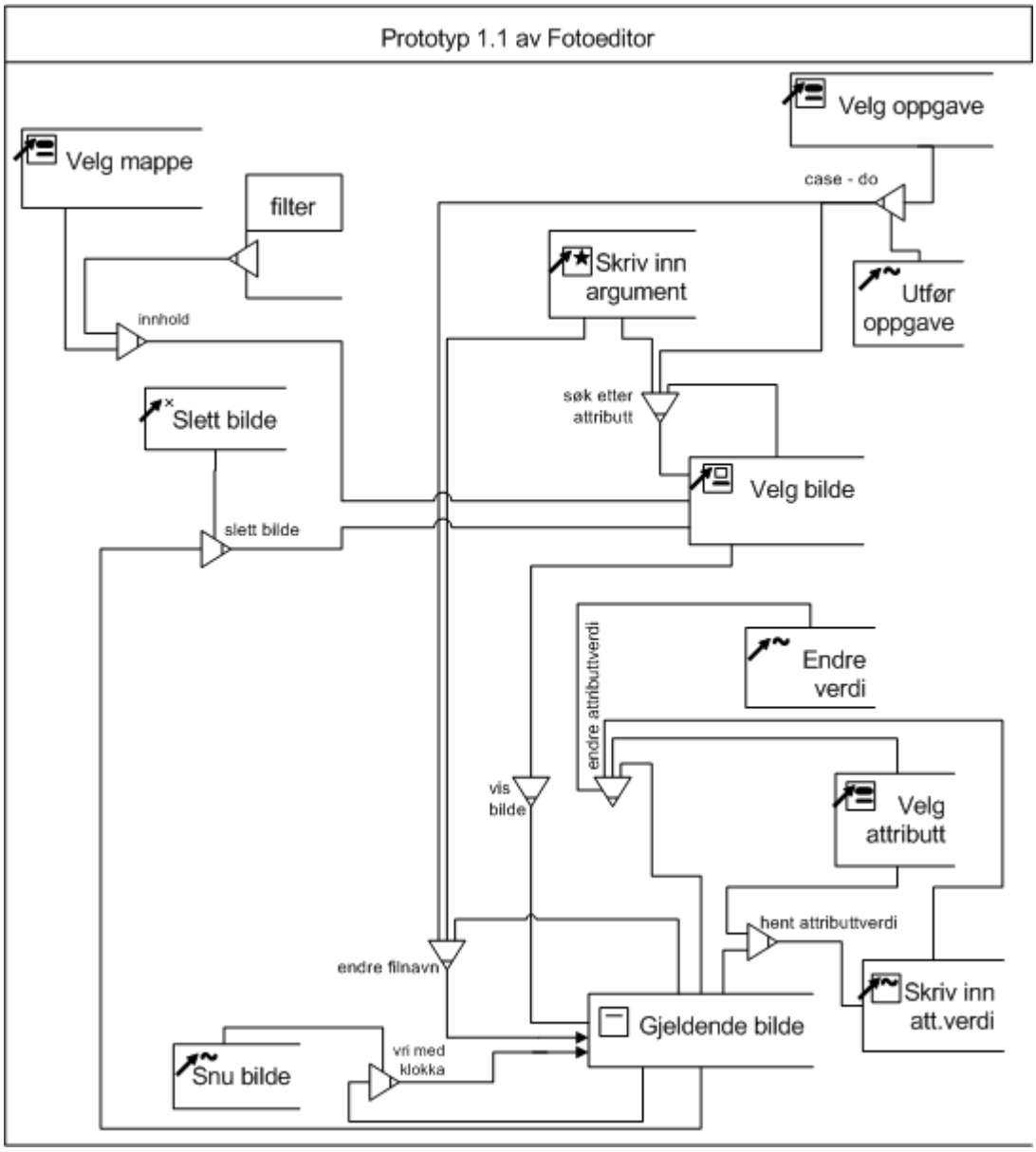


FIGURE 6-6: ProtoMODL-diagram 2 (interaksjon i brukergrensesnittet)

Interaksjonen dette diagrammet beskriver skal tolkes slik:

Når et element fra settet i interaktoren ‘Velg mappe’ blir valgt filtrerer funksjonen *innhold* ut bildefilene i mappa basert på reglene som hentes fra systeminteraktoren ‘Filter’. Bildefilene sendes så videre til interaktoren ‘Velg bilde’. Når et bildeobjekt fra settet i interaktoren ‘Velg bilde’ blir valgt, aktiveres funksjonen i *vis bilde* og bildeobjektet presenteres i interaktoren ‘Gjeldende bilde’.

Når innholdet av en av interaktorene ‘Gjeldende bilde’ eller ‘Velg attributt’ endres, avfyres funksjonen *hent attributtverdi* beregner en verdi som presenteres i interaktoren ‘Skriv inn attributtverdi’.

Når funksjonen *endre attributtverdi* aktiveres av interaktoren ‘Endre verdi’, hentes det aktive elementet i ‘Velg attributt’, tekstobjektet i ‘Skriv inn att. verdi’ og bildeobjektet i ‘Gjeldende bilde’, og bildeobjektets nye attributtverdi beregnes. Denne funksjonen er egentlig en metode og burde derfor, for å vise at den utfører en endring på det aktive bildeobjektet, pekt på interaktoren ‘Gjeldende bilde’. I diagrammet er koblingen som bærer data fra tuppen til interaktoren utelatt for å teste notasjonen og testdeltakerenes forståelse.

Når funksjonen *slett bilde* aktiveres av interaktoren ‘Slett bilde’, hentes bildeobjektet i interaktoren ‘Gjeldende bilde’ og settet som presenteres i ‘Velg bilde’ endres.

Når metoden *vri med klokka* aktiveres av interaktoren ‘Snu bilde’, hentes det aktive bildeobjektet i interaktoren ‘Gjeldende bilde’. Så endres bildeobektet før det tegnes opp på nytt. Innholdet av interaktoren er det samme og derfor er det riktig å benytte en metode i dette tilfellet.

Når brukeren endrer det aktive oppgaveobjektet i interaktoren ‘Velg oppgave’, aktiveres funksjonen *case-do*. Det aktive oppgaveobjektet sendes både til funksjonen *søk etter attributt* og metoden *endre filnavn* og bestemmer hvilken av dem som avfyres. Dersom *søk etter attributt* fyrer, hentes input-verdien i interaktoren ‘Skriv inn argument’ samt settet i ‘Velg bilde’, og endrer det viste elementet i ‘Velg bilde’. Dersom *endre filnavn* fyrer av, hentes input-verdien fra ‘Skriv inn bilde’ samt elementet i ‘Gjeldende bilde’. Resultatet av metoden er en endring på et attributt til bildeobjektet som ligger i ‘Gjeldende bilde’. Sammen med diagrammene skal testdeltagerne få utlevert denne følgeteksten: “I det første diagrammet identifiseres interaktorer på grunnlag av komponentene som finnes i den abstrakte prototypen. ProtoMODL-diagramet er så modellert på grunnlag av disse interaktorene.”

Testdeltakerene blir så bedt om å diskutere seg i mellom hvordan disse diagrammene kan tolkes, og notere ned dersom forståelsen av interaksjonen i brukergrensesnittet endres i forhold til den tolkningen som ble gjort på grunnlag av de to prototypene.

Observasjoner

Testdeltagerne første kommentar var at ProtoMODL-diagrammet som identifiserer interaktorene virker mer oversiktelig enn den kanonisk abstrakte prototypen.

Under tolkningen av interaksjonsdiagrammet tok det en del tid å identifisere interaksjonen, men når det var gjort mente de det var enkelt å se hva som faktisk skjedde i brukergrensesnittet. Testdeltakerne hadde noen vanskeligheter med å se hva som var inn-verdier og hva som var ut-verdier i diagrammet. Det virket også som om det å tolke hvilke data som transporteres i koblingene som går fra enkelte av funksjonene (slett bilde, vri med klokka, case - do), var problematisk. De kommenterte også at det var uklart om det i “Velg bilde” interaktoren ble vist ett og ett bilde, en liste av bilder, eller en liste av bildenavn. Hensikten med systeminteraktoren ‘Filter’ var uklar for dem og måtte forklares av testleder.

Testdeltakerene stilte også spørsmål om hvordan man kan se forskjell på en funksjon som endrer innholdet i en interaktor og en funksjon som endrer visningen av innholdet i en interaktor, og mente dette var en uklarhet i notasjonen som bør gjøres tydeligere.

Testdeltakerene kommenterte ikke feilen som var lagt inn i diagrammet, altså at metoden ‘endre attributtverdi’ ikke peker på objektet den utfører endringen på.

DiaMODL-diagram

Den siste representasjonen som deles ut er DiaMODL-diagrammet som beskriver interaktorene i brukergrensesnittet og dataflyten mellom dem. Denne gangen er det spesielt interessant å observere om dette diagrammet gir en økt eller endret forståelse av interaksjonen i forhold til det ProtoMODL-diagrammene gir.

DiaMODL-diagrammet er en mer formell beskrivelse av interaksjonen enn ProtoMODL-diagrammet er, men en leser skal ideelt oppnå samme forståelse for interaksjonen innad i brukergrensesnittet uansett hvilket diagram som benyttes.

Siden testdeltakerene er bedre kjent med denne notasjonen enn de andre, vil tolkningen av dette diagrammet fungere som en kontroll av eksperimentet. Det virker rimelig å gå ut fra at den forståelsen av interaksjonen som testdeltakerene får gjennom dette diagrammet vil være den mest nøyaktige.

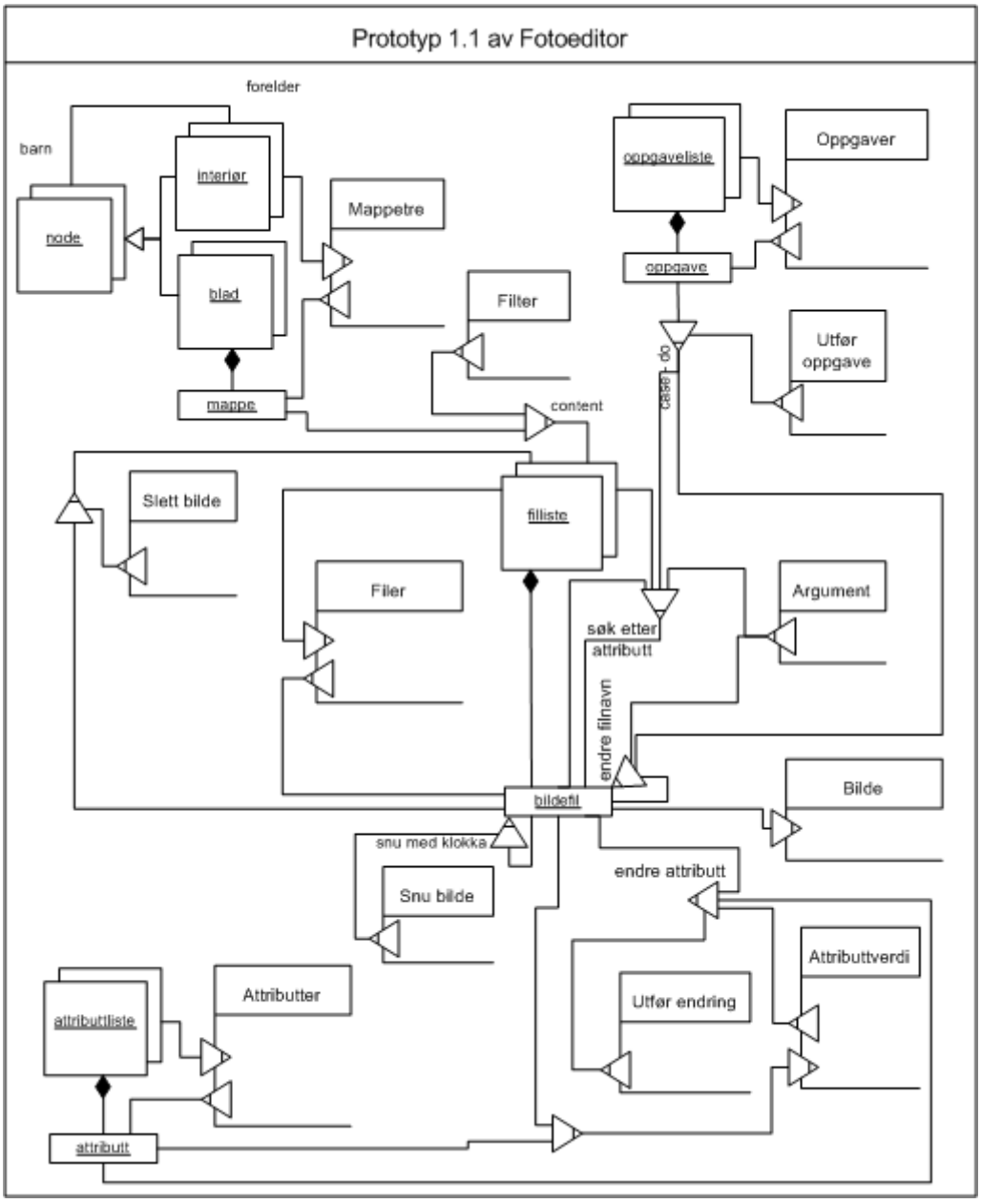


FIGURE 6-7: Utdelt DiaMODL-diagram

Diagrammet er ment å tolkes på følgende sett: I interaktoren ‘Mappetre’ presenteres objektsettet ‘interiør’. Fra dette settet velges et enkeltobjekt (‘mappe’). Innholdet av objektet ‘mappe’ filtreres via interaktoren ‘Filter’ slik at settet ‘filliste’ kun inneholder bildefiler. Dette settet presenteres i interaktoren ‘Filer’, og fra settet velges et enkeltobjekt (‘bildefil’). Dette objektet blir så presentert i interaktoren ‘Bilde’.

Settet ‘attributtliste’ presenteres i interaktoren ‘Attributter’ og fra settet velges et enkeltobjekt (‘attributt’). Fra objektet ‘bildefil’ hentes den verdien som er spesifisert av objektet ‘attributt’, og denne verdien presenteres i interaktoren ‘Attributtverdi’.

Når interaktoren ‘Utfør endring’ aktiveres, henter metoden *endre attributt* objektet ‘attributt’, det aktive innholdet til interaktoren ‘Attributtverdi’ og objektet ‘bildefil’. Ut fra disse verdiene beregnes en endring, men diagrammet spesifiserer ikke nærmere hva denne endringen går ut på. Som i ProtoMODL-diagrammet, burde denne metoden pekt på ‘Gjeldende bilde’ og vist at endringen skjer på et attributt til elementet i denne interaktoren. Også her ble denne koblingen bevisst utelatt for å teste lesbarheten til metodenotasjonen.

Settet ‘oppgaveliste’ presenteres i interaktoren ‘Oppgaver’ og fra settet velges det ut et enkeltobjekt(‘oppgave’). Når interaktoren ‘Utfør oppgave’ aktiverer funksjonen *case-do*, sendes objektett ‘oppgave’ til funksjonen *søk etter attributt* og metoden *endre filnavn*, og bestemmer hvilken av dem som aktiveres. Dersom *søk etter attributt* fyrer av henter den objektet ‘bildefil’, settet ‘filliste’ samt innholdet av interaktoren ‘Argument’, og endrer så det aktive objektet ‘bildefil’. Dersom *endre filnavn* fyrer av, henter den objektet ‘bildefil’ og innholdet av interaktoren ‘Argument’ og endrer en attributt i objektet ‘bildefil’.

Interaktoren ‘Slett bilde’ aktiverer en funksjon som henter objektet ‘bildefil’ og fjerner dette objektet fra settet ‘filliste’. Interaktoren ‘Snu bilde’ aktiverer en metode som henter objektet ‘bildefil’ og utfører en endring på dette objektet.

Sammen med diagrammet får testdeltagerne denne følgeteksten: “Her er DiaMODL-diagrammet, modellert på grunnlag av foregående diagrammer og prototyper.”

Som med det forrige diagrammet, blir testdeltakerene bedt om å diskutere seg i mellom hvordan dette diagrammene kan tolkes, og notere ned dersom forståelsen av interaksjonen i brukergrensesnittet endres i forhold til den tolkningen som har blitt gjort på grunnlag av de foregående representasjonene.

Observasjoner

Testdeltakerenes første kommentar var at DiaMODL ga et litt rotete inntrykk i forhold til ProtoMODL. De brukte relativt lang tid å sette seg inn i interaksjonen som ble beskrevet i diagrammet. Når de først hadde gjort dette virket det som om det var greit for dem å holde orden på dataflyten mellom interaktorene.

Mange av problemene som dukket opp under tolkningen av dette diagrammet, var rettet mot utydeligheter i modellen, ikke notasjonen i seg selv. Spesielt mente de det var vanskelig å tolke case-do funksjonen fordi samme signal går fra denne funksjonen til ‘søk etter attributt’ og ‘endre filnavn’ uten at det spesifiseres hvordan dette skal tolkes. Testdeltakerene mente også at funksjonen ‘endre attributt’ burde pekt inn mot elementet den endrer på. Feilen som ble lagt inn i diagrammet ble altså oppdaget.

Oppsummering

Ut fra skjermbildeprototypen identifiserte testdeltakerne en rekke tvetydigheter og uklarheter. Disse ble delvis oppklart av de andre representasjonene, men noen spørsmål forble uavklarte.

Etter å ha tolket alle representasjonene var spørsmålet om hvordan man velger det aktive bildet fortsatt ubesvart. Ingen av diagrammene klarte altså å vise denne interaksjonen.

Hvilket tekstfelt som skal brukes for å legge inn attributter ble avklart av den kanoniske abstrakte prototypen. ProtoMODL-diagrammet gjorde dette enda tydeligere og DiaMODL-diagrammet bekrefter dette.

Ingen av diagrammene spesifiserer funksjonalitet for å legge til nye attributter til et bilde. DiaMODL-diagrammet presiserer at et attributt kun kan ha en verdi.

ProtoMODL-diagrammet spesifiserer at det kun er mulig å søke etter attributter. Dette bekreftes i DiaMODL-diagrammet. At bildefilens navn kan være et attributt til bildeobjektet er imidlertid ikke spesifisert noe sted.

6.6 Utvidelse av modellen

I den andre delen av eksperimentet utleveres en ny skjermbildeprototyp av den samme applikasjon til testdeltakerene. Denne prototypen representerer en ny designiterasjon, hvor layouten er noe endret. I tillegg er funksjonaliteten i prototypen utvidet slik at det er mulig å plukke ut et utvalg bilder som kan editeres, for eksempel ved hjelp av å holde ned shift eller ctrl og samtidig klikke på bildefilene med musa. Det er også lagt til noen nye elementer i brukergrensesnittet for å støtte den utvidede funksjonaliteten.

Når et utvalg bilder er markert, vil en aksjon bli utført på alle bildene i utvalget, med unntak av aksjonene “slett bilde”, “vri bilde” og “Velg bort”. Disse aksjonene vil bare bli utført på det bildet som vises i bilderamma. Ved hjelp av trykknapper er det mulig å navigere *innenfor utvalget av bilder.*

Det er nå også mulig å vri bildet 90 grader både mot høyre og venstre. De fire trykknappene markert med liggende vinkler endrer hvilket bilde som vises i bilderamma til henholdsvis første, forrige, neste og siste bilde *i utvalget*. En annen trykknapp gjør det mulig å fjerne markeringen av filnavnet til det bildet som vises i bilderamma fra fillista.

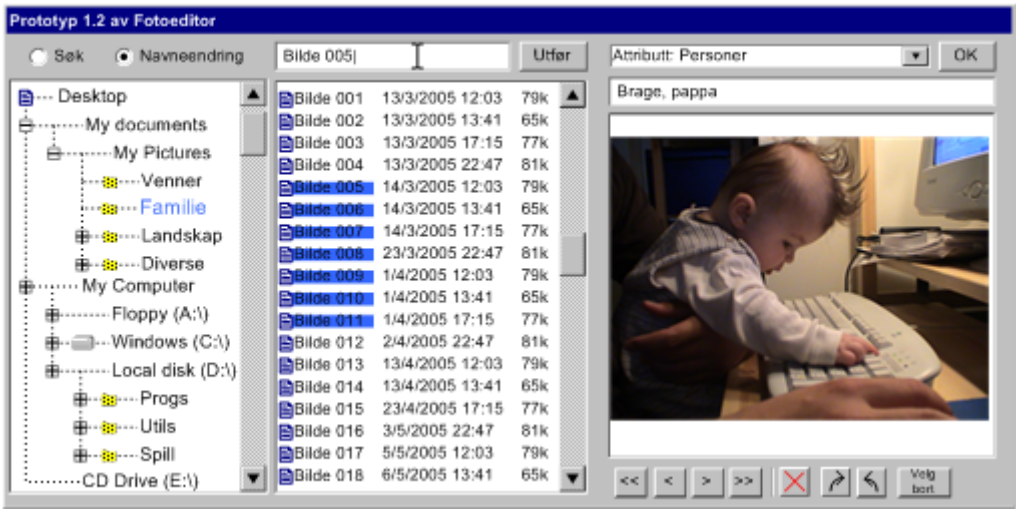


FIGURE 6-8: Utvidet skjermbildeprototyp av applikasjonen “Fotoeditor”

Sammen med skjermbildeprototypen får testdeltakerene denne følgeteksten: “Oppgaven her er å modellere den utvidede interaksjonen på bakgrunn av dette skjermbildet. Dere står fritt i å bruke den notasjon dere selv vil (også andre notasjoner enn CAP, ProtoMODL og DiaMODL dersom dere mener det er lettere eller mer hensiktsmessig).”

Hensikten denne gang er å se både hvilken representasjon testdeltakerene foretrekker og hvilken representasjon som best egner seg til å uttrykke den endringen i interaksjonen som oppstår når bruksområdet til et brukergrensesnitt endres.

Observasjoner

Testdeltakerenes førsteinntrykk var at både tekstfeltet over fillista, og nedtrekksboksen og tekstfeltet over bilderamma viser tilknytning mye tydeligere enn den forrige skjermbildeprototypen gjorde.

Det tok en del tid å tolke hva de nye elementene er ment å gjøre. De spekulerte over hva trykknappen ‘Velg bort’ egentlig gjør. Etter en stund valgte de helt korrekt å tolke at denne komponenten fjerner markeringen av filnavnet til det bildet som vises i bilderamma fra fillista.

Trykknappene som vrir bildet, bestemte de raskt at kunne modelleres ved hjelp av en enkelt interaktor, slik som i første del.

Navigasjonsknappene var vanskeligere å tolke, spesielt de knappene markert med dobbelte liggende vinkler; ville de bla gjennom alle bildene eller bare vise hhv det første eller siste bildet? De valgte helt korrekt å tolke interaksjonen til disse knappene slik at de viste henholdsvis det første og det siste bildet i utvalget.

Når det gjaldt selve utvalget var det vanskelig å tolke hvilket bilde som ville vises i bilderamma.

Modellering i ProtoMODL

Testdeltakerene bestemte med en gang at de ville modellere interaksjonen i ProtoMODL. Det første problemet de møtte på var hvilke kanonisk abstrakte komponenter som best ville representere interaksjonen de ønsket å modellere. Forklaringene og eksemplene over disse komponentene, tatt fra [Constantine et. al., 2003], var ikke presise nok til at de enkelt kunne plukke ut de komponentene de mente passet best til den aktuelle oppgaven. De bemerket også at selve symbolikken som brukes i komponentene ikke var særlig intuitive.

Når de skulle modellere hvordan man kan plukke ut et utvalg av bilder, spekulerte testdeltakerene en del på hvordan det er mulig å modellere et sett inne i en interaktor i ProtoMODL. I DiaMODL kunne de brukt sett-notasjonen til dette. De valgte å løse dette som en nøstet interaktor, der subsettet (bildeutvalget) ble tolket som en egen interaktor. Også trykknappen “Velg bort” var vanskelig å modellere. Spesielt hadde testdeltakerene problemer med å finne en kanonisk abstrakt komponent som kunne brukes til å formidle den ønskede interaksjonen.

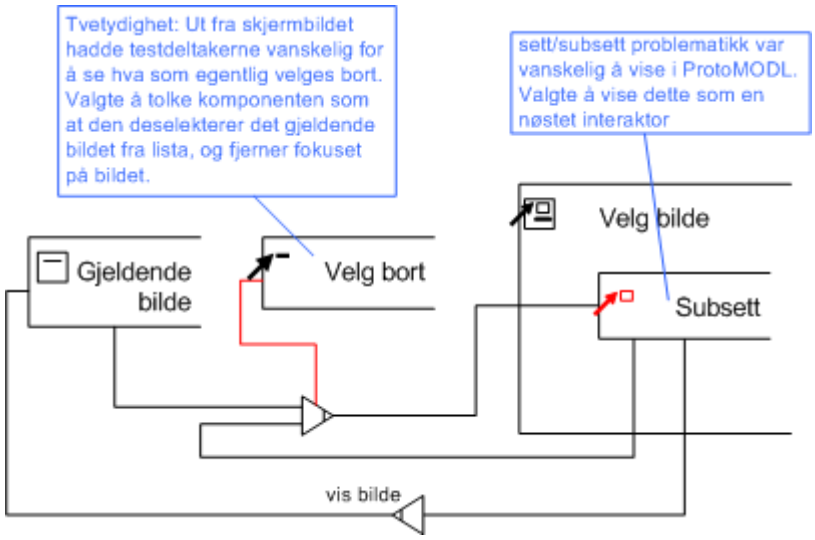


FIGURE 6-9: Testdeltagernes ProtoMODL-diagram 1: Modellering av filutvalg og deselektering

I denne modellen er det kun to notasjonfeil. Det ene er at koblingen fra "Velg bort" knappen går fra venstre side (og dermed markerer en inn-verdi), det andre er at den kanonisk abstrakte komponentent som de brukte på seleksjonsinteraktoren indikerer en aksjon (view), og ikke et aktivt materiale. Her hadde det antakelig vært mer riktig å bruke "selectable collection"på den ytterste interaktoren og kalt den "Velg bildeutvalg", og brukt "selectable view set" på den innerste interaktoren.

Testdeltakerene valgte å lage et nytt diagram for å modellere navigasjonsknappene. De hadde en del problemer med å finne kanonisk abstrakte komponenter som klarte å formidle interaksjonen på rett måte; ville interaksjonen best representeres som en action tool, en selecttool, en move (up/down) tool eller en view tool?

Til slutt ba de testleder si hvilken som ville være best å bruke. Siden de tydeligvis holdt på å kjøre seg fast, valgte testleder da å si at i eksempelet var interaksjonen til disse knappene smeltet sammen i en interaktor symbolisert som et ‘selectable action set’, men at flere av forslagene deres antakelig ville være like bra egnet til å modellere interaksjonen i ProtoMODL.

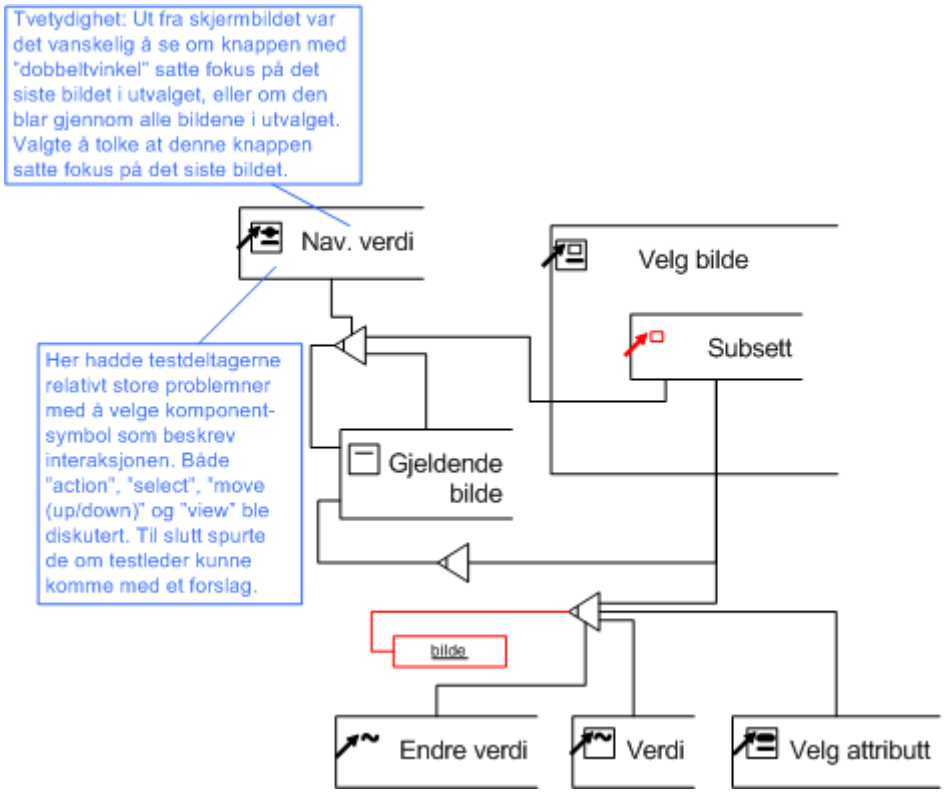


FIGURE 6-10: Testdeltagernes ProtoMODL-diagram 2: Modellering av navigering innenfor filutvalget

Mye av problematikken fra forrige modell gikk igjen her: Hvordan skal man tolke interaksjonen som navigasjonsknappene representerer og hvilken kanonisk abstrakt komponent viser denne interaksjonen best. En av testdeltakerene uttrykte frustrasjonen slik: "Skulle vært et ikke-view symbol".

Et annet problem gikk på hvordan man skulle indikere at en funksjon førte til endring av en attributtverdi. Her valgte testdeltakerene å vise dette ved å la funksjonen peke på et objekt som representerte bildet. Det riktige her ville vært å la en metode peke inn mot “subsett”. Når det er sagt, skal det anmerkes at diagrammene de gikk ut fra ikke var helt presis på dette området heller, ettersom det bevisst var lagt inn feil på akkurat dette området.

To problemer med notasjonen ble påpekt flere ganger av testdeltakerene: At koblingene er forvirrende, slik at det er vanskelig å se dataflytretningen og at de kanonisk abstrakte komponentene ikke er identifisert godt nok, slik at det er van-

skelig å plukke ut hvilke komponenter som passer best til å beskrive interaksjonen til en brukergrensesnittkomponent.

Modellering i DiaMODL

Når testdeltakerene skulle modellere den samme utvidelsen ved hjelp av DiaMODL var det færre problemer som dukket opp. Testdeltakerene gikk ut fra ProtoMODL-diagrammene og identifiserte raskt interaksjonen derfra. Både at notasjonen var velkjent fra før og at problemområdene var identifisert, hadde nok innvirkning her.

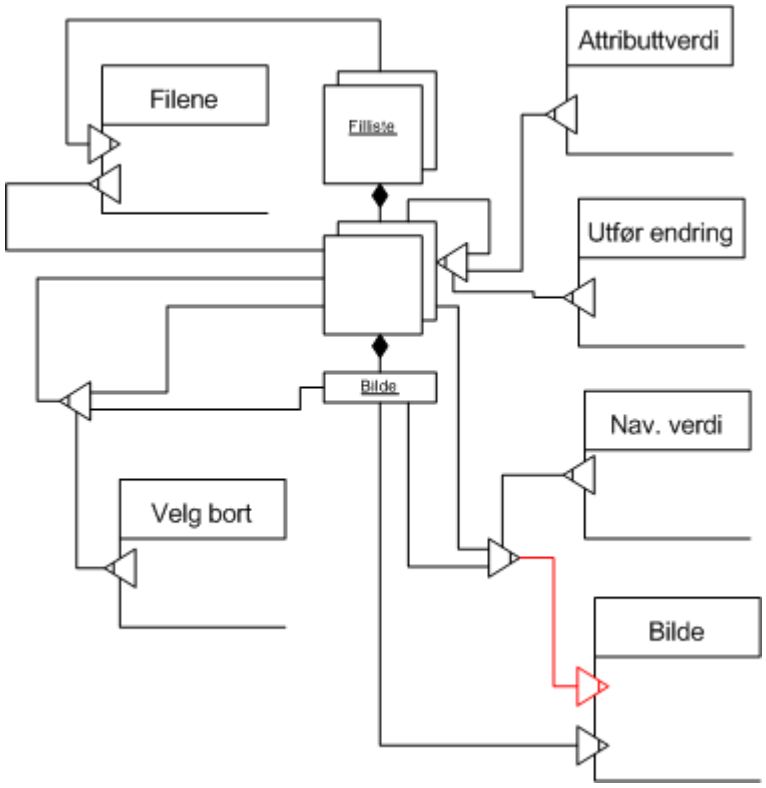


FIGURE 6-11: Testdeltagernes DiaMODL-diagram: Modellering av navigering innenfor filutvalget

I modellen er det bare en feil, nemlig at navigasjonen vil endre hvilket objekt som er valgt ut fra subsettet. Funksjonen bør derfor peke mot objektet, ikke interaktoren.

6.7 Debrifing av testdeltakerene

Etter gjennomføringen av selve eksperimentet er det lagt opp til en debrifing der testleder stiller testdeltakerene en rekke spørsmål som vil støtte oppunder de observasjoner som blir gjort. Altså avklare en del forhold rundt det å tolke, modellere og prototype brukergrensesnitt som ikke har kommet klart nok fram under gjennomføringen av eksperimentet.

Generelle spørsmål rundt eksperimentet

Testdeltakerene blir først spurt om de har noen kommentarer rundt utførelsen eller oppbyggingen av eksperimentet. Her blir det gitt en mulighet til å komme med generell kritikk til eksperimentet, noe som så kan brukes både til å identifisere mulige feilkilder i selve eksperimentet og bekrefte om oppbyggingen av eksperimentet er vellykket.

Kommentarer fra testdeltakerene

Kommentarene var jevnt over positive. Testdeltakerene nevnte blant annet at eksperimentet var "veldig bra lagt opp" og "profesjonelt gjennomført".

Det ble også nevnt at siden det samme caset ble brukt gjennom hele eksperimentet, var det "veldig lett å se hvor ting var oppklarende", og at det var artig å se hvordan de antakelsene som ble gjort ut fra skjermbildeprototypen, ble bekreftet eller avkreftet gjennom de abstrakte modellene. De mente også at dette gjorde at de forstod mer av interaksjonen i de abstrakte representasjonene.

De ble så spurt om de "mente denne måten å jobbe på kan ha noe for seg når et brukergrensesnitt skal designes". Dette for å se om testdeltakerene mente vekslingen mellom å benytte prototypingsspråk og modelleringsspråk gir en ekstragevinst, kontra det å bare benytte ett av språkene, eller benytte de hver for seg. På dette spørsmålet svarte testdeltakerene følgende:

Transkripsjon fra debrifing

A: "Ja… Jeg har vært litt ambivalent jeg egentlig. Jeg ser fordelen i forhold til det å kunne kommunisere funksjonalitet til kanskje en bruker, - i stedet for bare skjermbilder, men jeg hadde inntrykk av at DiaMODL var tenkt å brukes som en slags magisk måte å generere brukergrensesnittet på. Problemet der er at det er ekstremt vanskelig å tenke sånn før du lager den (peker på skjermbildeprototypen). Selv på eksamen fikk vi et grensesnitt som så skulle modelleres."
B: "Hvis man kan språket, og får både skjermbilde og den modellen samtidig, er det veldig greit. Men hvis man bare får modellen tar det litt tid å gå igjennom og forstå hva som virkelig skjer."
A: "Ja, i samarbeid med et slikt skjermbilde vil det kanskje fungere veldig bra."

Det var altså tanken på å bruke modeller sammen med skjermbilder som tiltalte testdeltakerene mest. Som designverktøy syntes de at den mest aktuelle måten å benytte modellene på var som et kommunikasjonsverktøy til andre interessenter i et utviklingsprosjekt.

Generelle spørsmål rundt bruk av prototyper og modeller

Det er viktig å vite om testdeltakerene er vant med å benytte seg av prototyping, modellering og modelleringsverktøy, siden dette kan ha innvirkning på hvordan de angrep problemene underveis.

Kommentarer

Da testdeltakerene ble spurt om "prototyping var noe de pleide å benytte som et hjelpemiddel når de designet brukergrensesnitt", svarte testdeltaker A slik: "Nå har ikke vi gjort så mange grensesnitt selv. Oppgaven har stort sett vært å lage en prototyp. Men når jeg lager ting selv, programmerer et spill eller noe sånt, er det rett

på. Det er ikke sånn at man lager en prototyp, det er mer sånn at man bygger på. Poenget er vel at vi ikke har laget så veldig store systemer."

I mindre prosjekter anså de det heller ikke som så veldig naturlig å lage skisser av brukergrensesnittet.

På spørsmål om "i hvilken grad de benyttet prototyping som verktøy for å utvikle og designe funksjonalitet", svarte testdeltaker A: "Det hender seg, det gjør det. Kanskje mer for å teste om det virkelig funker det jeg har tenkt på, så kan det hende at jeg lager en testklasse eller noe"

Når de ble spurt om de "pleide å benytte seg av modellering under utviklingen av helhetlige systemer", svarte testdeltaker A: "Ja, det har vi faktisk blitt ganske flinke på. Det vi har jobbet mest med de siste årene er interaksjon, det å analysere brukerne, å kunne modellere hva de gjør, oppgavene deres blant annet"

Testdeltakerene fortalte også at særlig klasse-, ER- og dataflytdiagrammer ble benyttet for å modellere systemer. Andre representasjoner de brukte for å beskrive funksjonalitet og oppførsel var blant annet use case, tekstbeskrivelser og uformelle figurer.

Generelle spørsmål rundt modellering og design av brukergrensesnittet

Videre er det ønskelig å få mer spesifik informasjon om testdeltakerene også benytter modellering når de designer brukergrensesnitt. Dette for å få kunnskap om testdeltakernme er vant med å tenke på brukergrensesnitt som noe abstrakt, siden dette kan ha innvirkning på gjennomføringen av eksperimentet.

På spørsmål om de "syntes modellering var like aktuelt ved design av brukergrensesnittet", svarte testdeltaker B: "Ikke en helt formell... (peker på dialogmodellene) men mer bokser og piler, kanskje." Testdeltaker A supplerte med: "For et par måneder siden satt vi og drøftet litt, og da var det post-it lapper som da representerte forskjellige funksjonaliteter i grensesnittet. Ellers blir det mer uformelle tegninger."

Generelle spørsmål rundt de forskjellige representasjonene brukt i eksperimentet

Det er ønskelig å undersøke hvilket overordnet inntrykk testdeltakerene sitter igjen med etter å ha utført tolknings- og modelleringsoppgavene. Den subjektive følelslsen man har etter å ha benyttet en representasjon kan være viktig å identifisere, ettersom den kan si noe om akseptansefaktoren til de forskjellige representasjonene.

Transkripsjon fra debrifing

Testleder: "Hvilke representasjoner synes dere ga mest informasjon om brukergrensesnittet?"
B: "Litt vanskelig å si siden vi har hatt samme caset, så vi er på en måte litt farget av hva vi vet der da."
A: "Jeg synes de gir forskjellig informasjon... altså jeg synes de fungerer veldig godt sammen".

B: “Kanskje ikke alle fem,men at det holder med... Jeg tror jeg ville hatt en sånn som viste det (peker på skjermbildeprototypen), også kanskje - hvis du ser på ProtoMODL, så hadde den, hvor ble det av den da? Den viser interaksjon. Så hvis du tar de to sammen.”
A: “Det jeg syntes var veldig fint med ProtoMODL, er at den gjør det veldig mye mer ryddig... Den er raskere å lese enn for eksempel DiaMODL.”
B:”Ja, den blir litt mer oversiktelig kanskje.”
A: “DiaMODL har fryktelig mange informasjonsobjekter, selv om jeg er veldig glad i de, så... er det, når man skal lese en modell ... er jeg er nok enig med B at som en informasjon til deg så er nok den (peker på ProtoMODL) bedre”.
B: “Det spørs hvem som er mottaker av det, hvis det er en som kanskje skal lage systemet..”
A: “Den er jo mer formell, DiaMODL”.
B: “Men hvis det er en bruker, er det kanskje bedre med den (peker på ProtoMODL)”
A: “Tror aldri jeg ville vist den til en bruker, ja jeg vet ikke jeg”
B: “Det spørs hvem brukeren er det?.”
A: “Ja, ikke sant. Er det en utvikler...”
Testleder: “Hva om det er en designer?”
B: “Ja heller den (ProtoMODL) enn den (DiaMODL) i så fall.”
A: “ Vanskelig å sette seg inn i ikke sant. Uansett, for folk som er utenforstående så blir sånt vanskelig, det blir det, men den (ProtoMODL) vil uansett være enklere enn den (DiaMODL). Hvis du ikke kan noenting om det vil kanskje et skjermbilde med en forklaring under være ekstra lett... For en som er mer involvert vil det her absolutt være mer informativt... Men jeg må si jeg syntes det var vanskelig å modellere den (ProtoMODL)”
B: “Syntes du?”
A: “Ja jeg syntes det.”
B: “Jeg synes egentlig den var enklere”
A: “Ja jeg skjønte det, jeg slet med den, men det var nok fordi jeg var mer inne i den (DiaMODL).”

Testleder: “Hvilken representasjon synes dere var mest unøyaktig, av alle de fire. Hvilken synes dere ga minst informasjon i forhold til de andre?”
A: “Jeg tror kanskje den kanoniske jeg altså, hva synes du?”
B: “Den (DiaMODL) er den nøyaktigste, men den her (skjermbildeprototyp) er kanskje den som er lettest å tolke.”
A: “Det kommer an på hva man er vant til å se, derfor er man ikke helt objektiv her.”
B: “Men samtidig, Hvis man kan noe om kanonisk prototyping, er den her like grei som den (peker på skjermbildeprototypen) i så fall”
A: “Når jeg ser det, tenker jeg at den ga meg litt lite... Det er eneste det at det stod forklart litt nærmere altså.”
B: “Det tok lenger tid å komme inn i den her (kanonisk abstrakt prototyp), men

egentlig så synes jeg nesten den her ga meg mer en den (skjermbildeprototyp) her i så fall... Den er mer nøyaktig.“
A: “Den er mer nøyaktig, det er den.”

Testleder: “Et veldig subjektivt spørsmål: hvilken representasjon likte dere best, av alle fire?”
A: “DiaMODL tror jeg. Jeg er mest sånn at jeg liker best det jeg kan.”
B: “Hvis jeg kan velge, får jeg både den og den da? (peker på de to tilhørende ProtoMODL-diagrammene) ... Da tror jeg velger den jeg, for da får jeg både litt om hvordan komponenten ser ut og interaksjon.”
A: “Ja det får du faktisk, at du har plassering av komponentene i grensesnittet”
B: “Ja ikke bare plassering, men at du med et blikk ser at komponentene på en måte er valgt ut.”
A: “Kanskje jeg skal være enig med deg, (testdeltaker B)”

Testleder: “Men begge likte de abstrakte modellene bedre enn prototypene?”
B: “Ja, jeg likte forsåvidt... Ville helst hatt begge da, både den (ProtoMODL) og den (skjermbildeprototypen), men måtte jeg velge en ville jeg helst hatt den abstrakte...”
A: “Jeg synes det er så vanskelig å velge mellom, det er så forskjellige måter å tenke på.”

Spørsmål rundt tolkning av den kanonisk abstrakte prototypen

For å få vite testdeltakerenes subjektive oppfatning om bruk av kanonisk abstrakte prototyper i praktisk bruk, stilles testdeltakerene spørsmål om hvordan de mener den abstrakte prototypen som ble brukt i eksperimentet fungerte som tolkningshjelpemiddel.

Kommentarer

Testdeltaker A svarer da at: “Vi fikk jo tolket at de hang logisk sammen, i tillegg til de oppgavene de forskjellige knappene utførte. Det var vel egentlig det som var mest oppklarende.”

Som respons på dette ble de så spurt “om de mente den kanonisk abstrakte prototypen og symbolene brukt i denne notasjonen ikke ga en dypere forståelse av interaksjonen?”

Transkripsjon fra debrifing

A: “For det første synes jeg de er vanskelige å forstå i forklaringen, for det andre så husker jeg ikke hva de er. Noen av dem , som ‘slett’, er greie å forstå”
B: “De gir en grei oversikt over komponentene, men jeg synes ikke de viser interaksjon like bra som DiaMODL eller den her (ProtoMODL)”
A: “Noen av disse komponentene... er enkle å forstå... men det er mange av dem som jeg sliter med å huske, og som jeg aldri ville klare å huske.”
B: “Men hvis man har symbolene ved siden av seg med en god forklaring, så ville det ikke være noe problem, akkurat?”
A: “Men det ville være litt slitsomt å drasse rundt på.”
B: Det er lettere å slå opp i ei liste med symboler å se hva de betyr, enn å prøve å

huske hva alle de forskjellige komponentene består av og hvordan de interagerer sammen, på en måte”

Spørsmål rundt tolkning av dialogmodellene

For å få vite noe om hvordan testdeltakerene opplever modellen i praktisk bruk, spørres testdeltakerene om hvordan de synes DiaMODL fungerer som tolkning-shjelpemiddel.

Transkripsjon fra debrifing

A: “Altså, den sier jo ingenting om utseendet, det gjør den ikke. Den gir et veldig sånn rotete inntrykk i første omgang, det tar litt tid å komme inn i den. men jeg synes den er fin på å følge med på objektene... altså informasjonsflyten synes jeg er lett å få med seg.”
B: “Hvis man først har satt seg inn i modellen, får man kanskje en større forståelse av hva systemet gjør enn i forhold til for eksempel en prototyp. Du sitter ikke å lurer på ting; hvis du først har lært deg hva alt betyr, vet du hva systemet gjør. “

På samme spørsmål om ProtoMODL svarte de følgende:

Transkripsjon fra debrifing

A: “Den gir jo også til en viss grad flyt da. Men jeg synes det var litt vanskelig å følge med på hva som faktisk flyter.”
B: “Jeg synes den var raskere å komme inn i den enn den”
A: “Der er jeg uenig.”
Testleder: “Hva om man hadde funnet en metode for å rette informasjonen som flyter, ville den da vært lettere å lese?”
A: “Ja”
B: “Jeg synes egentlig det var greit sånn som det var, at det var inn og det var ut.”
A: “Altså tolkningsmessig, det var det du var ute etter? Jeg er fortsatt ikke helt familiær med disse symbolene da, så de gir meg sånn i utgangspunktet ikke noe mer informasjon enn det som faktisk står inne i rutene... Jeg kjenner det selvfølgelig igjen etter hvert da, jeg ser at de har et slags logisk fellesskap da, og det kan man se ganske fort.”
B: “Jeg vet ikke helt. Jeg likte det egentlig ganske godt...”
Testleder: “Så det kommer an på hvor godt man kjenner symbolene?”
B: “Ja”.
A: “Det har mye å si. For meg akkurat nå så funker det dårlig.

For å se om testdeltakerene mente en mer abstrakt representasjon i form av en dialogmodell gir et mer entydig bilde av interaksjonen enn en skjermbildeprototyp alene klarer å formidle, ble de stilt spørsmål “om de hadde funnet eksempler på at utvetydigheter eller uklarheter ble avklart ved hjelp av noen av disse dia-grammene”. På dette svarte de slik:

Transkripsjon fra debrifing

B: “Så som for eksempel hva som skjedde med de her (peker på navigasjonsknap-pene i den utvidede skjermbildeprototypen).”
A: “Nei, vi fikk ikke vite noe der, for de måtte vi lage selv.”
B: “Men om vi hadde fått utlevert en modell, med de knappene modellert, så ville vi

ha forstått det"
A: "Ja."
Testleder: "Kan man si at modelleringen tvang dere til å tolke knappene?"
A: "Ja"
B: "M-m"

Det var videre interessant å se om testdeltakerene hadde funnet noen umiddelbare semantiske forskjeller på de to notasjonene, for eksempel om de ga dem forskjellig forståelse av interaksjonen, eller om det ene verktøyet var enklere å bruke, mer nøyaktig, osv, enn det andre. De ble derfor spurt "hvilke forskjeller de fant i bruken av ProtoMODL i forhold til bruken av DiaMODL".

Transkripsjon fra debrifing

A: "Oversikt. Det er vel en umiddelbar..."
B: "Ja."
Testleder: "At ProtoMODL gir bedre oversikt?"
A: "Ja."
B: "Ja."
A: "Jeg kommer tilbake til disse informasjonobjektene jeg, som jeg liker litt. Jeg føler at det mangler litt klar informasjonsflyt av og til."
B: "Ja, men samtidig er det litt deilig å slippe de ôg."

Spørsmål om bruksterskelen til ProtoMODL

For å finne ut noe om ProtoMODL kan gjøre det lettere å benytte interaksjonsmodellering med penn og papir, ble testdeltakerene også spurt om hvordan de syntes bruksterskelen var for å sette seg inn i notasjonen.

Transkripsjon fra debrifing

B: "Den var vel grei. Lettere enn DiaMODL
A: "Da sier jeg motsatt. Jeg vil si den ligger omtrent på nivå med DiaMODL, jeg. Den er lett å lære siden vi allerede kunne DiaMODL."

På spørsmål om hvor vanskelig det var å tolke brukerinteraksjonen i skjermbildeprototypen ut fra ProtoMODL-diagrammet, svarte de på denne måten:

Transkripsjon fra debrifing

A: "Vi ser jo, ikke sant... vi får jo beskjed om hva vi skal gjøre. Vi får beskjed om hva faktisk oppgaven er. 'Skriv inn'."
B: "Ja, den er grei den"
A: "Det er ingen tvil om at du kan gjøre det."

De kommenterte også at det å vise at en interaktor inneholder et subsett av et sett var vanskelig å vise i ProtoMODL.

Spørsmål rundt arbeidet med utvidelsen av prototypen

På spørsmål om hvordan de hadde opplevd størrelsen på utvidelsen av brukergrensesnittet i skjermbildeprototypen i forhold til størrelsen av den utvidete interaksjonen, svarte de:

Transkripsjon fra debrifing

A: “Første inntrykket vil jeg si at dette var ikke så mye, det var første tanken... Noen nye knapper.”
B: “Det er jo det at hoveddelene er ikke forandret, så da synes man ikke det er noen stor forandring, men det er jo...”
A: “Rent logisk er det jo en del.”

Da de så ble spurt om hvilken av notasjonene de mente ga mest utbytte når de skulle modellere denne utvidelsen, beholdt testdeltakerene samme rangering som når de skulle bedømme hvilken notasjon de foretrakk når de tolket interaksjonen. Testdeltaker A foretrakk DiaMODL, mens testdeltaker B foretrakk ProtoMODL. Begge var enige om at den kanonisk abstrakte prototypen ga minst utbytte i denne sammenhengen.

Generelle spørsmål om nytteverdien av prototyping og modellering av interaksjon

Til slutt ble testdeltakerene spurt noen spørsmål om de ville vurdert å benytte noen av verktøyene kanonisk abstrakt prototyping, ProtoMODL og DiaMODL dersom de skulle designe eller dokumentere et brukergrensesnitt. Spesielt interessant var det å se om svarene de ga her var konsistente med de svarene de ga i starten av spørsmålsrunden.

Transkripsjon fra debrifing

Testleder: “Ville dere vurdert å bruke abstrakt prototyping, dersom dere skulle designe eller dokumentere et brukergrensesnitt?”
A: “I hvert fall en abstrakt en ville vært aktuell. Men akkurat om jeg ville brukt denne typen, det er jeg ikke sikker på.”
B: “En abstrakt prototyp til design, det hadde antakelig vært aktuelt. Men for dokumentasjon tror jeg jeg heller i stedet for den der ville hatt skjembilde og tekst der eller noe sånt.
A: “Ja.”

Testleder: “Samme med DiaMODL, ved design eller dokumentasjon av et brukergrensesnitt, ville dere vurdert å bruke det?”
A: “Ja, jeg tror faktisk jeg ville ha gjort det. Og du har jo gjort det alt?”
B: “M-m”
A: “Jeg har jo forsåvidt... Jeg hadde planer om å gjøre det, på prosjektet mitt før jul, men det ble ikke tid. Jeg fant vel egentlig ut gjennom oppgavemodeller at det ikke var noe vits å gjøre det, det ble irrelevant.
B: “Vi brukte det til å dokumentere, det var egentlig det letteste. Vi brukte TaskMODL først, og så gjorde vi design. Og så brukte vi DiaMODL til å dokumentere, eller for å bekrefte det vi hadde laget.”

Testleder: “Hva med ProtoMODL?”
A: “Mest til dokumentasjon tror jeg, der også. Det er fryktelig vanskelig å gå designveien altså.”
B: “Men jeg tror jeg heller... ja jeg ville brukt den mer til dokumentasjon enn til des-

gin, men at det ville vært lettere å brukt den til design enn DiaMODL i alle fall.
A: “Ja.”

6.8 Resultat

Eksperimentet gir grunnlag for å trekke en del slutninger rundt de fire representasjonene generelt, og notasjonen til ProtoMODL spesielt.

Om datagrunnlaget

Datagrunnlaget som resultatene baserer seg på er hentet fra ett enkelt eksperiment med to deltakere. Eksperimentet produserte nok materiale til å slå fast at datagrunnlaget er solid og testdeltakernes kompetanse var høyt nok til kunne si at eksperimentets målsetninger ble oppfylt. Ettersom antall deltakere er såpass få, er imidlertid ikke det empiriske grunnlaget godt nok til å bygge bastante konklusjoner på. Datagrunnlaget må derfor sies å være såpass tynt at det i de fleste tilfeller bare er mulig å indikere funn, ikke slå de kategorisk fast.

Avklaring av tvetydigheter og uklarheter

Ut fra skjermbildeprototypen identifiserte testdeltakerne en rekke tvetydigheter og uklarheter. Disse ble delvis oppklart av de gradvis mer abstrakte representasjonene, men noen spørsmål forble uavklarte (se pkt 6.5 - “Oppsummering”).

Høyere abstraksjonsgrad ga økt forståelse for interaksjon

Ett av hovedformålene til eksperimentet var å undersøke om interaksjonen ble enklere å forstå for testdeltakerne etter som dekningsgraden falt og formaliteten, abstraksjons- og abstraheringsgraden økte.

Eksperimentet indikerer at denne antakelsen var rett og at eksperimentets grunnpremiss dermed holdt. Siden det samme caset ble brukt hele veien, var det mulig å observere hvor representasjonene virket oppklarende.

Den kanonisk abstrakte prototypen hjalp ikke testdeltakerene nevneverdig i forhold til skjermbildeprototypen med hensyn på å avklare spørsmål de hadde rundt brukergrensesnittets interaksjon, men var i stand til å avklare spørsmål rundt komponentenes konseptuelle tilhørighet.

Når det gjelder ProtoMODL og DiaMODL var det ingen tvil hos testdeltakerene at disse representasjonene ga en dypere forståelse av den underliggende interaksjonen enn skjermbildeprototypen. Ingen av representasjonene var i stand til å fjerne alle tvetydighetene, men ProtoMODL- of DiaMODL-diagrammene beskrev interaksjonen mest presist og oppklarte/bekreftet mange av dem.

Bekreftende observasjon

Disse funnene ble til en viss grad bekreftet i eksperimentets andre del. Testdeltakerne hadde store problemer med å tolke interaksjonen ut fra skjermbildeprototypen og måtte foreta mange kvalifiserte gjetninger. Dette tyder på at en skjermbildeprototyp i utgangspunktet er en tvetydig representasjon.

Fra de observasjonene som ble gjort under arbeidet med å modellere utvidelsen og svarene som ble gitt i debrifingen, er det tydelig at det å modellere interaksjonen til den utvidede skjermbildeprototypen gjør at man må tenke gjennom komponentenes interaksjon ekstra nøye, og at denne aktiviteten både identifisere tvetydigheter i brukergrensesnittet og hjelper til å avklare dem.

Representasjonenes informasjonsinnhold

Eksperimentet indikerer at verken skjermbildeprototypen eller den kanonisk abstrakte prototypen er i stand til å gi særlig *nøyaktig* informasjon om interaksjonen. Den kanonisk abstrakte prototypen forklarer litt mer, men er mye vanskeligere å tolke enn både skjermbildeprototypen og ProtoMODL-diagrammet. I en skjermbildeprototyp er tolkingen basert på gjenkjennbarhet; når man ser en komponent man har brukt før, vet man intuitivt hvilken funksjonalitet den har. Informasjonsmengden som en skjermbildeprototyp inneholder, kommer altså an på om komponentene som benyttes er velkjente for mottakreren.

Ut fra datagrunnlaget kan det virke som om DiaMODL er mest formell og gir best informasjon om dataflyten i brukergrensesnittet. DiaMODL har også den mest spesialiserte informasjonen av de fire representasjonene.

ProtoMODL gir mindre nøyaktig informasjon om dataflyt, men inneholder mer informasjon om interaktorenes funksjonalitet (gjennom interaksjonssymbolene) og interaksjonen med brukeren (gjennom interaktorbeskrivelsen). ProtoMODL inneholder altså mest informasjon totalt.

Det er også indikasjoner på at ProtoMODL er i stand til å gi fra seg informasjon om interaksjonen raskere og lettere. Personlig mente testdeltakerene at en kombinasjon av skjermbildeprototyp og ProtoMODL-diagram ville være mest informativt.

Modellering

Når interaksjonen i et brukergrensesnitt skal modelleres, kan det se ut som om ProtoMODL fungerer bra til dette formålet. Et stort problem som bør endres er at oversikten over de kanonisk abstrakte komponentene ikke er klar nok. Notasjonen bør også endres på enkelte punkter for å gjøre denne aktiviteten enklere.

Det er umulig å si hvordan DiaMODL egner seg til dette ettersom problemdomenet var velkjent når testdeltakerne tok til med denne aktiviteten. Imidlertid ser det ut tiol at det er relativt enkelt å overføre informasjon fra et ProtoMODL-diagram til et DiaMODL-diagram.

Analyse av representasjonenes notasjon

En annen viktig hensikt med eksperimentet var å undersøke notasjonen til den kanonisk abstrakte prototypen, ProtoMODL og DiaMODL, med hensyn på hvor lesbare de er, hvor mye informasjon som kan leses ut av representasjonene og hvor enkle de er å bruke.

Skjermbilde

Det ble også slått fast at en skjermbildeprototyp uten en tekstlig beskrivelse er en tvetydig representasjon når det gjelder å spesifisere interaksjonen innad i et bruke-

rgrensesnitt og hvordan samspillet mellom brukergrensesnittet og en bruker foregår.

Kanonisk abstrakt prototyp

Den kanonisk abstrakte prototypen ser ut til å gi testdeltakerene lite utfyllende informasjon om brukergrensesnittet. I motsetning til skjermbildeprototypen er den heller ikke intuitiv.

Den stiplede boksen som benyttes i notasjonen ser ut til å øke testdeltakerenes forståelse av brukergrensesnittets oppbygning og den konseptuelle tilhørigheten til komponentene. Mest oppklarende er nok allikevel den utfyllende informasjonen inne i krøllparantesene. Ingen av disse elementene er imidlertid sentrale for denne notasjonen, og lignende notasjon kan finnes igjen i mange typer abstrakte prototyper.

Når det gjelder interaksjonen virker det som om navnene som gis de kanonisk abstrakte komponentene er mer informative enn symbolene som representerer disse komponentene. Symbolene er vanskelige å forstå, og det kan virke som om dette gjør det verre å tolke selve brukergrensesnittet. Det kan se ut som om de kanonisk abstrakte komponentene i seg selv gir veldig lite informasjon om interaksjonen.

Dette indikerer at den kanonisk abstrakte prototypen ikke er i stand til å beskrive interaksjonen klart. En bedre symbolforklaring kan være nødvendig for at brukere skal få fullt utbytte av denne notasjonen.

ProtoMODL:

Notasjonen til ProtoMODL ser ut til å være rask å sette seg inn i. Diagrammet framstår som ryddig og rask å lese. Testdeltakerene brukte litt tid å sette seg inn i interaksjonen som ble beskrevet, men når det var gjort ga diagrammet god oversikt over dataflyt og funksjonalitet.

Det er et pluss at diagrammet gir beskjed om hvilke handlinger brukeren skal utføre mot systemet og å se de to ProtoMODL-diagrammene samtidig ser ut til å gi mye informasjon om både interaksjonen og komposisjonen i brukergrensesnittet.

Et problem med notasjonen er at koblingene kan være forvirrende og at dette gjør det vanskelig å se dataflytretningen. Det var dissens blant testdeltakerene om den nåværende notasjonen, med utkoblinger som går fra interaktorens over- og undersiden og innkoblinger som kommer fra venstre, er tydelig nok, men notasjonen bør uansett endres slik at retningen trer klarere fram i modellen..

Forskjellen på en funksjon (som endrer visningen av en interaktor) og en metode (som endrer innholdet i et objekt) er ikke tydelig nok. Notasjonen bør endres slik at den tar hensyn til dette.

Et tredje problem er knyttet til at de kanonisk abstrakte komponentene ikke er identifisert godt nok. Dette var spesielt tydelig når testdeltakerene skulle modellere

selv, hvor testdeltakerene slet mye med å plukke ut hvilke komponenter som passer best til å beskrive interaksjonen til en brukergrensesnitt-komponent.

En av testdeltakerene savnet informasjonsobjektene, at objektene og -settene ikke var klart angitt. Den andre mente det var "litt deilig å slippe ôg", og syntes notasjonen var i stand til å uttrykke dette gjennom interaktorsymbolene og -beskrivelsen. Dette understreker at forståelsen av hva interaktorsymbolene står for er svært viktig for hvor mye informasjon man kan lese ut av ProtoMODL-diagrammet. Ettersom det var dissens blant testdeltakerene er det umulig å si noe om denne forskjellen mellom DiaMODL og ProtoMODL.

Det kan videre se ut som om det å vise at en interaktor inneholder et subsett av et sett er vanskelig å vise i ProtoMODL. Slike detaljer kan være med å heve brukerterskelen for notasjonen, og i verste fall føre til at en eventuell bruker av notasjonen opplever break-downs og en følelse av at notasjonen ikke strekker til. Akkurat dette momentet var imidlertid dårlig forklart i den beskrivelsen testdeltakerene fikk utlevert om notasjonene til den kanonisk abstrakte prototypen, ProtoMODL og DiaMODL (se vedlegg 1). Det er derfor ikke mulig å trekke noen slutninger om dette.

Hvor godt ProtoMODL klarer å overføre informasjonen om et brukergrensesnitts interaksjon ser ut til å avhenge av hvor kjent man er med symbolene som representerer de kanonisk abstrakte komponentene. Den testdeltakeren som var mest familiær med betydningen av disse symbolene var også den som fikk mest utbytte av ProtoMODL-notasjonen. Uansett virker det som om bruken av slike symboler gjør det lettere å se hvilke interaktorer som har samme type funksjonalitet.

DiaMODL

Eksperimentet indikerer at modeller laget med DiaMODL-notasjonen er mer rotete enn de som er laget med ProtoMODL. Testdeltakerene brukte nevneverdig mer tid på å lese interaksjonen i denne notasjonen enn i ProtoMODL, selv om problemområdet skulle vært godt kjent fra før. Denne observasjonen forsterker det førsteinntrykket om DiaMODL-diagrammet testdeltakerene ga uttrykk for i debrifingen. Imidlertid skal det bemerkes at diagrammet beskrevet i DiaMODL-notasjonen på flere punkter var forvirrende. Det er derfor umulig å si om dette er en reel indikasjon.

Når først dataflyten er identifisert og man har satt seg inn i modellen ser det ut til at den beskriver interaksjonen internt i brukergrensesnittet svært presist. Spesielt på informasjonsflyt virker det som notasjonen er veldig oppklarende.

Ettersom de største problemene som oppstod rundt denne notasjonen var knyttet til selve modellen, er det ikke mulig å si noe håndfast om notasjonen, men det kan virke som om kombinasjonen interaktorer og interaksjonsobjekter krever mer av mottakeren, enn interaktorer med symbol og beskrivelse.

Foretrukne representasjoner

Det kan se ut som om ProtoMODL-notasjonen er enklere å komme inn i enn DiaMODL. Rent subjektivt, var det dissens mellom deltakerene om hvilken representasjon de foretrakk. Den ene likte best DiaMODL, siden den var familiær, ga best informasjon om objektene som inngikk i interaksjonen og var mest nøyaktig på dataflyt. Den andre foretrakk ProtoMODL-representasjonene, som ga informasjon om både komponentenes utseende og funksjonalitet, samt interaksjonen i brukergrensesnittet på samme tid.

Oppsummering

Det virker ikke som om de to dialogmodellene gir noen nevneverdig forskjell i forståelsen av interaksjonen innad i et brukergrensesnitt. Den tydelige forskjellen i oversikt, som ble bemerket av testedeltakerene, kontra forskjellen i kontrollstruktur er allikevel en interessant observasjon. Det kunne vært spennende å undersøke mer om det er en sammenheng som gir at en mindre streng kontrollstruktur gir mer oversikt og tydeligere diagrammene.

Ellers mente testdeltakerene at det var fint at diagrammene ikke bare viste flyt, men at man også kunne legge inn funksjoner. Også muligheten til å spesifisere i diagrammet hvilke data man ønsker å få inn til komponentene, og at man kan vise koblinger mot systemet ble nevnt som positive egenskaper ved de to dialogmodelleringsspråkene.

Bruksområde

Eksperimentet gir også en del indikasjoner på hvilke bruksområder som er mest velegnet for de forskjellige representasjonene.

Den abstrakte prototypen ser ut til å egne seg best som et verktøy for å støtte oppunder designprosessen. Imidlertid kan også ProtoMODL-prototypen benyttes til denne oppgaven.

De to interaksjonsmodellene kan være aktuelle å benytte som et kommunikasjonsverktøy i en designprosess for å formidle til andre interessenter mest mulig utvetydig informasjon om interaksjonen, men vil antakelig være vanskelig å benytte som kreative verktøy.

Til å dokumentere den helhetlige interaksjonen i et system finnes det indikasjoner på at et ProtoMODL-diagram sammen med et skjermbilde vil være best egnet. Skal kun dataflyten og brukergrensesnittets interne interaksjon dokumenteres, er kanskje DiaMODL bedre egnet.

DiaMODL vil det være naturlig å benytte som spesifikasjon til en utvikler når et system skal utvikles, men dette kommer an på mottakeren. Som spesifikasjon til en typisk designer mente testdeltakerene at ProtoMODL kanskje egnet seg bedre.

Hvis de resultatene om bruksområde som eksperimentet indikerer settes opp i en tabell, ser vi at ProtoMODL ser ut til å være egnet til både å utforske, spesifisere, dokumentere og kommunisere interaksjon i et brukergrensesnitt. Hovedformålet

med å utvikle notasjonen vil i så fall være oppnådd. Imidlertid må notasjonen prøves ut på flere personer, helst med forskjellige ferdigheter og fra ulike utviklings- og designmiljøer.

Tabell 6-1 viser de fire representasjonenes potensielle bruksområder, der et kryss indikerer hvor hver enkelt representasjon virker velegnet ut fra eksperimentet. Et kryss i parantes indikerer at representasjonen har potensiale til å brukes på disse områdene, men at en annen representasjon antakelig egner seg bedre. En strek indikerer at representasjonen virker uegnet til dette bruksområdet.

Tabell: 6-1: Representasjonenes potensielle bruksområder

	Design	Spesifikasjon	Dokumentasjon	Kommunikasjon
Skjermbilde	-	-	X	-
Kan. abstrakt prototyp	X	-	-	-
ProtoMODL	X[a]	(X)	X	X
DiaMODL	-	X	(X)	X

a. ProtoMODL-prototyp

6.1 Testing av hypotese

Hypotesen ble ikke falsifisert av eksperimentet. Testdeltakerene var i stand til både å tolke et diagram skrevet i ProtoMODL og modellere interaksjonen i et brukergrensesnitt ut fra en annen representasjon. Dette viser at ProtoMODL kan benyttes til både å dokumentere og spesifisere interaksjon i et brukergrensesnitt. ProtoMODL-notasjonen ble også brukt som et kommuniserende medium av testdeltakerne; de tegnet opp interaksjonen ved hjelp av ProtoMODL-interaktorer for å forklare hverandre hvordan de mente interaksjonen i det utvidede brukergrensesnittet foregikk. Hypotesen om at ProtoMODL er et effektivt designverktøy som kan hjelpe designere å utforske designrommet, ble imidlertid ikke testet i dette eksperimentet. Teoretisk skal den imidlertid fungere minst like bra som en kanonisk abstrakt prototyp på dette området.

Eksperimentet indikerer også at ProtoMODL-diagrammene er i stand til å oppklare tvetydigheter som vil finnes i en skjermbildeprototyp som ikke følges av en tekstlig forklaring på interaksjonen mot brukeren. Det er imidlertid ganske klart at et ProtoMODL-diagram alene neppe vil være i stand til å avklare alle tvetydigheter som vil eksistere i en skjermbildeprototyp.

Eksperimentet var imidlertid ikke i stand til å avkrefte eller bekrefte hypotesen om at ProtoMODL gjør det lettere for en designer å spesifisere, dokumentere eller

kommunisere interaksjon. Det er heller ikke mulig å slå kategorisk fast om ProtoMODL egner seg bedre eller verre til dette enn DiaMODL. Men datagrunnlaget indikererer at ProtoMODL er lettere å lese, har større dekningsgrad og gir informasjonen lettere fra seg, mens et DiaMODL-diagram inneholder mer spesialisert informasjon dersom man klarer å sette seg skikkelig inn i notasjonen.

6.2 Oppsummering

Oppgavens empiri er grunnlagt på et kvalitativt eksperiment, der testdeltakerne fikk i oppgave å først tolke interaksjonen som er spesifisert i fire forskjellige designrepresentasjoner, og deretter selv modellere interaksjon på bakgrunn av en utvidelse av funksjonaliteten i brukergrensesnittet.

Eksperimentet må anses som vellykket ettersom det var i stand til å gi klare data på flere områder, gi viktige indikasjoner på en del andre og teste det meste av hypotesen uten at den ble falsifisert.

Eksperimentet påviste også at det er nødvendig med en revisjon av notasjonen. En slik revisjon presenteres i neste kapittel.

Kapittel 7

Evaluering

Dette kapittelet presenterer en del generelle funn og indikasjoner som har kommet fram på bakgrunn av eksperimentets generelle datagrunnlag og selve prosessen med å utarbeide notasjonen. Det beskriver også den reviderte notasjonen som eksperimentet påviste var nødvendig. Så skisseres det hvordan denne oppgaven kan danne grunnlag for videre arbeid, før oppgaven avsluttes med en konklusjon.

7.1 Generelle funn

De fleste av funnene som ble gjort i forbindelse med eksperimentet er beskrevet i resultatdelen av forrige kapittel. På bakgrunn av prosessen med å utvikle og teste notasjonen, er det imidlertid også mulig å trekke en del generelle slutninger.

Presentasjon av målsetning

Antakelsen som ble gjort om at det var mulig å lage en hybrid representasjon av en interaksjonsprototyp og en interaksjonsmodell viste seg å være riktig. Resultatet ble en representasjon som både kan klassifiseres som en abstrakt prototyp, en innholdsmodell eller en dialogmodell, alt ettersom hvordan man betrakter representasjonen.

Hovedformålet med å utvikle ProtoMODL var først og fremst å lage en notasjon som beskriver den interne dataflyten i et brukergrensesnitt, funksjonaliteten til de enkelte brukergrensesnittkomponenter og interaksjonen mellom bruker og system, samt være i stand til å spesifisere hvilke generiske komponenter brukergrensesnittet består av og deres innbyrdes plassering. Det var også et poeng at notasjonen skulle være i stand til å utforske, dokumentere og kommunisere interaksjon. Både eksperimentet og prosessen viste at ProtoMODL til en viss grad greide å innfri alle disse målsetningene.

En annen målsetning var at notasjonen skulle være i stand til å virke som en brobygger mellom den kanonisk abstrakte prototypen og DiaMODL. Den enetse overgangen som ble testet empirisk var å overføre informasjonen i et ProtoMODL-diagram til et diagram beskrevet i DiaMODL-notasjonen. Dette gikk svært bra. De andre overgangene har ikke blitt testet empirisk, men i teorien skal den foreslåtte metoden som er beskrevet i kapittel 5 fungere. En av forutsetningene for å benytte denne metoden er imidlertid at man kan alle notasjonene svært godt. En editor som kan støtte slike overganger hadde vært til stor hjelp i denne prosessen.

Analyse kontra eksperiment

I kapittel 5 ble det foretatt en analyse av hvordan de tre representasjonene skjermbilde, kanonisk abstrakt prototyp og DiaMODL rent teoretisk burde være i stand til å utføre generelle oppgaver innen det å utforske, spesifisere, dokumentere og kommunisere interaksjonen som skjer innad i et brukergrensesnitt. Sammenligner vi tabellen som ble satt opp etter analysen med tabellen som ble laget på bakgrunn av observasjonene som ble gjort under eksperimentet og debrifingen etterpå, ser vi at analysen traff bare halvveis. Den kanonisk abstrakte prototypen viste seg å være dårlig egnet til å benytte som et kommunikasjonsverktøy, mens DiaMODL virker bedre egnet til å kommunisere interaksjon til andre interessenter enn forutsagt. Dette krever imidlertid at alle interessenter er godt kjent med notasjonen.

Problemdomenet utvikler seg

I prosessen med å lage forskjellige representasjoner av et brukergrensesnitt, var en av erfaringene at problemdomenet hele tiden utviklet seg. Spesielt ProtoMODL og DiaMODL tvinger designeren til å tenke gjennom designløsningene og fokusere på interaksjon i stedet for visuell design. Arbeidet med å spesifisere interaksjonen gjør også at forståelsen for hva som vil være en bra eller dårlig designløsning øker.

Krav til notasjonen

Kravene til notasjonen var at den skulle være lett forståelig, ha lav bruksterskel, kunne utføres med penn og papir og sist men ikke minst gi nøyaktig informasjon om sine dekningsområder. Eksperimentet som ble utført indikerer at disse kravene langt på vei er oppfylt, men at noe arbeid gjenstår med å gjøre notasjonen tydeligere.

Spesifisering av interaksjon i brukergrensesnitt

Hvilken informasjon om brukergrensesnittet man er i stand til å lese ut av en representasjon kommer helt an på mottakeren; altså hvilke notasjoner man er kjent med og hvordan man er vant med å få informasjonen servert. Observasjoner indikerer at DiaMODL er bedre å bruke som spesifikasjon til en utvikler, mens en typisk designer vil ha mer utbytte av ProtoMODL.

Bruksterskel kontra brukshøyde

ProtoMODL ser ut til å ha omtrent samme bruksterskel som den kanoniske abstrakte prototypen. Koblingene som viser dataflyt bidrar til å gjøre interaksjonen tydeligere, og fristillingen fra prototypens layout ser ikke ut til å ha stor betydning for forståelsen av brukergrensesnittets interaktorer. Brukshøyden er relativt stor. Notasjonen er i stand til å vise dataflyten like detaljert som DiaMODL, og selv om representasjonen mister en del informasjon ved å utelate objektstrukturen som vi ser i DiaMODL, inneholder den relativt detaljert informasjon om hvordan brukeren samhandler med det spesifiserte brukergrensesnittet. ProtoMODL kan derfor ha potensiale til å fungere i praksis.

Det er imidlertid et viktig poeng at testdeltakerene var familiære med en svært lik notasjon før de satte seg inn i ProtoMODL. For å kunne si noe konkret om brukerterskelen, vil det være nødvendig å observere personer som ikke kan DiaMODL fra før mens de setter seg inn i ProtoMODL, og intervjue de om deres erfaringer rundt denne prosessen etterpå.

7.2 Revidering av notasjon

På bakgrunn av eksperimentet, samtaler med fagpersoner samt innsikt opparbeidet etter hvert som flere diagrammer ble tegnet, har notasjonen blitt oppdatert og forbedret på flere punkter.

Eksperimentet viste at et av problemene for testdeltakerene var å identifisere retningen på dataflyten i koblingene som går mellom interaktorene.

Det første forslaget til en revidert notasjon ble laget på bakgrunn av observasjonene som ble gjort under eksperimentet, de umiddelbare kommentarene testdeltakerene kom med etterpå og diskusjoner med veileder.

I denne notasjon ble port-symbolet fra DiaMODL-notasjonen brukt som utgangspunkt for å finne et symbol som kunne vise inn- og ut-retning på koblingene. Tanken var at portens “base” kunne vise at data gikk mot en interaktor, mens portens “tupp” kunne vise at data gikk fra den.

Timeavtaleapplikasjonen modellert med denne ProtoMODL-notasjonen, ble da seende slik ut:

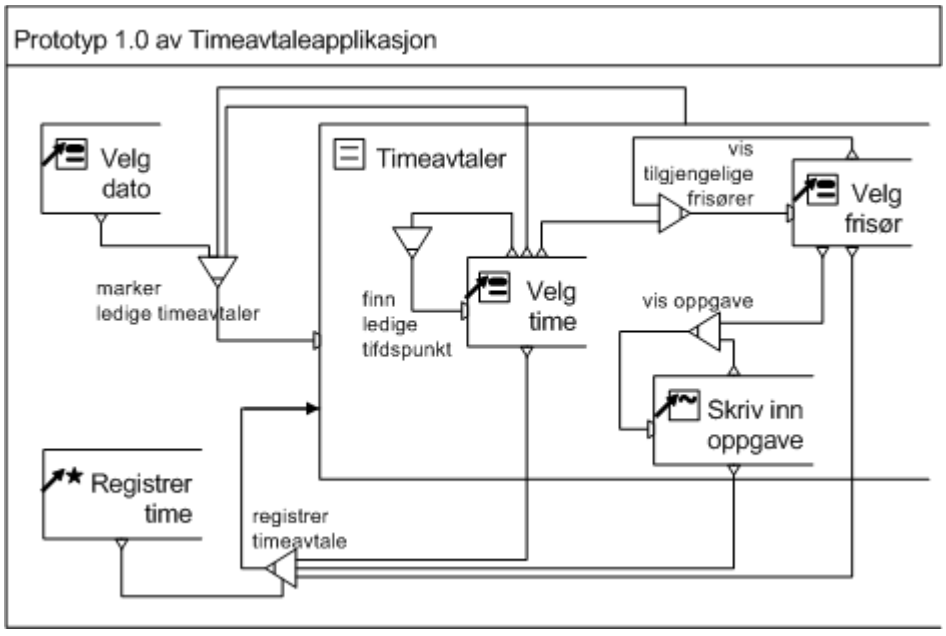


FIGURE 7-1: Initielt forslag til forbedringer i notasjonen

Et problem med en slik notasjon er at den nærmer seg kraftig notasjonen til DiaMODL. Brukere vil antakeligvis bli forvirret av at endene til koblingene grafisk ser ut som porter i DiaMODL, når de faktisk ikke har noen annen funksjon i notasjonen enn å vise retning. I tillegg er symbolbruken inkonsekvent, siden forskjellig symbolikk brukes for å markere funksjoner og metoder. Diagrammet blir også mer tungvint å tegne for hånd.

Den endelige notasjonen tar hensyn både til (1) nødvendig avstand til DiaMODL-notasjonen (2) konsekvent symbolbruk og (3) brukervennlighet ved skissering med penn og papir.

Her brukes konsekvent pilspisser mot interaktorenes venstre side for å identifisere koblinger som bærer inndata til interaktorene. Inndata fra *funksjoner* er symbolisert med en åpen pilspiss, mens inndata fra *metoder* er symbolisert med en fylt pilspiss. Siden alle koblinger som leder data inn til en interaktor er markert, er det ikke nødvendig å legge til symbolikk som understreker at en kobling leder fra en interaktor. Den opprinnelige notasjonen, der koblinger som bærer utdata er tegnet som en ren linje fra interaktorens over- eller underside, er tydelig nok. Denne notasjonen vil gjøre det lettere å følge dataflyten i diagrammet og være mindre forvirrende for brukeren.

Konseptet med at nøstede interaktorer fritt kan hente data fra sin superinteraktor viste seg også å være vanskelig å bruke i praktisk modellering. Det er derfor lagt til en mekanisme som eksplisitt angir når data fra en superinteraktor påvirker dataene til en subinteraktor. Når det er behov for å presisere at en interaktor skal inneholde de samme data som forelderinteraktoren, kan dette markeres med en sirkel og en kobling som leder direkte til den aktuelle interaktoren. Dersom dataene fra forelderinteraktoren trengs i en utregning, vil koblingen fra en slik sirkel i stedet lede til en funksjon.

Med den endelige notasjonen vil da timeavtaleapplikasjonen bli seende slik ut:

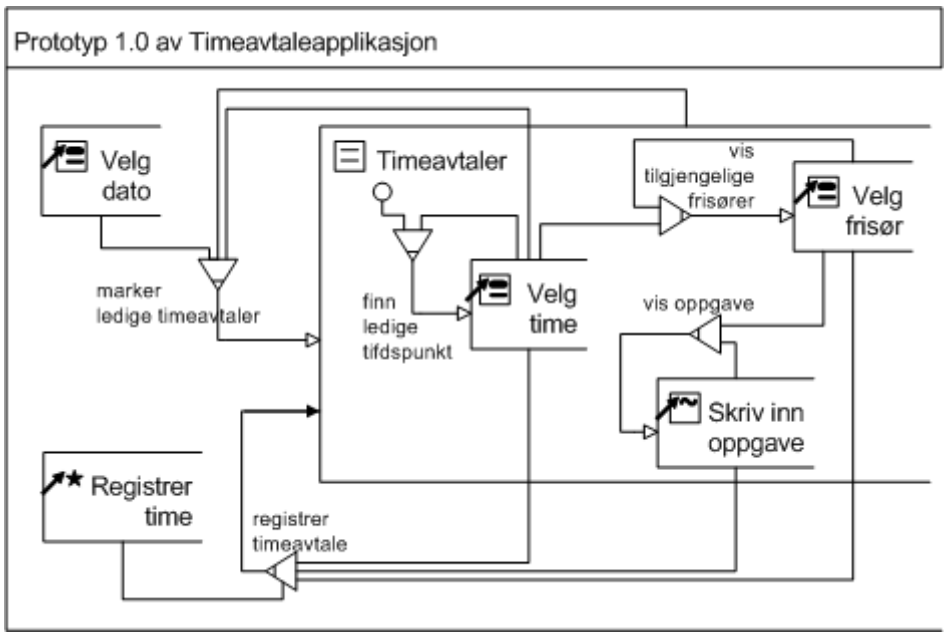


FIGURE 7-2: Endelig notasjon

7.3 Videre Arbeid

Utvikling av bedre interaksjonssymboler

Eksperimentet viste at det bør utarbeides nye, mer intuitive symboler som kan erstatte de kanonisk abstrakte komponentene. Dette vil gjøre notasjonen lettere å lese. Interaksjonssymbolene bør baseres på internasjonalt kjente symboler som hvem som helst, uansett nasjonalitet, kan gjenkjenne. En mulig ide er for eksempel å undersøke trafikk skilt og se om de symbolene som brukes kan overføres til generiske komponenter.

ProtoMODL som en brobygger mellom modell og prototyp

En av målsetningene til oppgaven var som sagt å lage en metode som gjør det enklere å overføre informasjon om interaksjon til/fra en kanonisk abstrakt prototyp og DiaMODL. Den foreslåtte metoden bør utprøves i et praktisk, empirisk eksperiment, helst in vivo.

ProtoMODLs virkning i det virkelige liv

I forhold til å undersøke hvordan ProtoMODL kan fungere som spesifikasjons-, dokumentasjons- og kommunikasjonsverktøy i relle design- og utviklingsmiljø er det flere ting å ta tak i.

Spesielt interessant er det å se på hvordan ProtoMODL fungerer sammen med andre representasjoner, og da særlig de som benyttes innen UML siden dette er en de-facto standard i mange utvikingsmiljøer. Et av spørsmålene da er hvordan man kan overføre informasjonen som finnes i en ProtoMODL modell til for eksempel Use case-, sekvens-, samarbeids- og klassediagrammer.

Også det å se hvordan ProtoMODL kan benyttes i underliggende design- og systemutviklingsprosessene som benyttes er en spennende utfordring. For eksempel kan man undersøke hvordan ProtoMODL fungerer som verktøy i henholdsvis RUP/UA og XP, og i hvilken grad måten man arbeider med verktøyet er forskjellig i disse to prosessene.

Utvikling av editor

På det mer tekniske plan gjenstår det selvsagt mye arbeid i å utvikle en editor som gjør det mulig å gå mellom de forskjellige representasjonene. Særlig interessant hadde det vært å utvikle en editor som gjør det mulig å generere et brukergrensesnitt basert på et ProtoMODL-diagram og som kan bryte opp et brukergrensesnitt i interaktorer og modellere interaksjonen mellom dem ved hjelp av ProtoMODL-notasjonen.

Automatisk kodegenerering

Et litt mer tvilsomt prosjekt er å undersøke om det er mulig å automatisk generere kode ut fra et ProtoMODL-diagram. Dersom man definerer en interaktor som et objekt med et grensesnitt mot andre komponenter og det underliggende systemet, er det en liten mulighet for at dette lar seg gjøre.

7.4 Konklusjon

Denne oppgaven har utforsket og sammenlignet de to fagfeltene interaksjonsmodellering og interaksjonsprototyping med det formål å lage en hybrid representasjon. Representasjonen som er utviklet er i stand til å utnytte modellens abstrakte tankegang og evne til å beskrive skjult funksjonalitet med prototypensevne til å presentere denne informasjonen på en konkret og intuitiv måte.

Eksperimentet som ble gjennomført indikerer at ProtoMODL er en representasjon med høy dekningsgrad som er rask å sette seg inn i. Den har relativt lav bruksterskel, men potensielt høy brukshøyde. Diagrammene fremstår som ryddige og lettleste, interaksjonen ser ut til å være beskrevet relativt presist og modellen virker i stand til å avklare tvetydigheter og uklarheter i et brukergrensesnitt. Dette er imidlertid stort sett indikasjoner, og mer empiri er nødvendig for å fastslå dette med sikkerhet.

ProtoMODL har potensiale til å bli brukt innenfor så forskjellige områder som utforskende design, spesifisering og dokumentering av interaksjon og til å kommunisere ideer rundt et brukergrensesnitts interaksjon mellom interessenter i en designprosess.

Ved å kombinere de to fagfeltene interaksjonsmodellering og -prototyping, har denne oppgaven åpnet for en ny måte å betrakte disse tilnærmingene på. Ulikhetene mellom tradisjonell systemutvikling og kreativ design er kanskje ikke så stor som mange har trodd. I denne oppgaven er det i hvert fall vist at det er mulig å dra nytte av begge områdene når man designer brukergresnsenitt.

Motivasjonen for å lage denne notasjonen var et ønske om at både designere og utviklere skulle få et verktøy begge var i stand til å bruke effektivt. Dette ser ut til å ha lykkes. En viktig oppgave videre vil derfor være å spre notasjonen til personer som trenger et verktøy som kan beskrive interaksjon internt i et brukergrensesnitt og interaksjon mellom system og bruker.

Referanser

Beaudouin-Lafon et. al., 2003	Beaudouin-Lafon, M., Mackay, W.: "The Human-Computer Interaction Handbook: Fundamentals, Evolving Technologies and Emerging Applications", kapittel 52: "Prototyping Tools and Techniques", Erlbaum Associates, 2003
Bråten, 2004	Bråten, A. E.: “Rational Unified Prosess - et godt rammeverk for design av brukergrensesnitt?”, semesteroppgave i faget MNFIT 364: “Systemutvikling, organisasjon og arbeidsliv”, NTNU, juni 2004 (vedlagt)
Campos et. al., 2004	Campos P. F., Nunes N. J.: “CanonSketch: a User-Centered Tool for Canonical Abstract Prototyping”, i “11th International Workshop on Design, Specification and Verification of Interactive Systems”, Springer-Verlag, 2004.
Constantine et. al., 1998	Constantine, L. L.: “Rapid Abstract prototyping”, i “Software Development”, 6, (11), Oktober 1998. Revidert utgave trykket i:S. Ambler and L. Constantine (eds.), “The Unified Process Elaboration Phase: Best Practices in Implementing the UP”, CMP, Lawrence, KS, 2000.
Constantine et. al., 2003	Constantine, L. , Windl, H., Noble, J., Lockwood, L.: “From Abstraction to Realization: Canonical Abstract Prototypes for User Interface Design REVISED”, http://www.foruse.com/articles/canonical.pdf, juli 2003
Constantine, 2003	Constantine, L. L.: “Canonical Abstract Prototypes for Abstract Visual and Interaction Design”, s 1-15, i Jorge J. A. et. al (Eds): “Proceedings DSV-IS 2003”, LNCS 2844, Springer Verlag, Berlin, 2003
Dick, 2001	Dick, D.: “Strategies for usability - Putting ISO standards to practice“, http://www.stc.org/confproceed/2001/PDFs/STC48-000071.PDF, 2001 (tilgjengelig pr. 02. 06 2006)

Dix et. al., 1993 — Dix, A., Finlay, J., Abowd, G. and Beale, R.: “Human-Computer Interaction”, Prentice Hall, 1993

Ehn, 1993 — Ehn, P.: Scandinavian Design: On Participation and Skill”, s. 41-78 i D. Schuler og A. Namioka (eds): Participatory Design: Principles and Practices . Lawrence Erlbaum ass., 1993

Flataukan, 2006 — Flataukan, K.: “Statecharts som dialognotasjon for xGUIMS”, Hovedfagsoppgave, NTNU, Mars 2001

Fleming, 1998 — Fleming, J.: “Web Navigation: Designing the User Experience”, O'Reilly & Associates, 1998.

Floyd, 1987 — Floyd, C.: “Outline of a paradigm change in software engineering”, s. 193-210, i Computers and democracy, Avebury, 1987.

Hartvigsen, 1998 — Hartvigsen, G.: “Forskerhåndboken”, Høyskoleforlaget, Norge, 1998

Hix et. al., 1993 — Hix, D., Hartson, H.R.: “Developing User Interfaces”, Wiley, New York, 1993

Houde et. al., 1997 — Houde, S., Hill, C.: “Handbook of Human-Computer Interaction”, 2nd edition, kapittel 16: “What do Prototypes Protoptype?”, Elsevier, 1997

Horrocks, 1999 — Horrocks, I.: “Constructing the User Interface with Statecharts”, Addison Weshley, 1999.

Kim, 1995 — Kim, S.: “Interdisciplinary cooperation”, s. 305-311. i Baeker, R. M. (ed.), “Readings in Human- Computer Interaction: Toward the Year 2000”, Morgan Kaufmann, San Francisco, 1995

Kruchten, mars 2001 — Kruchten, P., Ahlqvist, S., Bylund, S.: "User Interface Design in the Rational Unified Process”, s. 161-196, i Object Modeling and User Interface Design. Designing Interactive Systems, Mark van Hamelen (ed), Addison-Wesley, Mars 2001

Kruchten, juni 2001 — Kruchten, P.: The Rational Unified Process An Introduction, Second Edition, Addison-Wesley, juni 2001

Mao et. al, 2005 — Mao, J. , Vredenburg, K., Smith P.W., Carey, T.: “The state of user-centered design practice”, s.105-109, i Communications of the ACM, v.48 n.3, mars 2005

Mittet, 2006 — Mittet, N. J. K.: ”Diamodlgen - Kodegenerering basert på diamodl interaksjonsmodeller”, Hovedfagsoppgave, NTNU, 2006

Myers, et. al., 2000 — Myers, B. et al: “Past, Present and Future of User Interface Software Tools” s 3-28, i “ACM Transactions on Computer Human Interaction”, No 1, Mars 2000

Newman et. al, 2000 — Newman, M. W.,Landay, J.A.: “Sitemaps, storyboards, and specifications: a sketch of Web site design practice”, s. 263-274, i Proceedings of the conference

on Designing interactive systems: processes, practices, methods, and techniques, New York, United States, august 17-19, 2000

Nunes et. al., 2000 — Nunes, N.J. og Cunha, J. F.: “Wisdom - A UML based architechture for interaktive systems”, s.191-205, i “Proceedings of the 7th International Workshop on Design, Specification and Verification of Interactive Systems (DSV-IS2000)”, Patrno, F., Palanque, P. (Eds.), Springer-Verlag , New York, 2000

Preece et. al., 2002 — Preece, J., Rogers, Y., Sharp., H.:”Interaction design: Beyond human-computer interaction”, John Wiley & Sons, 2002

Sinha, 2004 — Sinha, R., Boutelle , J.: “Rapid information architecture prototyping”, s. 349 - 352, “Proceedings of the 2004 conference on Designing interactive systems: processes, practices, methods, and techniques”, Cambridge, USA, ACM Press, New York, USA, 2004

Trætteberg, 2002:1 — Trætteberg, H: “Model-based User Interface Design”, Doktoravhandling, NTNU, 2002

Trætteberg, 2002:2 — Trætteberg, H.: “Using User Interface Models in Design”, i Computer-Aided Design of User Interfaces III: Proceedings of the Fourth International Conference on Computer-Aided Design of User Interfaces, 15-17 May 2002, Valenciennes, France

Trætteberg, 2003 — Trætteberg, H.: “Dialog modelling with interactors and UML Statecharts – A hybrid approach”, Proceedings of DSV-IS 03, Funchal, Madiera, Portugal, June, 2003.

van der Aalst, 1998 — van der Aalst, Will M. P.: “Three Good Reasons for Using a Petrinet based Workflow Management System”, i “Information and Process Integration in Enterprises: Rethinking Documents”, bind 428, T. Wakayama, S. Kannapan, C.M. Khoong, S. Navathe og J. Yates, (red), Kluwer Academic Publishers, Boston, Massachusetts, 1998.

Vredenburg, K., Mao. J., Smith P. W, Carey, T.: “A survey of user-centered design practice”, Proceedings of the SIGCHI conference on Human factors in computing systems: Changing our world, changing ourselves, Minnesota, USA, april 20-25, 2002

Nettsider:

Marion, Nettside:1, 200? Marion, C.: Nettside: “The Software Design Smorgasbord”, http://www.chesco.com/~cmarion/, tilgjengelig pr. 02.06.2005

Marion, Nettside:2, 200? Marion, C.: *“What is Interaction Design and What Does It Mean to Information Designers?”*, http://www.chesco.com/~cmarion/PCD/WhatIsInteractionDesign.html, 18.04.2005

Navalkar, Nettside:3, 200? Navalkar, A.: “Usability Engineering – Quality Approach (ISO 13407)”, http://www.humanfactors.com/asia/iso.asp, 02. 06 2006

Scanlon, Nettside:4, 1998 Scanlon, T.: “Paper prototypes: Still our favorite”, http://www.uie.com/articles/paper_prototyping/, 1998, tilgjengelig per 02. 06. 2006

Snyder, Nettside:5, 2001 Snyder, C.: “Paper Prototyping” (2001), http://guir.berkeley.edu/courses/SummerHCI04/readings/Paper%20Prototyping.htm, tilgjengelig per 23. 03 2006

Tajakka, Nettside:6, 2005 Tajakka, S.: “ISO 9241-11, standarden för användbarhet”, http://www.santai.nu/artiklar/iso.htm, 20.12 2005

Trætteberg, Nettside:7, 200? Trætteberg, H.: Forelesningsfoil på nett: “Konseptuelle modeller, mentale modeller og metaforer“, http://www.idi.ntnu.no/emner/tdt4180/forelesninger.php, tilgjengelig pr. 20.12 2005

Zimerman, Nettside:8, 200? Zimmermann, A.: “Petri Nets”, http://pdv.cs.tu-berlin.de/~azi/petri.html, tilgjengelig pr. 02. 06 2006

Organisasjoner

IDI, NTNU, Org:1, 200? Opprinnelig referanse fjernet. informasjonen finnes i revidert utgave på: “IT3900 - Hovedfagsoppgave”, http://www.idi.ntnu.no/undervisning/allmenn/masteroppgaven.php, tilgjengelig pr. 02. 06 2006

ISO 9241-11, Org:2, 1998 SO 9241-11:1997, Guidance on Usability (1998), http://www.idemployee.id.tue.nl/g.w.m.rauterberg/lecturenotes/ISO9241part11.pdf, tilgjengelig pr. 02. 06 2006

SmartDraw UML Center, Org:3, 200? “SmartDraw UML Center: HOW TO DRAW UML DIAGRAMS”, http://www.smartdraw.com/tutorials/software-uml/uml.htm, tilgjengelig pr 02. 06 2006

Wikipedia, Nettside:11, 2006 Nettside: “Wikipedia, the free encyclopedia”, http://en.wikipedia.org/wiki/Main_Page, Søkeord: “Human–computer interaction”, 2006

Kompendium:

Trætteberg, Kompendium:1, 2004? Trætteberg, H.: Kompendium i faget sif8018: “Et lite kompendium i Systemutvikling”, NTNU, 2004?

Vedlegg 1

Testdeltakerene fikk på forhånd utlevert en beskrivelse av eksperimentet som beskrev kort hensikten med eksperimentet og hvordan selve gjennomføringen ville foregå. I denne beskrivelsen var også notasjonen til kanoniske abstrakte prototyper, DiaMODL og ProtoMODL beskrevet.

Tolkning av interaksjon i brukergrensesnitt.

- Eksperimentbeskrivelse -

Først vil jeg takke for at dere er villig til å stille opp på dette eksperimentet, hvor vi ønsker å undersøke om det er hensiktsmessig å benytte modeller til å tolke interaksjonen i et brukergrensesnitt.

Utførelsen av eksperimentet vil bli dokumentert på video dersom dere godtar dette. Videoen vil bare bli sett av meg selv, og eventuelt mine veiledere Hallvard Trætteberg og Dag Svanæs.

Eksperimentet vil utføres i to faser. I den første fasen er det meningen at dere skal tolke interaksjonen slik som dere oppfatter den ut fra noen ferdiglagde designartefakter. Artefaktene vil bli presentert etter abstraksjonsnivå, fra mest konkret til mest abstrakt.

I den andre fasen er det meningen at dere selv skal modellere den endringen i interaksjonen som oppstår etter en tenkt utvidelse av prototypens funksjonalitet. Her står dere fritt til å benytte de verktøy som dere selv synes virker mest hensiktsmessige. Modelleringen skal foregå med penn og papir.

Før vi starter kan det være nyttig å se gjennom den vedlagte beskrivelsen av notasjonen til hhv Constantines ”Canonical abstrakt prototyping” (heretter CAP) og Trættebergs DiaMODL. I tillegg vil det bli brukt et modellverktøy basert på CAP og DiaMODL som jeg kaller ProtoMODL. Denne er beskrevet til slutt i dette heftet.

Anders E. Bråten
- testleder

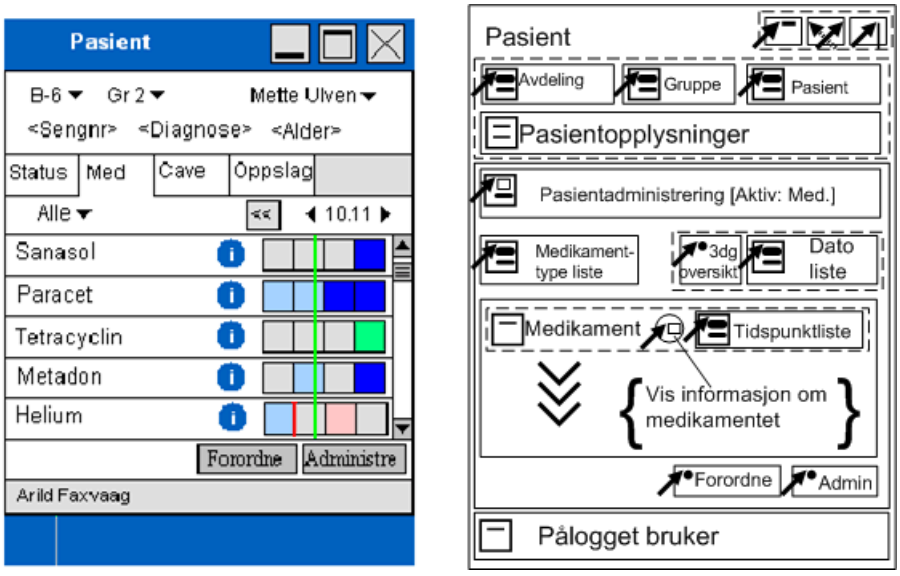
Canonical Abstrakt Prototypes:

CAP er et lofi prototypingsverktøy som baserer seg på bruk av abstrakte komponenter beskrevet i en ensartet (kanonisk) notasjon. Hensikten er at designere relativt enkelt skal kunne modellere innholdet til et brukergrensesnitt og eksperimentere med generell layout, uten å være nødt til binde seg til detaljer rundt selve utseendet og grafisk design. [1] Constantine selv hevder at CAP på dette viset, i tillegg til å være en relativ konkret designartefakt, også representerer en modell av den underliggende arkitekturen til brukergrensesnittet som skal designes. [1]

Selve prototypen konstrueres ut fra et verktøysett bestående av tre generiske abstrakte komponenter: materialer (materials), verktøy(tools) og aktive materialer (active materials). Under hver av disse generiske komponentene finnes et utvalg spesialiserte hjelpekomponenter, for mer eksakt beskrivelse av komponentens formål. [1]

Materialer brukes for å representere objekter i brukergrensesnittet som viser innhold eller informasjon, eller som på et vis blir manipulert eller presentert for brukeren under utførelsen av en oppgave, for eksempel en feilmelding. Verktøy blir brukt for å representere manipulering eller endring av materialer ved hjelp av operator, kontroller eller andre mekanismer, for eksempel en trykknapp eller en sjekkboks. [2] Aktive materialer er en kombinasjon av verktøy og materialer, og brukes for å representere et objekt som har karakteristikker fra begge komponentklassene, for eksempel editerbart tekstfelt eller en rullegardin. [3]

Notasjonen er senere blitt utvidet med tre spesialsymboler. En trippel vinkel indikerer en repeterende komponent eller komponentgruppe. En oppstreket ramme indikerer at komponentene har en konseptuel tilhørighet til hverandre. Kommentarer i krøllparanteser brukes for å avklare komponenters oppførsel eller spesielle karakteristikker i den abstrakte designen. [2]



Figur 1: CAP basert på skjermbilde i spesifikasjonen til GUIMED prosjektet.

Table 1 – Summary of Canonical Abstract Materials

MATERIALS

SYMBOL	INTERACTIVE FUNCTION	EXAMPLES
	container*	Configuration holder, Employee history
	element	Customer ID, Product thumbnail image
	collection	Personal addresses, Electrical Components
	notification	Email delivery failure, Controller status

Table 2 – Summary of Canonical Abstract Tools

TOOLS

SYMBOL	INTERACTIVE FUNCTION	EXAMPLES
	action/operation*	Print symbol table, Color selected shape
	start/go/to	Begin consistency check, Confirm purchase
	stop/end/complete	Finish inspection session, Interrupt test
	select	Group member picker, Object selector
	create	New customer, Blank slide
	delete, erase	Break connection line, Clear form
	modify	Change shipping address, Edit client details
	move	Put into address list, Move up/down
	duplicate	Copy address, Duplicate slide
	perform (& return)	Object formatting, Set print layout
	toggle	Bold on/off, Encrypted mode
	view	Show file details, Switch to summary

Table 3 – Summary of Canonical Abstract Active Materials

ACTIVE MATERIALS

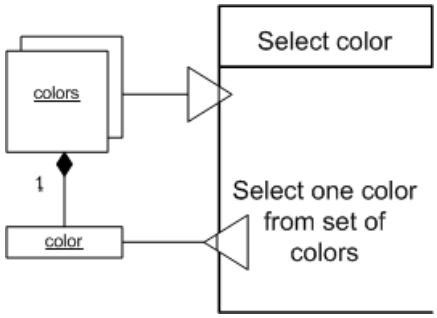
SYMBOL	INTERACTIVE FUNCTION	EXAMPLES
	active material*	Expandable thumbnail, Resizable chart
	input/accepter	Accept search terms, User name entry
	editable element	Patient name, Next appointment date
	editable collection	Patient details, Text object properties
	selectable collection	Performance choices, Font selection
	selectable action set	Go to page, Zoom scale selection
	selectable view set	Choose patient document, Set display mode

DiaMODL

Mens en oppgavemodell er en representasjon av de mål en bruker ønsker å få utført og de oppgaver som er nødvendige for å nå disse målene, beskriver en dialogmodell hvordan en bruker og et system samhandler via et brukergrensesnitt for å nå disse målene.

DiaMODL er et dialogmodelleringsspråk basert på interaktorabstraksjon i den hensikt å uttrykke informasjonsflyt innenfor en designrepresentasjon, ved å beskrive funksjonaliteten og oppførselen til konkrete interaksjonsobjekter. Interaktorkomponenter brukes for å gi input fra og generere output av spesifikk data til en bruker i direkte interaksjon. [4]

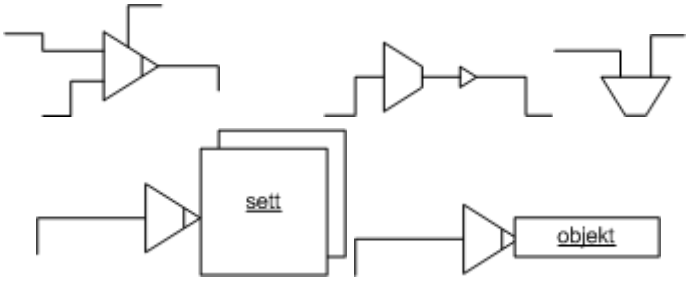
Informasjonsoverføringen som ivaretas av interaktorer og interaktorstrukturer defineres av: ”(1) Et sett porter som definerer interaktorenes eksterne grensesnitt til systemet, brukeren og andre interaktorer; (2) et sett koblinger som bærer data mellom portene; og (3) en kontrollstruktur som utløser eller forhindrer dataflyt mellom porter på interaktorens inn- og utside og langs koblinger.” [4]



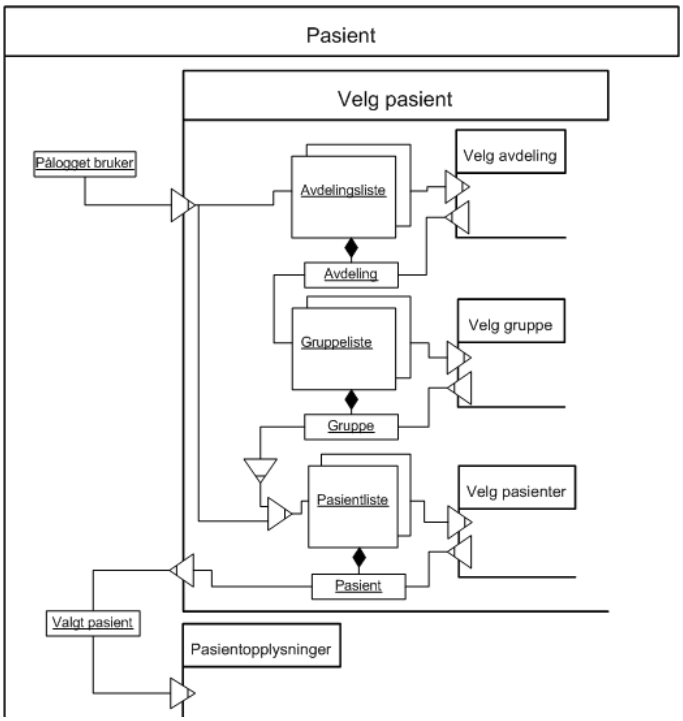
Figur 2: Interaktor, porter, koblinger og kontrollstruktur

En spesialisert variant av en port er den generiske funksjonen som benyttes når det er nødvendig å beregne nye verdier. Mer kompleks oppførsel enn beregning av en enkelt verdi indikeres ved at tuppen og basen til funksjonssymbolet er atskilt. [4] En metode er en funksjon som virker direkte på et attributt til det objektet eller settet den peker på. Metoder indikeres ved at tuppen på en funksjon leder bort til kanten på et objekt eller sett.

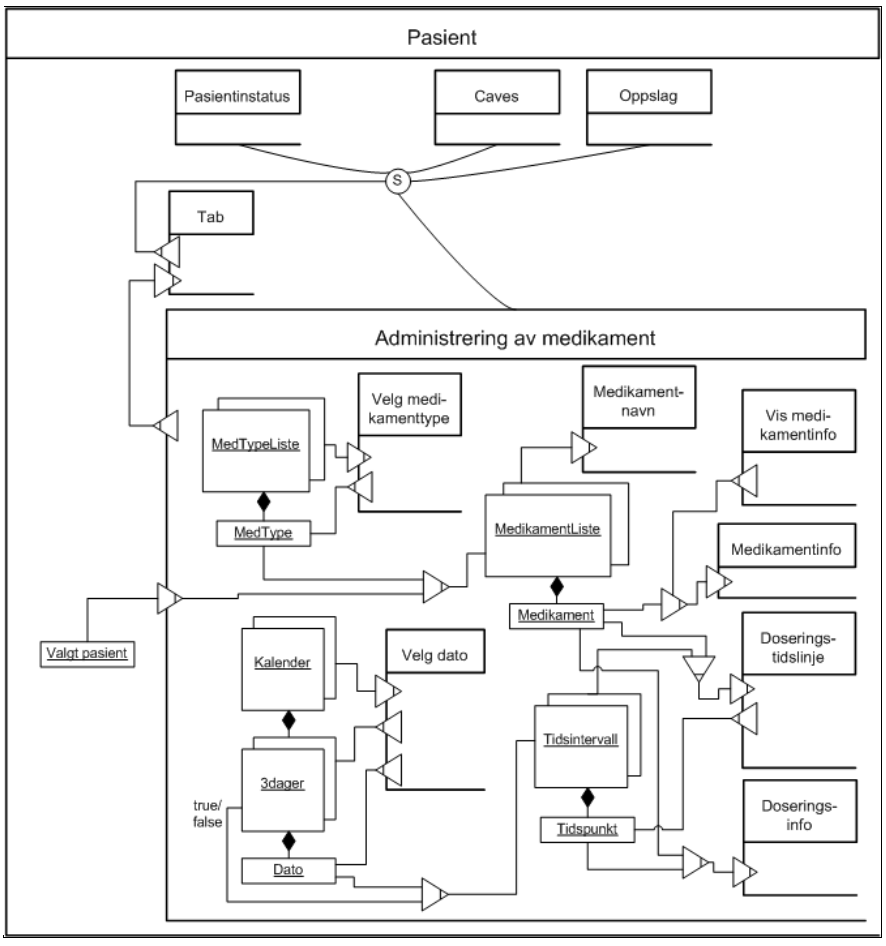
En kobling som leder til en port eller funksjons bakside, gir denne et argument. Koblingen som leder fra tuppen på porten eller funksjonen tar med et ferdigbehandlet resultat videre. En kobling som leder går inn i siden til en funksjon, fungerer som en trigger.



Figur 3: Eksempler på en funksjon med trigger, to komplekse funksjoner og to metoder.



Figur 4: DiaMODL-diagram av viewet ”velg pasient” i GUIMED

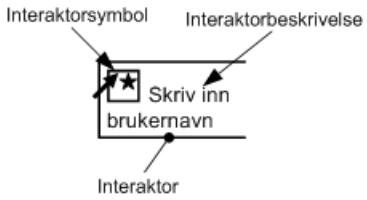


Figur 5: DiaMODL-diagram av viewet ”administrering av medikament” i GUIMED

ProtoMODL

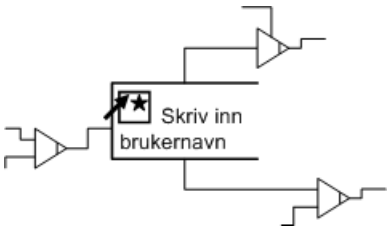
ProtoMODL er et mer uformelt verktøy for å beskrive interaksjon mellom de komponentene som er identifisert i en CAP enn DiaMODL. Notasjonen er basert på en forenklet versjon av DiaMODL, med to viktige forskjeller.

Den ene forskjellen er at i stedet for å beskrive kontrollstrukturen eksplisitt, benyttes symbolene fra CAP-notasjonen til å beskrive interaktorenes funksjonalitet. En kort beskrivelse spesifiserer brukerens interaksjon med komponenten, typisk ”Velg pasient” eller ”Skriv inn brukernavn”. Interaktorer uten en aksjon i beskrivelsen, for eksempel ”Pålogget bruker” eller ”Pasientopplysninger” viser at denne interaktoren kun er ment å vise informasjon til brukeren.



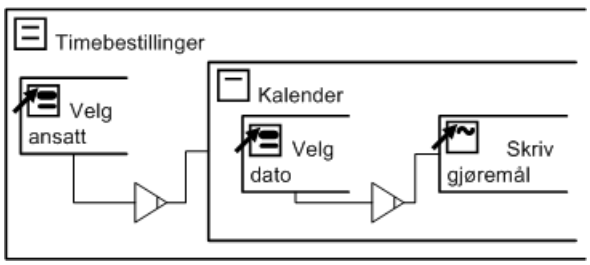
Figur 6: Interaktorbegrepet i ProtoMODL

I tillegg er portbegrepet fra DiaMODL abstrahert vekk. I stedet er koblingene festet direkte til interaktoren. Koblinger leder til interaktorens venstre side, viser hvilke inn-verdier som går til interaktoren, koblinger som er festet på interaktorens over- eller underside viser hvilke ut-verdier som gåtr fra interaktoren. Funksjonsbegrepet er det samme som i DiaMODL.



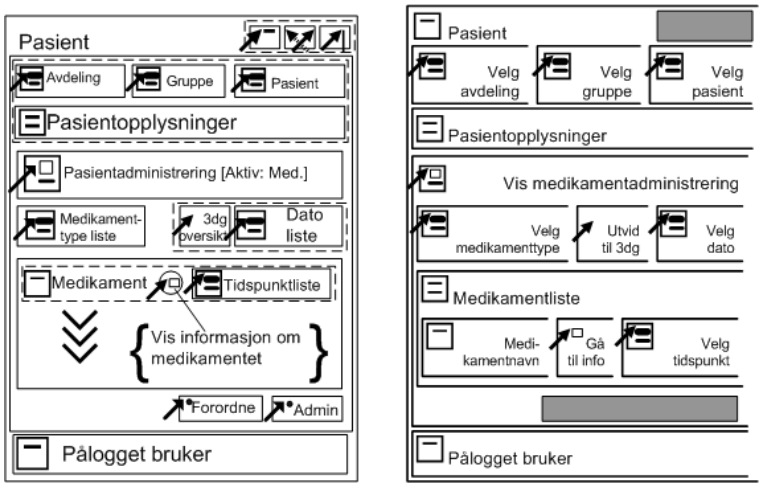
Figur 7: Koblinger og funksjoner i ProtoMODL

En nøstet interaktor, arver den utenforliggende interaktorens kunnskap om attributter og funksjoner. Det er derfor ikke nødvendig å legge inn koblinger mellom slike interaktorer.



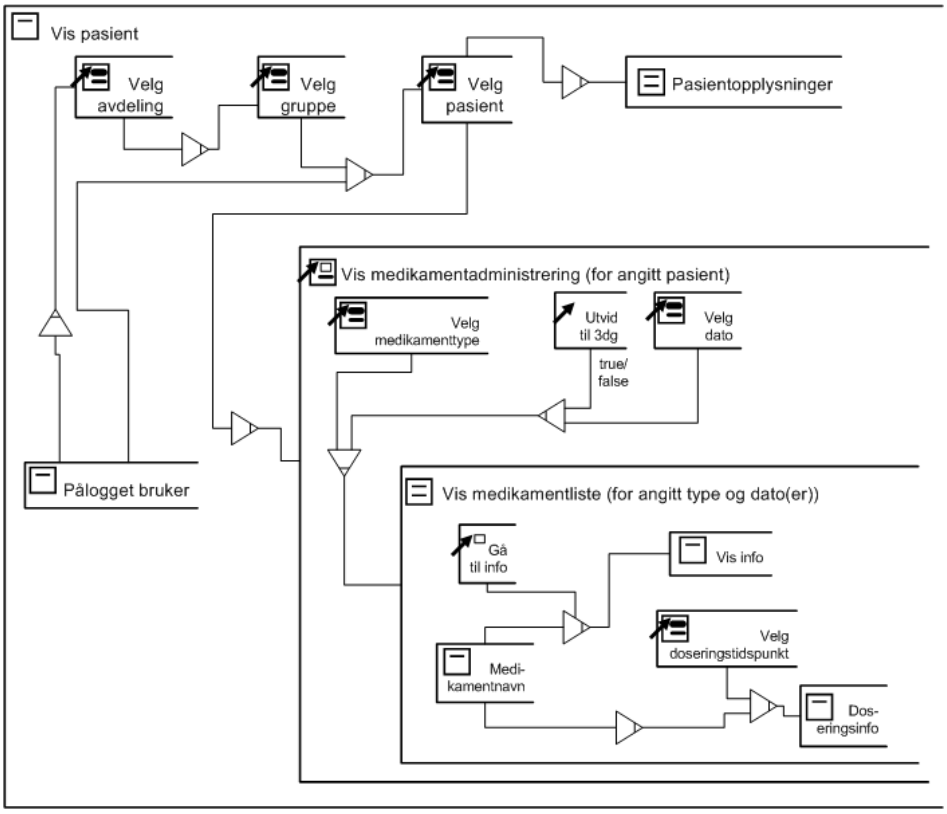
Figur 7: Nøstede interaktorer i ProtoMODL

For å produsere en ProtoMODL fra et skjermbilde eller en prototype, er det lettest å først lage en CAP. Ved å beskrive brukergrensesnittet på denne måten, identifiseres også komponentene som ligger der.



Figur 8: Identifisering av ProtoMODL-interaktorer fra CAP

Deretter blåses prototypen opp, og man kobler interaktorene sammen slik man mener interaksjonen mellom dem foregår.



Figur 9: ProtoMODL-diagram basert på CAP

Referanser:

1.Constantine, L. et al.: “From Abstraction to Realization in User Interface Designs: Abstract Prototypes Based on Canonical Abstract Components REVISED”, Juli 2003

2. Constantine, L. et al.: “PREPRINT Canonical Abstract Prototypes for Abstract Visual and Interaction Design”, revidert utgave august 2003

3. Campos P. F og Nunes N. J.: “CanonSketch: a User-Centered Tool for Canonical Abstract Prototyping”, i “Proceedings of DSV-IS'2004, 11th International Workshop on Design, Specification and Verification of Interactive Systems”, Springer-Verlag, 2004.

4. Trætteberg, H: “Model-based User Interface Design”, NTNU, 2002

Vedlegg 2

Semesteroppgave i faget MNFIT 364, Systemutvikling, organisasjon og arbeidsliv, 20.06 2004

Denne oppgaven er referert til i oppgaven, men var ikke tilgjengelig på nett pr. 01.06 2006.

Rational Unified Prosess

- et godt rammeverk for design av brukergrensesnitt?

Anders E. Bråten

Studnr 648598

Sammendrag

Denne oppgaven har til hensikt å undersøke hvorvidt RUP egner seg som rammeverk for design av brukergrensesnitt. Problemstillingene rundt brukergrensesnittdesignaktiviteten blir sett i lys av produktorientert- og prosessorientert design, og måten RUP løser disse utfordringene på sammenliknes med metodikken knyttet til tilsvarende aktivitet i systemutviklingsprosessen XP.

Først beskrives bakgrunnen for oppgaven. Deretter presenteres RUP som prosessrammeverk generelt, og som ramme for brukergrensesnittdesign spesielt. Videre kommer en kort presentasjon av produktorientert design slik den framstod i metoden Strukturert Analyse, samt filosofien bak prosessorientering og metodikken i Ekstremprogrammering. Til slutt diskuteres forskjellene og likhetene mellom tilnærmingene i lys av det politiske grunnlaget, verktøystøtten og holdning til brukersentrering.

Designere er i denne oppgaven brukt om både systemutviklere som designere av et system, og brukergrensesnittdesignere.

Innholdsfortegnelse

Innledning

I min hovedfagsoppgave har jeg konsentrert jeg meg om problematikken rundt det å modellere brukergrensesnitt, og da spesielt interaksjonen innad i grensesnittet. Bakgrunnen for valget av dette temaet kommer av min erfaring fra utdanning og arbeidsliv der formalismen rundt design av brukergrensesnitt ofte blir neglisjert eller tillagt liten betydning. Dette til tross for at undersøkelser har vist at 48% av kildekode og 50% av utviklingstid går med til nettopp utvikling av brukergrensesnittet [8]. Selv om disse tallene kan ha gått ned det siste tiåret pga økt kunnskap på området og bedre verktøystøtte, er det ingen tvil om at brukergrensesnittdesign fortsatt krever store ressurser.

Hvorfor blir så designen av brukergrensesnitt fortsatt nedprioritert? Er det fordi det er lagt liten vekt på denne biten i prosessrammeverkene som benyttes, eller er det måten man bruker rammeverkene på som fører til dette? Eller er det rett og slett en holdning i bransjen som henger igjen fra tidligere?

Fra før kjente jeg til Rational Unified Process, et rammeverk som de siste årene har fått stor utbredelse og blir sett på som en de facto standard i mange utviklingsmiljøer. [12] For å prøve å finne ut av problematikken rundt utvikling av brukergrensesnittdesign og dets rolle i system-utviklingsprosessen, var det derfor et naturlig utgangspunkt å se på hvilke holdninger RUP har til problemstillingene knyttet til (1) politikk i utviklingsmiljøet, (2) verktøystøtte og (3) bruker-medvirkning og brukbarhet. For å kunne si noe om kvaliteten til prosessen var det videre nødvendig å sammenlikne resultatene jeg fant med andre viktige systemutviklingstilnærminger som har vært og er i bruk.

Rational Unified Process

RUP er en kontrollert iterativ programvareutviklingsprosess basert på det forfatterne mener er bransjens beste tradisjoner, og har som mål å dekke hele livssyklusen til et utviklingsprosjekt. RUP tar ikke bare for seg det tekniske aspektet rundt utvikling av programvaresystemer, men søker også å ivareta det administrative arbeidet rundt prosessen ved å sørge for en disiplinert tilnærming til det å fordele oppgaver og ansvar i en utviklingsorganisasjon. Dens mål er å kvalitetssikre produksjon av høykvalitetsprogramvare som møter kravene og behovene til sluttbrukeren innenfor forutsigbare tidsplaner og budsjett. [10]

Historisk bakgrunn

I 1995 startet Rational Software Corporation arbeidet med å utvikle et nytt prosessrammeverk for utvikling av programvare. I sitt arbeid med kunder og partnere hadde Rational opparbeidet mye uformell kunnskap om gode, etablerte løsninger i bruk innen programvareutviklingsindustrien; iterativ utvikling, arkitektursentrering, visualisering vha modeller og use-case drevne prosjekter var stikkordene som ble lagt til grunn for prosessen. Det ble lagt sterk fokus på behovene til prosessbrukeren bl.a. ved tett integrering mot utviklingsverktøy. [14]. Det var i tillegg viktig at det tekniske aspektet var forankret i en solid prosjektstyringsprosess, slik at risikoer ble adressert tidlig, at prosjektene ble utført kostnadseffektivt og at de anliggende objektivene ble møtt. [10]

I årene 1996 til 1999 ble det jobbet med å dokumentere den kunnskapen Rational satt på i en konsistent kunnskapsbase [14]. Fra før brukte Rational en intern prosess kalt Rational Approach som fokuserte på iterativitet og arkitektur. Ved oppkjøp hadde de også fått tilgang til bl.a Ivar Jacobsons Objectory Process, hvor de hentet prosessmodellen og sentreringen mot use cases, og andre prosesser som omhandlet kravhåndtering, prosjektstyring og testing. Den første versjonen ble kalt Rational Objectory Process og var den første metoden som aktivt tok i bruk det nyutviklede Unified Modeling Language som senere har blitt en standard i systemutviklingsindustrien.

I 1998 skiftet prosessen navn til Rational Unified Process, etter at det hadde blitt styrket på områdene systemutvikling, web, businessmodellering og prosjektledelse, og i 1999 oppnådde man målet om et prosessrammeverk som kunne dekke hele livssyklusen til et prosjekt. I 2000 ble det lagt ved et konseptskriv som gir en introduksjon til brukersentrert design, men uten at resultatet er fullt ut integrert med metodikken i RUP. [12]

RUP ble raskt en av de mest brukte prosessene for iterativ og komponentbasert utvikling. Suksessen ble for en stor del forklart med at RUP var webbasert, enkel i bruk og lite påtrengende. [14]

Hvordan fungerer RUP.

En prosess beskriver hvem som gjør hva, og hvordan og når det gjøres. RUP representerer dette ved hjelp av fire modellelementer: Roller, artefakter, aktiviteter og arbeidsflyt. Rollene definerer oppførsel og ansvar til et individ eller en gruppe av individer i et team. Oppførselen til hver rolle uttrykkes i form av aktiviteter som rolleinnehaverne utfører. Ansvaret til hver rolle uttrykkes i forhold til bestemte artefakter som rolleinnehaverne lager, modifiserer eller kontrollerer. Arbeidsflyten beskriver en sekvens av aktiviteter som ender i produksjon av et observerbart resultat.

RUP opererer langs to dimensjoner

Prosesstrukturen i RUP har to dimensjoner; prosjektets livssyklus og prosjektets arbeidsflyt. [10] Et prosjekts livssyklus er delt inn i fire faser; diagnostiserings-, utviklings-, pilot- og utrulleringsfasen. Fasene er orientert i tidsaksen og hver av dem utføres gjennom flere iterasjoner. I diagnostiseringsfasen prøver man å forstå de generelle kravene og bestemme utvikingsområdet. I utviklingsfasen fokuseres det på kravhåndtering, arkitekturprototyping og minimering av teknisk risiko ved å prøve ut forskjellige løsninger og opplæring i verktøy og teknikker. Pilotfasen fokuserer på design og implementasjon, og avsluttes med evolusjonær utvikling av initielle prototyper til det første operasjonelle produkt. Og i utrulleringsfasen sikres det at systemet har høy nok kvalitet i henhold til kravene, bugs fikses, brukere får opplæring, moduler justeres, savnede elementer blir lagt til og det igangsattes levering av det ferdige produkt til kunden.

Arbeidet innenfor hver fase utføres i en kjerneprosessflyt og en kjernestøtteflyt. Disse deles så videre inn i aktiviteter, henholdsvis foretningsmodellering, kravspesifisering, analyse og design, implementasjon, testing og utrullering for kjerneprosessflyten, og styringskonfigurering, prosjektstyring og miljø for kjernestøtteflyten. Den relative betoningen på de forskjellige aktivitetene varierer etter hvert som de forskjellige fasene gjennomføres.

RUP forsøker å fange opp de beste tradisjonene fra systemutviklingsbransjen

RUP bygger på de erfaringer forfatterne hadde opparbeidet om de faktorene som oftest ga gode resultater i systemutviklingsprosjekter; iterativ utvikling, fokus på kravhåndtering, en sterk og robust arkitektur, visualisering av systemet vha modellering og vektlegging programvarekvalitet. [10]

Iterativ utvikling av programvare gir mulighet for å ta hensyn til endrede krav underveis, gradvis integrering og økt gjenbruk av programvareelementer, enklere og tidligere risikoberegning, mer robust arkitektur og bedre ressursfordeling [10]. I RUP er den iterative tilnærmingsmåten strengt kontrollert ved at iterasjoner er planlagt i antall, lengde og målsetninger. Oppgaver og ansvar til deltakerene er definert. Objektive mål av utviklingen blirt tatt underveis. Endringer i planene mellom iterasjonene blir nøye kontrollert

Et programvareintensivt system er dynamisk. Det vil si at man må være forberedt på at de endres i løpet av et prosjekts livssyklus. Med unntak av de mest trivielle systemer, er det umulig å fastsette komplette krav før den faktiske utviklingen har startet. RUPs kravhåndtering er ment å sikre en systematisk tilnærming til å finne, organisere, kommunisere og håndtere endrede krav til et programvareintensivt system eller applikasjon. Målet er å redusere prosjektkostnader og -forsinkelser, i tillegg til å øke programvarekvalitet, øke kundetilfredshet, og bedre gruppekommunikasjonen. Tidlig brukermedvirkning skal bidra til at programvaren møter kundens krav og behov, og gi alle interessenter en felles forståelse for hva som skal bygges og testes. [10]

Designaktivitetene i RUP er sentrert rundt begrepet arkitektur og bruk av komponentbaserte arkitekturer [10]. Arkitekturen skal gi grunnlag for en felles forståelse og en effektiv kommunikasjon mellom de forskjellige interessentene ved å etablere et felles grunnlag av referanser og vokabular for diskusjon av designspørsmål. Arkitekturbygging er en del av designprosessen; det handler om å ta avgjørelser om hvordan systemet skal bygges. Hovedfokuset i tidlige iterasjoner er å produsere og validere en programvarearkitektur. I den initielle utviklingssyklusen tar denne form av en eksekuterbar strukturell prototype som gradvis utvikler seg evolusjonært til et ferdig system i senere iterasjoner.

En stor del av RUP er sentrert rundt modellering av systemet som skal utvikles. [10] Modellene representerer en forenkling av virkeligheten og forsøker å fullt ut beskrive systemet fra et spesifikt perspektiv. Av denne grunn er det nødvendig med forskjellige modeller for å forstå og forme forskjellige problemstillinger og løsninger. Det er essensielt at disse modellene er koordinert for å sikre konsistens og hindre overflødighet. RUP skal sørge for veiledning i effektiv bruk av modellering, ved å beskrive hvilke modeller man trenger, hvorfor man trenger dem og hvordan man skal konstruere dem. RUP er konstruert for å kunne modifisere, justere og utvide og legge til nye modeller for å passe til en organisasjons spesielle behov, karakteristikk, rammer, kultur og historie. Modelleringsspråket UML benyttes for å visualisere, spesifisere, konstruere, dokumentere og vedlikeholde artefakter som inngår i et programvareintensivt system. Ideer og løsninger kommuniseres dermed utvetydig og øker interessentenes evne til å håndtere programvarekompleksitet.

Kostnadene knyttet til feil i programvaren er 100 til 1000 ganger dyrere å finne og reparere etter at systemet er tatt i bruk, enn dersom de hadde blitt oppdaget underveis i utviklingsløpet. [10] Programvarekvaliteten skal i RUP sikres ved at testing med hensyn på dets funksjonalitet, pålitelighet og ytelse utføres ved hver iterasjon, dvs en kontinuerlig, kvantitativ prosess. I RUP er programvarekvalitet ansvaret til hele organisasjonen, derfor defineres ingen enkeltrolle som ansvarlig for å sikre kvaliteten i et utviklingsprosjekt. I stedet for å fokusere entydig på

produktkvalitet og prosesskvalitet i programvareutvikling, fokuserer RUP på å kontinuerlig verifisere og objektivt aksessere hvorvidt produktet møter det forventede kvalitetmålet [10].

RUP har et 4+1 arkitektonisk view

Et arkitektonisk view er en abstraksjon av et system fra et spesielt perspektiv, og søker å ta for seg de viktigste, signifikante elementene ved en modell. For hvert view trengs en klar identifisering av synsvinkel; dvs interessenter og deres krav, elementer som må representeres i viewet og deres relasjoner, de organisatoriske prinsipper brukt for å strukturere viewet, hvordan elementene relaterer til elementer i andre view og den beste prosessen å bruke for å lage viewet.

RUP foreslår bruk av fem views: [10] Logisk view, realiseringsview, prosessview, utrulleringsview og Use-Case view. Logisk view adresserer de funksjonelle krav knyttet til systemet, dvs hva systemet faktisk skal gjøre for sluttbrukeren. Realiseringsviewet beskriver organiseringen av statiske programvaremoduler innenfor utviklingsmiljøet, f.eks kildekode og bibliotek. Prosessviewet tar for seg de sammenfallende aspekter ved systemet ved kjøring, - oppgaver, tråder og prosesser og deres interaksjoner, f.eks feiltoleranse, responstid og skalabilitet. Utplasseringsviewet viser hvordan eksekverbare og kjørbare komponenter passer i forhold til den underliggende plattformen. Det omhandler bl.a. utplassering, installering og utførelse. Use-case viewet inneholder scenarioer som brukes for validering av de andre viewene, og illustrerer hvordan de andre viewene fungerer i arkitekturen .

Brukergrensesnittdesign i RUP.

RUP definerer en prosjektrolle, brukergrensesnittdesigneren, som ansvarlig for å skape det visuelle designet til et brukergrensesnitt. [11] Denne rollen er involvert i to aktiviteter: brukergrensesnitt-tmodellering, representert ved use-case storyboards, sekvensdiagrammer, kollaborasjonsdiagrammer og state charts, og brukergrensesnittprototyping.

Det er ofte vanskelig å se ut fra en objektorientert modell hvordan et system utfører sine pålagte oppgaver. RUP har en use-casedrevet innfallsvinkel, hvilket betyr at et systems definerte use-cases er selve fundamentet for resten av utviklingsprosessen. Use-cases brukes for inspirasjon i design-fasen og for å definere oppførselen til et system, og sørger i tillegg for kontinuitet mellom system-krav og andre artefakter som designdokumenter og testdokumenter. Sekvensdiagrammer, kollaborasjonsdiagrammer og state charts er i virkeligheten abstraksjoner av use-case modellen og gir bare et annet perspektiv på det som beskrives der.

Hvilke prototyper som blir brukt sier noe om hva de forskjellige modellene legger vekt på. I RUP defineres fire forskjellige varianter av prototyper: oppførende prototyper brukes for å utforske oppførselen til et system fra brukerens perspektiv, strukturelle prototyper brukes til å evaluere systemets

arkitektur, utforskende prototyper brukes for ren eksperimentering og evolusjonære prototyper introduserer gradvis utvikling mot det ferdige produktet.

Fra RUPs perspektiv er det den strukturelle prototypen som er viktigst, siden hensikten med denne prototypen er å bekrefte at den foreslåtte løsningen kan realiseres. Oppførende prototyping anerkjennes mest som en aktivitet i kravspesifiseringsaktiviteten, med den hensikt å finne kravene til systemet. Det legges vekt på at utviklingen av selve systemet skal være basert på evolusjonær prototyping, hvor hver iterasjon ender i en eksekuterbar prototype som kan bli målt, testet og evaluert mot systemets krav. [12]

Oppsummering

RUP gir støtte for kontrollerte og iterative arbeidsprosesser, og nøyaktig prosjektstyring på både tekniske og administrative områder. Overordnet beslutningsmyndighet ligger i stor grad hos prosjektledelsen, mens designere har innflytelse over utviklingsforløpet. Brukere har liten innflytelse utover innhenting av krav. Prosessrammeverket har verktøystøtte på de fleste områder, og formalismen er ivaretatt gjennom standardisert dokumentasjon. Fleksibiliteten er stor i forhold til hvilke verktøy man ønsker å benytte både på modellerings- og prosesstyringssiden.

RUP legger liten vekt på brukermedvirkning i sitt prosessrammeverk. Brukbarhet og brukeres anledning til å påvirke systemet dekkes bare i de tidlige iterasjonene i forbindelse med innhenting av krav og den initielle designen. Brukere blir ikke tildelt en aktiv designrolle i de videre iterasjonene.

Andre tilnærminger

Utvikling av programvaresystemer har tradisjonelt vært et håndverk i stadig forandring. I denne delen presenteres kort noen forskjellige tilnærminger til design og utvikling som har vært og er i bruk, og hvordan disse har forsøkt å løse problemene rundt programvareutvikling og brukergrensesnittdesign.

Produktorientert design; strukturert analyse

Strukturert analyse ble introdusert i slutten av syttiårene, og tilhører en familie designmetoder som går under navnet funksjonell analyse. Prosessen bærer sterkt preg av produktorientering, og organisasjonen blir betraktet rent funksjonelt, i tråd med Fredrik Winslows Taylors vitenskapelige ledelsesteori. Produktorienteringen er ment å gjøre det enklere å utvikle kompleks programvare i store grupper, ved å øke standardiseringen og å innføre en større grad av spesialisering, samt å overføre kontrollen av utviklingen fra programmererne til ledelsen. Dette medfører en introduksjon av programmeringsstandarder, ettersyn av kode, strukturerte gjennomganger og diverse målemetoder. [2] Kravspesifisering og dokumentasjon gir alle medlemmene i gruppene et felles utgangspunkt.

Tankegangen bak strukturert analyse er at et komplekst problem best løses når det deles opp i mindre problemer som hver for seg kan løses selvstendig og implisitt. Metoden går ut på å å utvikle programvare etter formaliserte prosedyrer med utgangspunkt i en abstrakt spesifikasjon. [1] Først modelleres organisasjonen og arbeidsprosessene i form av informasjonsprosesseringssystemer, der dataflyten er det viktigste. Disse modellerer systemet ved å beskrive flyten av data mellom systemets forskjellige komponenter. Dataflytdiagrammene er ment å hjelpe brukere og designere i kommunikasjonen, siden de er grafiske og ikke tekniske. Etter hvert har flere verktøy som ER-diagram og tilstandsovergangdiagrammer blitt lagt til metoden. Deretter skiller man mellom de fysiske og de logiske aspektene ved systemet. [3]

Utvikling av programvare skjer etter fossefallsmetoden, hvor utviklingen utføres i en rett linje fra krav, via analyse, design og implementasjon, til testing. Hver fase blir dokumentert for seg, og inngår som en selvstendig del av sluttdokumentasjonen. [6] Utvikingen av systemet blir betraktet lineært, med ordnede og avsluttende faser, inntil selve systemet kan implementeres til slutt. Programvaren betraktes som et produkt for seg selv. Man antar at konteksten programvaren skal brukes i er veldefinert og at programvaren kan spesifiseres på forhånd. [1]

Innbygget i strukturert analyse er troen på en instrumentell rasjonalitet. Designprosessen blir hovedsakelig sett på som en problemløsingsaktivitet. Den største konsekvensen av denne tanke-

gangen er at mennesker blir gjort om til systemkomponenter i systemet. Det blir ikke gjort forskjell på hvordan mennesker og maskiner fungerer. Arbeidsprosessene blir betraktet som rene prosedyrer det er mulig å bryte ned i et fast sekvensmønster.

Strukturert analyse legger ikke opp til brukermedvirkning i noen større grad. Brukere kan gi tilbakemelding på designerens forslag og fungere som kilde til informasjon, men kommunikasjon med brukere for å forstå deres behov blir ikke regnet som en vesentlig del av designprosessen.

Prosessorientert design

Systemer med høyt innslag av brukerinteraksjon kombinert med kompliserte arbeidsprosesser og samfunnets stadig økende avhengighet av dataressurser, førte til at man etter hvert så mer og mer på informasjonssystemer i kontekst med både utviklingen og bruken av den. [1] På slutten av åttitallet kom det en ny tilnærming til design av programvaresystemer, som gikk under betegnelsen prosess-orientering. Dette er en overordnet filosofi som ligger til grunn for mange av de senere metodene som har sprunget fram de siste årene, som f.eks 'Usage-centered Design'(1996) og 'Rapid Application Development'(1991).

Den prosessorienterte retningen betrakter programvaren både utviklet og brukt i sammenheng med menneskelig læring, arbeid og kommunikasjon, [1] og det faktiske produkt er et resultat av de sammenvevde prosesser i analyse, design, implementasjon og evaluering. Et av de viktigste prosess-elementene i prosessorientert utvikling er betydningen av brukermedvirkning, og brukeren tildeles en aktiv rolle gjennom hele utviklingsløpet. [5]

Felles for metodene som baserer seg på prosessorientert design, er at utvikling av programvare-systemer betraktes om en iterativ, syklisk prosess basert på analyse, design, implementasjon og testing. Formaliserte modeller og dokumentasjon er viktige for å kvalitetssikre kontroll over utviklingsforløpet selv ved kraftige endringer av designet, men man står fritt i å velge verktøyene selv. Det er prosessen som er viktig, ikke virkemidlene. Beslutningsmyndigheten ligger hos alle interessenter i fellesskap, hvilket vil si at kontrollen over utviklingsløpet er delt mellom ledelse, designere og sluttbrukere.

Hver syklus leder til en prototype som evalueres i sammenheng med arbeidsprosessen for å luke vekk rene systemfeil, feil ved designen og arkitekturen og mangelfulle eller feilaktige bruker-grensesnitt underveis i utviklingsforløpet. Den sykliske basismodellen åpner for utstrakt bruk av prototyping, noe som gir rom for en utforskende, eksperimentell og evolusjonær tankegang.

Prosessorientering som retningsgiver for design av programvaresystemer, kombinerer produkt-orienteringens krav til formalisme med sykliske utviklingsforløp. Dette gjør det mulig å utvikle

informasjonssystemer i store grupper, samtidig som gjentatte iterasjoner og prototyper skal ivareta brukermedvirkning underveis.

Ekstremprogrammering - XP

XP ble lansert som utviklingsmetode i 1996 av Kent Beck, som ønsket seg en enklere og mer effektiv tilnærming til utvikling av informasjonssystemer. Ved å konsentrerte seg om å skille ut de elementene som gjorde utvikling av programvare enkelt og hva som gjorde det vanskelig, kom han fram til fire essensielle måter for å forbedre et utviklingsprosjekt: Kommunikasjon, enkelhet, tilbakemelding og vågemot.

XP har mange fellestrekk med prosessorientert design, men henter samtidig inspirasjon fra hacker-bevegelsen. Hackere mener at programmering er et håndverk og at kvalitet innen systemutvikling er forankret i individuelle erfaringer og egenskaper, og ikke i regulering. [2] Åpne systemer hvor kildekoden er fritt tilgjengelig, integrerte løsninger og distribuerte systemer er sentrale elementer i denne filosofien, og hurtig prototyping, sterk brukermedvirkning og mangel på formalisme preger arbeidsmetodene.

Kontinuerlig kommunikasjon mellom brukere og designere ligger i bunn for XP. For å samordne designteamet og utveksle designideer benyttes CRC (Class, Responsibilities, and Collaboration)-kort. Programvaren utvikles i form av prototyper og testes helt fra start. Informasjonssystemet skal leveres til kunden så tidlig som mulig, og foreslåtte endringer blir implementert fortløpende. Metoden oppfordrer til at programmerere og kunder skal våge seg på nye ideer og ta sjanser, og gå nye veier i utviklingsprosessen.

XP virker kaotisk, men har en disiplinert tilnærming mot programvareutvikling. Ved design følger man et enkelt utviklingsløp gjennom systematisk testing og design av forbedringer. Selve metoden er basert på 12 prinsipper [10]: Grundig planlegging, små utgivelser, enkle metaforer, enkel design, kontinuerlig testing, reprodusering i stedet for feiloppretting, parprogrammering, kollektivt eierskap, kontinuerlig integrering og distribuering, 40 timers arbeidsuke, reell kundeinnflytelse og standardisert kode.

Metoden baserer seg på å bli brukt i små grupper av designere som trenger å utvikle programvare raskt i et miljø hvor kravene og spesifikasjonene raskt kan bli endret, selv sent i utviklingsløpet. XP har stor tilpasningsevne i forhold til skiftende krav, og utviklingsprosessen kommer raskt i gang. Kommunikasjonsveiene mellom medlemmene i gruppen er korte og parprogrammering gir effektiv opplæring. Ved å ha kunden tilstede får man raskt tilbakemeldinger. Parprogrammering, korte iterasjoner og kontinuerlig testing skal sikre god kvalitet på koden.

Oppsummering

Strukturert analyse er en metode som kjennetegnes av sterk styring og sterk formalisme, men liten grad av brukermedvirkning. Utvikling skjer etter fossefallsmetoden og med klare krav til dokumentasjon og formalisme, noe som gir en enkel oppskrift på hvordan utviklingsløpet utføres i tid, men som gir stor risiko for at feil og mangler i den initielle designen først blir oppdaget sent i utviklingsløpet, når de er tidskrevende og kostbare å rette opp. Overstyring fra ledelsens side kan også føre til problemer både med designere og brukere. Svak brukermedvirkning gir risiko for at brukbarheten og/eller brukeraksept til det endelige systemet kan være lav, siden designere sjelden vet eksakt hva brukere ønsker og trenger ut fra rene spesifikasjoner.

De prosessorienterte metodene kjennetegnes av en sterk grad av brukermedvirkning, sykliske utviklingsløp, iterativ prototyping og ansvarsdeling mellom alle interessenter i et utviklingsprosjekt. Dette krever at man benytter robuste rammeverk med sterk grad av formalisme rundt utvikling og prosjektstyring for å unngå misforståelser og forsinkelser. Manglende formalisme rundt verktøy-støtte krever at designerne holder seg oppdatert selv og er i stand til å velge rett verktøy til oppgavene som skal utføres. Motstridende interesser kan føre til at man må foreta avveininger underveis. Sterk grad av brukermedvirkning i alle ledd gir mindre risiko for lav brukbarhet og brukeraksept.

Ekstremprogrammering gir designere og brukere stor frihet og beslutningsmyndighet i forhold til utviklingsforløpet. Metoden vektlegger kreativitet, iterativ prototyping og uformell kommunikasjon på bekostning av dokumentasjon og formalisme. Metoden krever at man må jobbe i små, tette grupper for å samordne kode og formidle designløsninger. Mangel på formalisme kan føre til at det oppstår misforståelser mellom designere og brukere, og det kan være lett å gå seg vill i utviklings-løpet. Svært mye av dokumentasjonen befinner seg i hodet på interessentene. Brukbarheten til systemet avhenger sterkt av enkeltdesignere og deres evne til å kommunisere med brukeren og evne visualisere løsningene man kommer fram til i fellesskap.

Diskusjon

Et paradigmeskifte i utviklingsbransjen

De tradisjonelle, produktorienterte prosessene med dokumentsentrering, funksjonell analyse, utvikling etter fossefallsmetoden og fokus på behovene til ledelsen i stedet for å leveranse av brukbare produkter til kundene, viste seg utover åttitallet å være ineffektive på flere områder: [1] [14] Stadig raskere utvikling av ny teknologi gjorde det vanskeligere for designere å holde seg oppdatert på sine fagfelt. Det ble derfor nødvendig å fokusere på rammeverk som ga rask tilgang til veiledning og gode, etablerte bransjeløsninger i sine utviklingsverktøy. Krav om at informasjonssystemer skulle betraktes både i den konteksten de ble brukt, og læringseffekten i bruk og under utvikling, gjorde utviklingsprosessen mer komplisert når det gjaldt å spesifisere og analysere informasjonssystemene. Videre sørget standardisering av metoder, verktøy og prosesser det økonomisk forsvarlig å utvikle store prosessrammeverk ettersom utgiftene kunne fordeles på flere selskaper. Mange konsulentfirmaer ble etter hvert presset av markedet til å benytte kommersielle produkter som kunne vise til gode resultater, framfor egne, hjemmelagde prosessrammeverk. De fossefallsbaserte prosessene fokuserte mest på hva som måtte gjøres i stedet for hvordan, og hjalp derfor ikke designere å gjøre en bedre jobb. De hadde i tillegg store svakheter med å adressere risikoer tidlig i utviklingsløpet. På toppen av dett var prosessveiledningene ofte bare tilgjengelige i form av store dokumentsamlinger som var vanskelige å orientere seg i. [14]

Paradigmeskiftet fra produkt- over til prosessorientert design bante vei for utviklingen av nye prosessrammeverk på nittitallet. RUP og XP ble utviklet parallelt i dette miljøet preget av til dels radikale nyvinninger. I tillegg til prinsippet om iterativ og inkrementell utvikling og testing basert på evolusjonære prototyper som de hentet fra prosessorienteringen, baserte begge prosessene seg på de etablerte praksiser i bransjen som hadde vist seg å være effektive med hensyn på å levere prosjekter med høy brukeraksept og kvalitet innenfor fastsatte tidsrammer og budsjetter.

Prosessenes underliggende politiske avhengigheter

Den overordnede forskjellen på de to prosessene ligger i hvilke praksiser som er utelatt i forhold til den prosessorienterte filosofien; RUP har unnlatt å bygge inn brukersentrering til fordel for en sterk prosjektstyringsaktivitet som sikrer politisk makt til ledelsen. XP satser sterkt på brukersentrering og uformell kommunikasjon, på bekostning av formalisert dokumentasjon og prosjektledelse. På dette feltet etablerer derfor RUP og XP seg som to motpoler i bransjen; RUP foretrekkes av tunge selskaper med en sterk politisk agenda i sine utviklingsprosjekter, mens XP hovedsakelig benyttes av små, mellomstore og distribuerte selskaper og grupperinger som vektlegger kreativitet, uformell kommunikasjon og maktfordeling mellom alle interessenter involvert i utviklingen av et

programvaresystem. Mellom disse motpolene finner vi de fleste andre prosesser; oftest lettere metoder med større fokus på brukersentrering og mer desentralisert beslutningsmyndighet enn RUP, men med større krav til formalisme, dokumentasjon og rolledefinering enn XP.

Den demokratisk deltakende designtradisjonen beskrevet i [4] preges av maktfordeling mellom ledelse, designere og brukere, og deres respektive fagforeninger. Problematikken som beskrives i disse tilfellene av tidlig deltagende design, finner vi igjen i brukersentreringen som er grunnlaget for prosessorientert design. Med sterke fagforeninger som kan ivareta brukernes interesser, ville antakelig RUP kunne vært en bra ramme for disse utviklingsprosjektene, siden maktfordeling er et større tema enn brukersentrering. Fagforeningene ville i et slikt scenario kunne tatt en aktiv part som designere og på denne måten fått stor beslutningsmyndighet over utviklingsforløpet, mens ledelsen hadde kunnet beholdt kontrollen over selve prosjektstyringen.

I [5] beskrives en politisk situasjon der et programvaresystem vil endre den politiske maktbalansen mellom to brukergrupper, leger og sykepleiere. I denne situasjonen finner man klare motstridende interesser i forhold til bruken av programvaresystemet. Denne typen problemer er vanskelig å fange opp i et prosessrammeverk hvor fokus ligger på uniforme interessentgrupperinger. I RUP ville en grundig forundersøkelse av arbeidsmiljøet kanskje avdekket denne betente situasjonen i krav-innhentingsfasen. Dersom problemet derimot ikke ble oppdaget, ville i verste fall denne designfeilen blitt hengende igjen gjennom hele utviklingsløpet og ført til lav brukeraksept hos enten den ene eller den andre, eller begge brukergruppene, noe som ville ført til at programvaresystemet ikke kunne utføre oppgaven det var designet for, nemlig å administrere tidsbruken til brukerne. Hvis et slikt system ble designet ved hjelp av XP, ville antakelig den tette brukersentreringen avdekket problemet, dersom man på forhånd hadde klart å innse at det var to fundamentalt forskjellige brukergrupper involvert.

Prosessenes verktøystøtte og innebygget metodologi

Prosesser som har en streng formalisme med hensyn på hvilke verktøy som skal benyttes til hvilke oppgaver får ofte problemer med at brukerne av prosessen føler seg overkjørt. I [3] beskrives en situasjon der designere i de tre firmaene undersøkelsen omfattet, er pragmatiske i forhold til bruken av strukturert analyse. Fire tydelige avvik fra prosedyrene gjør seg gjeldende. (1) Designerne bruker ikke dataflytdiagram når de kommuniserer med brukere, men baserer seg i stedet på skjermbilder og prototyper, (2) dataflytdiagrammene viser ikke manuelle prosedyrer, (3) designerne reduserer antall modeller til en modell av det nye systemet, og det blir ikke skilt mellom fysiske og logiske dataflyt-diagrammer. og (4) de supplerer denne modellen med andre typer beskrivelser, diagrammer, eller prototyper. Prosessorientert design har tatt lærdom av dette og absolutte krav om bruk av spesifikke verktøy har mer eller mindre forsvunnet fra moderne designprosesser.

RUP er svært fleksibelt med hensyn pådesign verktøy. Prosessen har støtte for et stort utvalg integrerte programvareverktøy og UML-modeller som skal kunne dekke de fleste situasjoner. Designere har stor frihet i forhold til hvilke verktøy man vil benytte til de respektive oppgavene. Men denne fleksibiliteten kan i noen tilfeller være en belastning, siden prosesser som prøver å være alt ofte ender opp med å være ingenting. Et av hovedankepunktene mot RUP er at mange foretrekker en prosess som beskriver nøyaktig hva som skal gjøres heller enn å gi et utall forskjellige løsninger. [15]

RUP har et ensidig fokus på Use-cases i sin brukergrensesnittstrategi. Selv om Use-cases kan være kraftige verktøy for å modellere og dokumentere brukerkrav, er de ofte generelt for vage i sin definisjon og variert i bruken av til å være skikkelig effektive. Use-cases, eventuelt brukt sammen med statecharts, sekvens- og kollaborasjonsdiagrammer, blir lett overfladiske og har en tendens til å si mer om hva som blir utført enn hvordan. Gode interaksjonsmodeller med evne til å gå dypere inn i grensesnittet ville ha kunnet hjulpet en del på dette problemet.

Jacobsens originale betoning på bruk i sin Use-case teori har i RUP blitt flyttet over mot å bli mer systemsentrert, slik at fokuset er på hva systemet utfører av oppgaver og ikke på hva brukeren gjør eller ønsker å gjøre. Legger man dette argumentet sammen med det faktum at Use-cases sjelden blir skrevet av brukergrensesnitt- eller brukbarhetsdesignere, er det vanskelig å se hvordan Use-cases kan lede til gode spesifikasjoner sett fra et brukeropplevelsesperspektiv. [17] RUP legger opp til at prototyping i forbindelse med brukergrensesnittdesign hovedsakelig skal brukes til innhenting av krav, og forsterker bare manglende som kan oppstå ved Use-case modellering..

XP har svært få formelle verktøy å forholde seg. Designere bruker CRC-kort for å representere objekter og samordne design, men har ikke innlemmet noen støtte for mer analytiske og strukturerte metoder. [9] Størrelsen på prosjektet kan være en hindring. Å integrere kode er en stor utfordring og en opplagt flaskehals. Kommunikasjon mellom designerne blir problematisk i større grupper, og parprogrammering er en situasjon mange føler seg ukomfortable med. [8] Manglende dokumentasjon kan være en stor ulempe f.eks ved utskiftninger i designteamet, eller når andre designere overtar videreutviklingen av et prosjekt.

Brukergrensesnittdesignen i XP skjer hovedsakelig ved hjelp av evolusjonære prototyper utviklet i tett samarbeid med brukeren. Dette gir høy brukeraksept, men garanterer ikke brukbarheten siden abstrahering ofte er nødvendig for å avdekke mange feil i designen. Spesielt er mangelen på analysering av brukere og oppgaver, og spesifisering av brukbarhetsmål en stor risiko. XPs manglede formalisme og fokus på hurtige iterasjoner kan dermed føre til problemer i utviklingsløpet.

Prosessenes tilnærming til brukermedvirkning og brukbarhet

Produktorientert design åpnet ikke for brukersentrering. Dette gjorde at kommunikasjon med brukere bare var tilgjengelig i de aller tidligste fasene av utviklingsforløpet. Man fikk dermed ikke sjekket relevansen til spesifikasjonen, ikke ble spørsmål tilhørende bruken av programvaren avklart før programvaren var levert og brukergrensesnittet ble som regel konstruert etter at funksjonaliteten var bestemt, mest for å bedre brukermottakelsen. Ved å ta for gitt at problemer kunne løses implisitt, ignorerte man behovet for en utarbeidelse av designets mål som del av designprosessen. Disse åpenbare manglene var lenge ignorert, med den følge at både folk og organisasjoner har måttet tilpasse sine arbeidsvaner og kommunikasjonsmønstre etter programvaren. [1]

I Rationals designstrategi er ikke brukeren tildelt noen vesentlig rolle. I deres egne ord er RUP "en arkitektursentrert prosess" som definerer "en sekvens av aksjoner som et *system* utfører". Altså kan heller ikke RUP sees på som en brukersentrert designprosess. Det som utvikler seg gjennom iterasjonene er systemarkitekturen og og de forskjellige klassene som former basisen for det objekt-orienterte synet. [12] Ved å introdusere brukere i begynnelsen av prosjektet uten å gi de en aktiv rolle i utviklingsforløpet er det en stor risiko for at de skal bli glemt ettersom fokuset skifter mer og mer over på systemarkitekturen.

Manglende brukersentrering gjør at mange av de samme problemene som oppstod i produktorientert design kan finnes igjen i RUP, om enn litt svakere nedtonet siden iterativ utvikling gir mulighet for å rette opp mye underveis. Liten grad av brukermedvirkning gir risiko for både lav brukbarhet og liten brukeraksept ved utrullering av systemet. Brukergrensesnittdesigneren er den eneste definerte prosjektrollen involvert i problematikken rundt brukbarhet, men trenger ikke å ha nødvendig kompetanse innen området brukbarhet. [12]

XP kan være en krevende metode både for kunde og designer. Brukermedvirkning er essensielt under hele utviklingsforløpet ,og kunden må være villig til å ta en del av risikoen for prosjektet i tillegg til å tåle belastningen ved å måtte være tilgjengelig store deler av tiden. Designere må være villige til å lytte til kundens behov og ha evne til å inngå kompromisser når det oppstår motstridende interesser. Brukbarheten skal ivaretas ved hjelp av direkte kommunikasjon med brukerne og hurtige iterasjoner som ender i eksekuterbare prototyper, i stedet for analyser, modeller og spesifikasjoner. Dette gir ofte kreative løsninger og gode brukeropplevelser, men gir ingen garantier for at kvaliteten blir sikret.

Tegn i tiden

Mange har forsøkt å utnytte fleksibiliteten som er bakt inn i RUP til å modifisere prosessen mot en mer brukersentrert designprosess. En av disse er Robert Martins dX prosess. dX er en fullt ut kompatibel avlegger av RUP, som tilfeldigvis er nesten identisk med XP. dX er konstruert for folk som

ønsker å bruke XP, men som må bruke RUP, og er på denne måten et godt eksempel på fleksibel bruk av RUP. [15]

En annen tilnærming til brukersentrert RUP presenteres i [12], der Gulliksen utnytter fleksibiliteten som ligger i rolledefineringen. Den nye rollen brukbarhetsdesigner har ansvar for å holde prosessen brukersentrert ved å fokusere på brukbarhetsaspektene ved systemet. Denne rollen kan spesifisere brukbarhetsmål og designkriterier, utføre bruker- og oppgaveanalyser, få brukerbehov og -krav fram i lyset, designe brukergrensesnittet, eller i større prosjekter, lede designteamet og delta i evalueringen. For å unngå tap av viktig informasjon i overgangen mellom de forskjellige design-aktivitetene, er det viktig at brukbarhetsdesigneren deltar i alle brukersentrerte aktiviteter. Det er viktig at brukbarhetsdesigneren arbeider tett med brukerorganisasjonen og deltar i alle analyser relatert til brukbarhet.

Konklusjon: Et steg videre…

RUP er et prosessrammeverk som har tatt et stort steg videre i forhold til design av brukergrensesnitt når man sammenligner med produktorienterte tilnærminger. Verktøystøtten og den iterative utviklingsmetoden sammen med sterk prosjektstyring og god dokumentasjon danner et godt utgangspunkt for å designe stødige systemer.

Brukersentrert design har nærmest blitt en de facto standard for programvareutvikling etter paradigmeskiftet på begynnelsen av nittitallet. RUP skiller seg ut fra andre moderne prosesser og metoder ved at brukersentrert tilnærming er nærmest fraværende. Brukersentrert design prøver å styrke de kreative aspektene av brukergrensesnittdesign, og dette passer dårlig inn i den mer strukturerte, objektorienterte systemutviklingsprosessen som RUP er. [13] Sett i forhold til RUPs store markedsandeler, kan dette ha hatt avgjørende betydning når det gjelder den generelt lave prioriteringen av brukergrensesnittdesignaktiviteten i systemutviklingsbransjen. Selv om RUP siden 2000 har begynt å fokusere mer på dette området, er det langt igjen til det prosessorienterte idealet med full brukermedvirkning.

Maktforsyvningen mot prosjektledelse og designere i forhold til brukere, gir også et dårlig utgangs-punkt for å lage systemer med høy brukbarhet. XP er mye bedre med hensyn på brukersentrering, men det denne metoden tjener her, taper det i mangelen på analytiske metoder for avdekking av brukbarhetsmål og brukerbehov.

RUPs store fordel er den innebygde fleksibiliteten. Ved å innføre brukersentrering og brukbarhet som begrep i både rollebesetning, aktiviteter og artefakter, og bygge ut UML med bedre interaksjonsmodeller, har RUP potensial til å bli et fullverdig prosessrammeverk som også ivaretar brukernes interesser i designprosessen.

Referanser

1 Floyd, C.: “Outline of a paradigm change in software engineering”, s. 193-210, i *Computers and democracy*, Avebury, 1987.

2. Hannemyr, G.: “Technology and pleasure: hacking considered constructive”, *First Monday*, vol. 4:2, February, 1999

3. Bansler, J.P. og Bødker, K.: “A reappraisal of structured analysis: Design in an organizational context”, ACM Trans. on information systems, 11(2):165-193, 1993

4. Ehn, Pelle, “Scandinavian Design: On Participation and Skill”, s. 41-78 i D. Schuler og A. Namioka (eds): *Participatory Design: Principles and Practices* . Lawrence Erlbaum ass., 1993

5. Grønbæk, K. et al.: “Achieving Cooperative System Design: Shifting from a Product to a Process Focus” s. 79-98 i D. Schuler og A. Namioka (eds): *Participatory Design: Principles and Practices* . Lawrence Erlbaum ass., 1993

6. Nordfalk Jacob. “Objektorienteret analyse og design”. i *Objektorienteret programmering i Java*, Espergærde, Danmark, juli 2002,

7. Wagner, I.: “A Web of Fuzzy Problems: Confronting the Ethical issues” i Communication of the ACM, 36(4), 1993, s. 94-101

8. Myers, Brad A. "User Interface Software Tools." ACM Transactions on Computer-Human Interaction 2, 1 (March 1995): 64-108.

9. Wells D, “Extreme Programming: A gentle introduction”, på nettside http:// www.extremeprogramming.org/ ,1999

10. Kruchten, P.: *The Rational Unified Process An Introduction*, Second Edition, Addison-Wesley, juni 2001

11. Kruchten, P. et al.: "User Interface Design in the Rational Unified Process”, s. 161-196, i *Object Modeling and User Interface Design. Designing Interactive Systems*, Mark van Hamelen (ed), Addison-Wesley, Mars 2001

12. Gulliksen, J. et al.: “A User-Centerd Approach to Object-Oriented User Interface Design”, s. 283-312, i *Object Modeling and User Interface Design. Designing Interactive Systems*, Mark van Hamelen (ed), Addison-Wesley, Mars 2001

13. Hudson, W.: “Toward Unified Models in User-Centered and Object-Oriented Design”, s. 313-360, i *Object Modeling and User Interface Design. Designing Interactive Systems*, Mark van Hamelen (ed), Addison-Wesley, Mars 2001

14. Kroll, P.: “The RUP: An industry-wide platform for best practices”, på nettside *http://www-106.ibm.com/developerworks/rational/library/873.html* ,1999

15. Fowler, M.: “The New Methodology”, på nettside *http://www.martinfowler.com/articles/newMethodology.html*, 2003

16. Ukjent forfatter: “UI RUPture - Can Rational Unified Process really facilitate a better experience?”, på nettside *http://www.uidesign.net/2000/opinion/UIRupture.html*,

17. Constantine, L. og Lockwood, L.: “Structure and Style in Use Cases forUser Interface Design”, i *Object Modeling and User Interface Design. Designing Interactive Systems*, Mark van Hamelen (ed), Addison-Wesley og på *nettside http://www.foruse.com/articles/structurestyle2.pdf,* Mars 2001

18. Beaudouin-Lafon, M., Mackay, W.: "The Human-Computer Interaction Handbook: Fundamentals, Evolving Technologies and Emerging Applications", kapittel 52: "Prototyping Tools and Techniques", Erlbaum Associates, 2003