TILRETTELEGGING FOR PROTOTYPING

AV NY TEKNOLOGI

I EN UTVIKLINGSOMGIVELSE

En designers perspektiv

Hovedfagsoppgave i informatikk

Bjørn Sand Evjen

Systemarbeid og Menneske-maskin interaksjon

Forord

Materialet som presenteres her er resultat av et to års hovedfagsstudium ved Institutt for Datateknikk og Informasjonsvitenskap, NTNU. Denne hovedfagsoppgaven handler om hvordan man kan tilrettelegge for prototyping av ny teknologi i en utviklingsomgivelse. Målet med dette er å la designere prøve ut ny teknologi i en tidlig fase av systemutviklingsprosessen.

Jeg vil takke min veileder, Dag Svanæs, både for interessante tanker og innspill og for tilrettelegging av studiet.

Den praktiske delen av studiet har blitt gjort sammen med hovedfagsstudent Frank Werner Johansen. Samarbeidet med Frank har vært preget av spennende men også til tider frusterende utfordringer og konstruktivt teamarbeid med mange nyttige diskusjoner.

Mitt fokus har vært på hvordan man kan abstrahere teknologiens kompleksiteter og presentere teknologien i en begripelig innpakning. Abstraksjonen legger således grunnlaget for et grensesnitt som kan benyttes av folk som har begrenset innsikt i hvordan Bluetooth-teknologien fungerer. Jeg har konsentrert meg om konseptuelle sider ved teknologien og har lagt vekt på anvendelsesaspektet.

Han har fokusert på realiseringen av kommunikasjonen mellom en utviklingsomgivelse og Bluetooth-teknologien som vi presenterer i vår grensesnittløsning. Hans oppgave har vært rettet mot tekniske aspekter rundt det å legge til rette for ny teknologi, og hvordan man kan bruke et rammeverk til å abstrahere vekk vanskelige teknologiske begrep.

Det har vært nødvendig å samarbeide om den praktiske løsningen, da den tekniske løsningen for å abstrahere teknologien og det konseptuelle designet av grensesnittet henger nøye sammen. Da den praktiske delen av studiet har vært et samarbeid mellom Frank og meg, har deler av teorien noen likhetstrekk. Perspektivene er imidlertid tilpasset hver vår problemstilling.

Sist men ikke minst vil jeg rette en takk til min samboer, Siri, som har vært en viktig støtte gjennom hele hovedfaget og som har tatt seg av lille Leif på dagtid.

Del I: Teori

Del II: Komponent og test

Del III: Refleksjoner

Vedlegg:

Del I

Teori

1. Innledning

1.1 Bakgrunn

Når nye teknologier blir utviklet er de gjerne forbeholdt noen få, da stor kompleksitet kan innebære en for høy terskel til at andre enn fageksperter kan forstå teknologien fullt ut. For at en teknologi skal gjøres tilgjengelig til et større antall personer, kan det bli nødvendig å senke terskelen. Utfordringen blir da å forenkle teknologien og presentere den i en mer "spiselig" utgave.
En stor del av aktiviteten innen teknologisk utvikling kan sies å være relatert til forenkling av teknologisk kompleksitet. I dagens teknologiintensive samfunn er forenkling helt nødvendig for at teknologi skal oppnå en viss utbredelse. Det kan være konkurranse mellom flere lignende teknologer, og det trenger ikke være den mest teknisk optimale, men like gjerne den mest brukervennlige løsningen som vinner utbredelse.

Bakgrunnen for hovedfagsoppgaven var at jeg ville se på hvordan man kunne forenkle ny teknologi slik at den ble lettere tilgjengelig for ikke-eksperter. Jeg ville se om teknologien kunne presenteres på en begripelig måte i en utviklingsomgivelse slik at man raskt kunne designe funksjonelle prototyper. For at teknologien skulle være lettere tilgjengelig måtte kompleksiteten reduseres slik at terskelen for å prøve ut den nye teknologien ble senket.

1.2 Hensikten med hovedoppgaven

Jeg har ønsket å se på hvordan man kunne gi støtte til uttesting av teknologiens funksjonalitet, slik at designere (målgruppen) kan få synliggjort så tidlig som mulig hvordan teknologien kan anvendes, og få verdifulle ideer om aspekter ved teknologien.
Denne hovedoppgaven tar for seg hvordan man kan tilrettelegge for ny teknologi i en utviklingomgivelse. Det er satt fokus på hvordan man kan abstrahere teknologisk kompleksitet og presentere teknologien til designerne ved hjelp av et grensesnitt det er lett for dem å forholde seg til. Ved å legge til rette for konkretisering av ulike begrep knyttet til teknologien, håper jeg å vise hvordan man kan gi støtte for prototyping av teknologien og gi rom for ideer og anvendelsesområder.

I noen systemutviklingsprosjekter kan det være hensiktsmessig å vente med å utvikle den reelle funksjonaliteten til senere i systemutviklingsprosessen, fordi det å få teknologien til å fungere kan være teknisk komplisert, tidkrevende og kreve mye "plunder". Man har vurdert det som mest hensiktsmessig å prototype simuleringer av teknologien i designfasen av utviklingen – med papirsketcher eller skjermbilder - for å få testet det konseptuelle designet.

I andre sammenhenger kan det derimot være behov for å prøve ut den reelle funksjonaliteten i en tidlig fase, for at designere og brukere skal få se mulighetene og begrensningene som ligger i samspillet mellom ulike interaksjonsteknikker og den underliggende teknologien. Teknologien kan ha egenskaper som gjør det vanskelig å danne seg et realistisk inntrykk av teknologiens funksjonalitet og hvordan den kommer til uttrykk i brukergrensesnittet. Kommunikasjonsteknologier - som krever en stor grad av interaksjon mellom de kommuniserende enhetene, er eksempler på teknologier som gjerne krever realistisk prototyping.

Det vesentlige her er at designer så tidlig som mulig kan få en realistisk demonstrasjon av hvordan teknologien virker, samtidig som at teknologien presenteres i en slik innpakning at den er stemmer overens med det bildet man som tilrettelegger antar at designeren har. Slik kan han se med sine egne øyne hvordan teknologien virker uten å rote seg bort i detaljer eller vage antakelser om hvordan den fungerer. Folk kan ha lettere for å forholde seg til det som er håndfast enn det som er vagt og usikkert.

I kapittel 1.4 og 1.5 vil jeg forklare begrepene utviklingsomgivelse og prototyping.

1.3 Disposisjon

Denne hovedfagsoppgaven består av tre hoveddeler.

Del I ser på formålet med dette hovedfagsstudiet, problemstillingen som ligger til grunn for oppgaven og den teoretiske bakgrunnen.
Bluetoothgrensesnittet blir introdusert som et case som illustrerer hvordan ny teknologi generelt kan gjøres tilgjengelig for prototyping i en utviklingsomgivelse. Det blir gjort rede for hvorfor caset er interessant sett i lys av problemstillingen og det antydes hvordan problemstillingen ble angrepet. I forbindelse med dette blir det framhevet viktigheten av å abstrahere ut kompleksiteten, slik at teknologien blir presentert i et grensesnitt som målgruppen lettere kan forholde seg til.
En utviklingsomgivelse bør helst gi både design - og programmeringsmuligheter hvis den skal kunne brukes til prototyping av ny teknologi. Den historiske utviklingen innen programmeringsspråk og brukergrensesnittdesign blir derfor gjennomgått.
Det blir sett nærmere på framveksten av fagfeltet menneske-maskin-interaksjon og hvordan teorier fra blant annet kognitiv psykologi har blitt anvendt til å finne nye måter å utvikle brukergrensesnitt på. Konseptuelle modeller presenteres som hjelpemiddel for å lage brukervennlige grensesnitt.
I forbindelse med brukergrensesnittdesign blir det satt søkelys på hvorfor og hvordan prototyping inngår som en integrert del av designprosessen.
Til slutt ses det på ulike egenskaper ved moderne utviklingsomgivelser som egner seg til hurtig prototyping og forskjellige utviklingsomgivelser blir vurdert ut fra om de tilbyr utvidelsesmulighet og dermed egner seg til hurtig prototyping av ny teknologi.

Del II tar for seg den empiriske delen av oppgaven og beskriver hvordan Bluetooth-grensesnittet ble utformet og erfaringer som ble gjort under uttesting av grensesnittet.

Egenskaper ved teknologien blir beskrevet og det gjøres rede for hvordan de viktigste egenskapene blir abstrahert og hvordan en konseptuell modell for teknologien består av sentrale absraherte begrep.
Valg av utviklingsomgivelse og grensesnitt blir begrunnet og det ses på hvordan man kan utnytte de muligheter utviklingsomgivelsen gir for utvidelse av funksjonaliteten. Grensesnittets interaksjonsmekanismer (se kap. 6.8) og utviklingsomgivelsens programmerings –og designmuligheter blir beskrevet nærmere.
Deretter forklares det hvordan interaksjonsmekanismene utnyttes i grensesnittet.
I testdelen beskrives de fire forsøkene; hvordan de ble gjennomført, hvilke observasjoner som ble gjort og noen betraktninger rundt resultatet av forsøkene.

I **Del III** blir Bluetooth-grensesnittet drøftet i forhold til hensikten, problemstillingen, den teoretiske bakgrunnen for studiet og empiri.
Det blir vurdert hvordan hver av delproblemene blir løst og det foreslås ulike forbedringer av løsningen til grensesnittet.
Videre blir det lagt fram retningslinjer for hvordan man generelt kan utvikle et grensesnitt mot ny teknologi. Disse retningslinjene er på grunnlag av erfaringene som ble gjort i dette hovedfagsprosjektet med å utvikle et Bluetooth-grensesnitt og hvordan grensesnittet ble mottatt av testpersonene.
Til slutt konkluderes arbeidet som er gjort med hovedoppgaven..

Nedenfor er hovedfagsoppgavens hovedkapitler oppsummert for å gi en kort oversikt over hva oppgaven omhandler og hvordan den er strukturert.

- **Del I: Innledning, problemstilling og teori**
  -Kap.1 omhandler bakgrunnen og hensikten med hovedfagssoppgaven og definerer noen sentrale begrep knyttet til problemstillingen.
  -Kap.2 beskriver problemstillingen og hvordan den ble angrepet. Prototyping av Bluetooth blir introdusert som case.
  -Kap.3 beskriver hvilke metoder som ble benyttet for å løse problemstillingen.
  -Kap.4 ser på utviklingen innen programmeringsspråk og framveksten av fagfeltet menneske-maskin-interaksjon samt kognitive modeller som har betydning for løsningen på problemstillingen. Det blir også gjort rede for sammenhengen mellom designprosessen og prototyping i dette kapitlet.
  -Kap.5 ser på ulike utviklingsomgivelser og hvordan de egner seg til å lage funksjonelle prototyper.

- **Del II: Komponent og test**
  -Kap.6 beskriver hvordan teknologiske begrep ble abstrahert slik at teknologien kunne prototypes. Det blir redegjort for valg av utviklingsomgivelse og grensesnitt og den konseptuelle modellen til utviklingsomgivelsen og teknologien blir presentert.
  Det blir gjort rede for hvordan utviklingsomgivelsen og interaksjonsmekanismene i grensesnittet kan benyttes til å prototype ny teknologi.
  -Kap.7 tar for seg uttestingen av grensesnittet. Hvert av de fire forsøkene blir beskrevet og resultatet av forsøkene blir drøftet.

- **Del III: Refleksjoner**
  -Kap.8 drøfter ulike forslag til forbedring av grensesnittet.
  -Kap.9 drøfter hvordan de enkelte delproblemene ved problemstillingen ble løst og hvilke lærdommer man kan dra ut av løsningen på problemstillingen.
  -Kap.10 foreslår noen retningslinjer for hvordan man kan gå fram for å lage et grensesnitt mot ny teknologi, på bakgrunn av erfaringene fra hovedfagsprosjektet.
  -Kap.11 oppsummerer hovedfagsoppgaven og erfaringene jeg har gjort meg i forbindelse med skrivingen av oppgaven.

1.4 Hva er en utviklingsomgivelse?

Det er nødvendig å definere hva som legges i begrepet utviklingsomgivelse i forbindelse med dette studiet.

En utviklingsomgivelse kan defineres som et programvaresystem som støtter utvikling og vedlikehold av programvare (Habermann, 1986).

En mer detaljert definisjon av utviklingsomgivelser:

> "A system for supporting the process of writing software. Such a system may include a syntax-directed editor, graphical tools for program entry, and integrated support for compiling and running the program and relating compilation errors back to the source.
> Such systems are typically both interactive and integrated, hence the ambiguous acronym. They are interactive in that the developer can view and alter the execution of the program at the level of statements and variables. They are integrated in that, partly to support the above interaction, the source code editor and the execution environment are tightly coupled, e.g. allowing the developer to see which line of source code is about to be executed and the current values of any variables it refers to."

(FOLDOC, Free On-Line Dictionary of Computing, 2003)

Fellestrekk ved disse definisjonene er at utviklingsomgivelsen ses på som et system, en samling av elementer som er organisert for et felles formål, for å forenkle programvareutviklingen.

Hva jeg legger i begrepet sett i kontekst av denne oppgaven er at en utviklingsomgivelse er en applikasjon som støtter opp om programmering og design av programvare.
Hensikten med en utviklingsomgivelse er å lette programutviklingsprosessen ved å tilby et integrert sett med verktøy. Med verktøy mener jeg i denne oppgaven hjelpemidler som letter arbeidet med å lage grafiske grensesnitt, programmere, strukturere programkoden, finne feil i programkoden, administrere interne og eksterne systemressurser osv. i den gitte utviklingsomgivelsen.
Designverktøy til bruk i systemeringen av designet, som UML, RUP osv. regnes vanligvis ikke som en del av selve utviklingsomgivelsen, men det er viktig at overgangen fra for eksempel UML-representasjoner og til programmering blir ivaretatt.
Siden en moderne utvklingsomgivelse gjerne brukes til prototyping, er det avgjørende at den legger til rette for at designeren/programmereren kan konsentrere seg om selve utformingen av produktet som skal utvikles. Utviklingsomgivelsene som jeg tenker på i denne oppgaven abstraherer derfor bort datatekniske hindre og tilbyr f.eks grensesnittbyggere, relativt enkle programmeringsspråk og automatisk håndtering av omgivelsesvariabler og protokoller gjennom wizarder og andre overbygninger.

Eksempler på hva jeg vil definere som utviklingsomgivelser er Flash, Director, Visual Basic, Visual C++, Supercard og JBuilder. Alle tilbyr en grensesnittbygger, kodeeditor og en del andre verktøy til å bygge opp et program/prototyp.

I et systemutviklingsperspektiv har dataprogrammene blitt større og mer omfattende, noe som har gjort at nye metoder og verktøy har blitt introdusert for å lette prosessen med å utvikle applikasjoner. Trenden har gått i den retning at kompleksiteter har blitt abstrahert vekk og programmereren har fått nye begreper på høyere abstraksjonsnivå å forholde seg til, fra maskinkode til ”naturlige språk”. Ved hjelp av parsere og kompilatorer ble det ”naturlige språket” gjort om til maskinkode.

1.5 Hva er prototyping?

I denne oppgaven inngår prototyping som et sentralt begrep.
Kort fortalt vil prototyping si å lage foreløpige og ufullstendige utgaver av et produkt.

Generelt blir det laget prototyper for å finne ut om produkter som utvikles fungerer som de skal og for å oppdage svakheter i produktdesignet. I tradisjonell ingeniørkunst har prototyping lignet på et "laboratorieforsøk" der man har utformet et konsept og så konstruert en forenklet versjon av konseptet og testet under kontrollerte forhold om konseptet fungerte som det skulle (Winograd, 1995).

Programvareindustrien brukte allerede tidlig på 1970-tallet prototyper til modellering, simulering eller delvise implementering av systemer. Prototypene ble så testet for å se om ulike tekniske aspekter ved prototypene var mulig å implementere i de ferdige produktene eller for å fastslå forskjellige krav til produktene (Carr og Verner, 1995).

I systemutviklingssammenheng kan en prototyp sies å fange inn de karakteristiske trekk ved et ferdig system. En prototyp er ufullstendig og vil ofte bli endret, supplert eller erstattet av en ny versjon (Naumann og Jenkins, 1982).
Prototyper er ansett for å være viktige for å uttrykke, utforske og evaluere designet av datasystemer.

Prototyping kan bli sett på som en prosess, der prototyping inngår som en viktig del av systemutviklingen. Prototyping er nødvendig for å sikre at en applikasjon tilfredsstiller behovene til brukere av applikasjonen. Testing av prototyper på brukere kan gi verdifull informasjon som kan brukes til å forbedre produktet. Det kan ellers være problematisk å involvere brukere underveis i designet av et system, da brukerne vanskelig kan få et inntrykk av designet kun ved å se på tekniske dokumenter og spesifikasjoner. Prototyping er altså et middel for å kommunisere designet.
Det er allmenn praksis å bygge prototyper for å demonstrere ulike utgaver av et design som er under utarbeidelse og for å tydeliggjøre ulike designvalg. Siden datasystemer/applikasjoner kan bli store og kompliserte kan det imidlertid være vanskelig å uttrykke alle aspektene ved designet på en gang (Houde og Hill, 1997). En prototyp tar derfor for seg et utvalg aspekter ved et system.
Det finnes ulike prototypingsteknikker som kan benyttes til å belyse ulike aspekter ved produktet som er under utarbeidelse. Hvilke teknikker som benyttes kommer an på hva som skal prototypes og i hvilken sammenheng. Ulike former for prototyper kan brukes til forskjellige formål (Carr og Verner, 1995). Dessuten kan det benyttes flere teknikker til å prototype det samme designet.

Prototypingsteknikker kan karakteriseres som :

- hurtig eller evolusjonær prototyping
- low-fidelity eller high-fidelity prototyping
- horisontal eller vertikal prototyping
- rolle, look-and-feel eller implementasjons-prototyper (Houde og Hill, 1997)

Hurtig prototyping (throw-it-away) : mulige funksjonelle og designmessige aspekter blir demonstrert i en prototyp som vurderes av brukere. Ligner på en annen teknikk kalt *requirement animation*, men hurtige prototyper har det kjennetegnet at de ikke utvikles videre til ferdige produkter, men forkastes og erstattes av en ny prototyp eller et ferdig produkt.

Evolusjonær prototyping : Prototypen bygges, evalueres og videreutvikles kontinuerlig, helt til det ferdige produktet foreligger. Prototypen inngår altså i den endelige løsningen, og det blir dermed lagt noe mer vekt på funksjonaliteten, der prototypen implementeres og videreutvikles underveis.

Low-fidelity prototyper er sketcher eller papirprototyper som først og fremst viser ideer begrep knyttet til designet, men som også kan gi informasjon om interaksjonen i designet. Slike prototyper brukes ofte i idestadiet av en designprosess.

High-fidelity prototyper framstilles på et medium som en PC, video eller lignende og er ment å gi en høyere grad av realisme og ligne på det endelige designet. Slike prototyper framstilles gjerne som skjermbilder eller sekvenser av skjermbilder og belyser gjerne interaksjonen i et brukergrensesnitt. Designet er forseggjort og funksjonaliteten blir gjerne simulert slik at prototypen virker mest mulig realistisk for brukeren.

Horisontal prototyping legger vekt på å belyse mest mulig av brukergrensesnittet og kalles også for *mock-up* prototyping. Funksjonaliteten bak "knappene" blir ikke demonstrert.

Vertikal prototyping benyttes for å belyse visse aspekter ved funksjonaliteten og går i dybden når det gjelder å demonstrere avgrensede deler av systemet.

Videre definerer Houde og Hill prototyper som enhver framstilling av en designide uansett bruk av medium (Houde og Hill, 1997). Prototyper lages for å undersøke ulike aspekter ved designet og hvilke sorter prototyper som utvikles avhenger av de designspørsmålene som skal besvares.

Rolle-prototyper kan utvikles for å undersøke hvilken betydning et system vil ha for brukeren og hvilke muligheter systemet skal tilby for å tilfredsstille behovene til brukeren.

Look-and-feel-prototyper utvikles for å undersøke hvordan designet bør presenteres, det vil i praksis si hvordan brukergrensesnittet kan utformes.

Implementasjons-prototyper kan utvikles hvis formålet med prototypen er å finne ut av funksjonaliteten til en ny teknologi.

2. Problemstilling

2.1 Presisering av problemstillingen

Jeg vil nå komme nærmere inn på problemstillingen som denne hovedfagsoppgaven bygger på og hvilke delproblemer denne problemstillingen innebærer.

Denne hovedfagsoppgaven handler om det å abstrahere ny teknologi og presentere abstraksjonen på en slik måte at det er enkelt å prototype teknologien. For å oppnå dette stilles det krav til hvordan abstraksjonen presenteres. Når man legger til rette for prototyping av ny teknologi er det gjerne naturlig å fastslå hvordan den som prototyper teknologien vil ha det – det kan knyttes ulike kvalitetskrav til bruken av teknologien som presenteres.

Oppgavens problemstilling kan uttrykkes ved hjelp av følgende spørsmål :

"Hvordan best legge til rette for prototyping av ny teknologi i en utviklingsomgivelse?"

Begrepene i problemstillingen krever en nærmere forklaring.

Ny Teknologi er et vidt begrep og jeg tar ikke for meg prototyping av alle typer teknologi. Med ny teknologi mener jeg i denne oppgaven primært informasjons – og kommuniksjonsteknologi og multimediaelementer som for eksempel ulike lyd og bilde-formater. Teknologiene jeg sikter til er også relativt nylanserte og hittil helt eller delvis utilgjengelig for prototyping for folk flest.

Når det gjelder *utviklingsomgivelser* kan man si at de fleste støtter prototyping på en eller annen måte, da systemutvikling som oftest innebærer bygging av prototyper. Men det er bare et begrenset antall av dem som det er hensiktsmessig å tilrettelegge for prototyping av ny teknologi i. Utviklingsomgivelsen må tilby muligheten til å utvide funksjonaliteten slik at en bestemt teknologi kan anvendes. I tillegg må utviklingsomgivelsen kunne benytte den aktuelle teknologien i bygging av prototyper på en enkel måte.
Problemet er altså å finne en egnet utviklingsomgivelse som gir tilgang til en gitt teknologi på enklest mulig måte slik at teknologien kan prototypes.

Men hvordan kan man så *best* mulig legge til rette for prototyping av teknologien i utviklings-omgivelsen?
Det er ikke noe entydig svar på dette spørsmålet, men det kan identifiseres ulike kvalitetskrav som man mener at målgruppen stiller. For å kunne forutsi hvilke krav som stilles er det hensiktsmessig å definere målgruppen.

Målgruppen i denne oppgaven er designere med en viss datakunnskap og noe erfaring med enkel programmering. Med designere mener jeg enhver person som lager prototyper ved hjelp av en utviklingsomgivelse og som ikke har programmering som viktigste arbeidsoppgave.

Egenskaper ved den enkelte designer, slik som erfaring, motivasjon og personlighet kan ha innvirkning på brukskvaliteten.
Hva som gir god brukskvalitet for en gitt designer kan i sin tur avhenge av hva som er formålet med prototypingen og i hvilken sammenheng det prototypes, noe man bør ta hensyn til når man legger til rette for prototyping av ny teknologi.

Hvis man skal tilby til designere en abstraksjon av ny teknologi representert i form av et forenklet grensesnitt – slik jeg har gjort i dette hovedfagsstudiet - kan man identifisere bestemte overordnede kvalitetskrav.

Jeg har utviklet et grensesnitt med tanke på følgende krav til brukskvalitet :

1. *Grensesnittet skal gjøre det enkelt for designer å prototype brukergrensesnitt og funksjonalitet.*
   Hvordan kan man så måle hvor godt en designer klarer å lage et brukergrensesnitt og funksjonalitet?
   Hvor hurtig en designer klarer å lage en prototyp ved hjelp av grensesnittet kan si noe om hvor bra brukskvalitet grensesnittet gir for den bestemte designeren til det gitte formålet i en gitt sammenheng.
   Man kan også se på hvor stor nytteverdi prototypen som produseres har - hvor nøyaktig designeren klarer å demonstrere teknologien og hvor "brukbare" prototyper som kan produseres. Dette kan det imidlertid være vanskelig å finne et presist mål på.
   Brukbarhetstester som designere senere utfører på prototyper som de har benyttet grensesnittet til, kan gi svar på dette. Det kan i tillegg settes fram andre mål på om dette brukskravet oppnås.

2. *Grensesnittet skal gi designeren rom for kreativitet og ideer for anvendelse*
   Det er også naturlig å spørre seg hvorvidt grensesnittet gir designeren rom for ideer og kreativitet. Det kan være vanskelig å måle dette i et kontrollert forsøk der testpersonene har begrenset tid på seg lage en prototyp og gjerne har liten erfaring med utviklingsomgivelsen som prototypen skal lages i.

Det er disse kravene jeg tar utgangspunkt i når jeg i denne oppgaven vurderer hvordan man kan legge best mulig til rette for prototyping av ny teknologi. Om det legges mest vekt på enkelhet eller kvalitet, eller på om prototypingen gir rom for ideer og kreativitet, avhenger blant annet av brukssammenhengen og hensikten med prototypen.

I forbindelse med prototypingen er det vanlig at designer gjennomfører brukbarhetstester. Tilbakemeldinger fra testene kan også si noe om brukskvaliteten til teknologigrensesnittet. Testing av prototyper er en av flere måter å evaluere brukskvaliteten (usability) på, men de ulike måtene å evaluere på vil gjerne ta utgangspunkt i bestemte brukskvalitetskriterier, som for eksempel de to brukskravene ovenfor.

Når ny teknologi, utviklingsomgivelse, målgruppen og kvalitetskravene knyttet til problemstillingen nå er definert, kan følgende delproblemer trekkes ut for å få klarlagt de enkelte aspektene ved problemet :

- *Hvordan kan man abstrahere de viktigste egenskapene ved en ny teknologi slik at man reduserer teknologiens kompleksitet?*
- *Hvordan kan man presentere en ny teknologi for designere?*
- *Hvilken nytte gir det for designere å kunne prototype teknologiens funksjonelle egenskaper?*

2.2 Aktualisering av problemstillingen

I denne oppgaven ser jeg på hvordan ny teknologi kan gjøres tilgjengelig for prototyping ved å ta for meg et bestemt case; kommunikasjon mellom mobile enheter - Personal Digital Assistents (PDA) - ved hjelp av den trådløse radioteknologien Bluetooth.

Det som gjør Bluetooth interessant sett i lys av problemstillingen er at den illustrerer de utfordringene som ligger i å gi tilgang til prototyping av teknologisk funksjonalitet generelt. Bluetooth er en kompleks teknologi som man ennå har lite erfaring med. Det kommer nye spesifikasjoner og bruksområder etter hvert som teknologien modnes, men det er forholdsvis lite standardisering sammenlignet med programvare og maskinvare på stasjonære PC'er.

En teknologi har både tekniske og konseptuelle aspekter knyttet til seg. Når teknologien skal anvendes må man forholde seg til begge aspekter. Jeg ser i denne oppgaven på hvordan man best kan legge til rette for å prototype de tekniske/fysiske egenskapene til teknologien og reflekterer samtidig over nødvendigheten av å få demonstrert disse egenskapene. En av svarene er at funksjonelle prototyper kan få en designer til å se muligheter og begrensninger ved Bluetooth. Det er vanskelig å simulere egenskapene til Bluetooth og jeg prøver i denne oppgaven å få svar på om designere kan lage brukergrensesnitt med høyere brukskvalitet hvis de får demonstrert teknologien på en realistisk måte i prototypen.

2.3 Hva er Bluetooth?

Bluetooth er en trådløs teknologi som sender informasjon via radiosignaler (2,4 GHz båndet). Teknologien fjerner behovet for kabler og ledninger mellom både stasjonære og mobile enheter. Den gir muligheter for både data og taleoverføring, samt at den gir muligheter for "ad hoc" nettverk hvor enheter kan synkroniseres mot hverandre.
Når to Bluetooth-enheter kommer innen 10 meters rekkevidde, kan de opprette en forbindelse med hverandre. Siden Bluetooth benytter en radiobasert forbindelse, trengs ikke direkte synslinje for å kommunisere.

Ideen bak Bluetooth teknologien kom fra Ericsson, som i 1994 bestemte seg for å utforske mulighetene for et strømbesparende og lavkostnads radiobølge-grensesnitt mellom mobiltelefoner og tilhørende utstyr.

I 1998 ble Special Interest Group (SIG) opprettet. Den består av bedrifter som i fellesskap overvåker utviklingen av kortdistanseradio og utvikler en global standard for Bluetooth teknologien. Dette arbeidet førte til at første utgave av Bluetooth-spesifikasjonene ble utgitt i 1999, og spesifikasjonene blir stadig utviklet i dette samarbeidet. Det viktigste målet for SIG var full samhandling mellom forskjellige enheter fra forskjellige hardwareprodusenter.

Figur 2.1 RF400 Spread Spectrum Bluetooth-radio

Alle Bluetooth-enheter testes for å verifisere at de oppfyller de krav som stilles til radiokommunikasjonskvaliteten, lavnivå protokoller, profiler og informasjon som presenteres til sluttbrukeren.
Typisk for en kommunikasjonsteknologi som Bluetooth er høy interaktivitet og et tett forhold mellom teknologien og maskinvaren på enhetene som kommuniserer.

2.4 Mobil IT

De seneste årene har mobile enheter slik som laptop, mobiltelefoner, PDA'er begynt å utfordre de tradisjonelle stasjonære PC'ene.

Det er ikke lenger slik at databehandling nødvendigvis skjer på en stasjonær PC knyttet til et bestemt sted. Ideen med *ubiquitous computing* er at ulike elektroniske enheter blir integrert overalt i omgivelsene. Disse enhetene har ulik størrelse og egenskaper og har innebygde mikroprosessorer og annen maskinvare som gir muligheten til å samhandle med andre enheter rundt dem på kontoret, i offentlige rom eller privat. Enhetene kommuniserer med hverandre ved hjelp av trådløs teknologi som Bluetooth (Myers, 2000), eller andre trådløsteknologier som IR eller WLAN.
Håndholdte enheter og stasjonære PC'er blir anvendt veldig forskjellig, selv om begge har en informasjonsbehandler (information management) og kommunikasjonsegenskaper og synkroniserer data på en sammenlignbar måte. Applikasjoner designet på en type enhet trenger ofte å designes på nytt igjen for å fungere eller være nyttig på en annen enhet (Weiss, 2002).

En PDA er en høyfunksjonell håndholdt enhet med en liten touchscreen og mulighet for å legge inn input ved hjelp av penn og digitalt tastatur etc. adressebok, kalenderfunksjoner og annen programvare (Weiss, 2002). Som andre håndholdte enheter har PDA'en gjerne kommunikasjons-programvare - og maskinvare.

Figur 2.2 Mobile enheter med display

Meningen er at enhetene skal være i stand til å respondere på hverandre på en fornuftig måte. Enhetene vil interaktivt kunne utveksle data ut fra den bestemte situasjonen det skjer i. Dette kalles kontekstsensitivitet og utgjør hovedaspektet ved begrepet ubiquitous computing. Et enkelt eksempel på kontekstsensitivitet kan være at elementer i skjermbildet til en PDA endrer seg når det kommer i nærheten av en annen enhet.
Å utvikle programvare og verktøy for mobile plattformer gir nye utfordringer i forhold til for stasjonære PC'er.

- Det er ikke praktisk mulig å bruke de samme grafiske brukergrensesnittelementene på de små skjermene som mobile enheter gjerne har. Begrensede ressurser med hensyn til internminne, prosessorhastighet og lagringsplass gjør at utvikling mot mobile enheter blir nærmere knyttet til maskinvaren enn for PC'er, fordi de som designer/utvikler programmer og verktøy hele tiden må være oppmersom på maskinvarens egenskaper.

- En annen betydningsfull faktor er at enhetene som kommuniserer kan ha vidt forskjellige egenskaper og kan inneholde programvare på ulike maskinvare – og operativsystemplattformer. PDA'er med innebygd Bluetooth har oftest en egen Bluetooth-manager hvor brukeren kan konfigurere kommunikasjonsegenskaper, navn på enhet, tilgjengelighet mot andre enheter, LAN-oppkobling osv. Man må ta hensyn til Bluetooth-manager når man lager programmer/prototyper.

- En faktor jeg også vil framheve er egenskapene til den teknologien som brukes for å knytte enhetene sammen. PC'er benytter ofte den standardiserte TCP/IP og relaterte protokoller definert i OSI-modellen til å kommunisere. Utviklere trenger da kun forholde seg til veldefinerte grensesnitt og API'er for å få applikasjoner til å kommunisere i en klient-tjener-arkitektur. Mangelen på allmenne kommunikasjonsstandarder for mobil IT, selv for samme type enheter, gjør det dessuten enda mer krevende å utvikle programvare. Den trådløse kommunikasjonsteknologien Bluetooth medfører kommunikasjon og ad-hoc utveksling av data der det ikke er definert noen klar klient eller tjener. Dette kompliserer samhandlingen mellom enheter og gjør at designere/utviklere må konsentere seg mer om selve kommunikasjonen enn på stasjonære datamaskiner. I omgivelser der Bluetooth benyttes varierer det hvilke enheter med ulike tjenester som er tilgjengelige. Kalenderen til en PDA kan for eksempel bli synkronisert automatisk med en PC når den kommer innen rekkevidde av den. Det er dette som gjør Bluetoothkommunikasjon så ulikt mer tradisjonelle nettverksbaserte omgivelser.

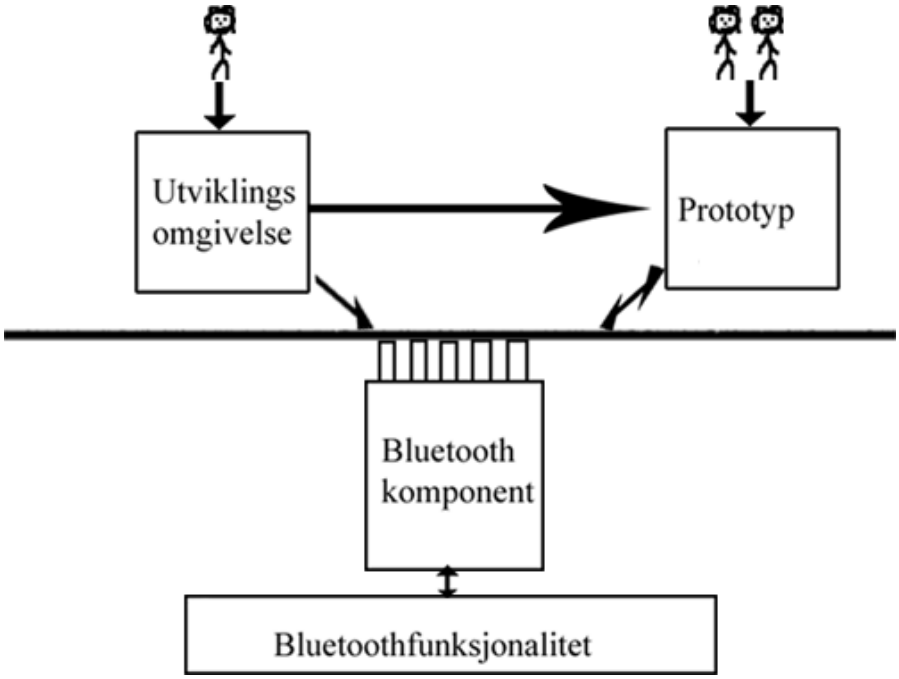
Siden mobile enheter er så lite standardiserte og så maskinvare- og kommunikasjonsteknisk avhengige, er terskelen sannsynligvis høyere for å utvikle programvare for kommunikasjon mellom slike enheter enn mellom ordinære PC'er.

Samtidig vil det være verdifullt å kunne lage funksjonelle prototyper for kommunikasjon mellom mobile enheter, da aspekter ved maskinvaren og kommunikasjonsteknologien kan ha føringer på programvaredesignet.

2.5 Hvordan problemstillingen ble angrepet

I den praktiske delen av hovedfaget har jeg sammen med Frank Werner Johansen arbeidet med å gjøre Bluetooth tilgjengelig i en utviklingsomgivelse. Vi så for oss at det var nødvendig å tilby et forenklet grensesnitt mot bluetooth-teknologien, og startet dermed å lete etter en utviklingomgivelse hvor Bluetooth kunne integreres. Dette medførte at utviklingsomgivelsens funksjonalitet måtte utvides for å nyttegjøre muligheter med Bluetooth, siden vi i utgangspunktet ikke kjente til noen utviklingsomgivelse som tilbød dette i selve omgivelsen. Utviklingsomgivelsen måtte dessuten kunne benyttes til å lage hurtige prototyper, så den måtte ha en lav terskel og være enkel å designe skjermbilder og programmere funksjonalitet i. Videre måtte grensesnittet mot Bluetooth være lett å sette seg inn i og være så intuitivt forståelig som mulig, slik at det enkelt kunne benyttes i den aktuelle utviklingsomgivelsen. Vi ville redusere kompleksiteten til teknologien ved å trekke ut de egenskapene som vi anså som viktige for den tenkte brukergruppen – designere som vil prøve ut mulighetene med Bluetooth. Vi måtte ta hensyn til at brukergruppen gjerne hadde begrensede tekniske – og programmeringsmessige forkunnskaper.
Det ble ansett som viktig at designeren kunne benytte grensesnittet ut i fra forestillinger om - eller tidligere erfaring med lignende type teknologi.

Problemstillingen innebar at teknologien måtte presenteres til designere i en begripelig innpakning. Vi startet med å undersøke hvordan ulike utviklingsomgivelser egnet seg til å prototype Bluetooth-funksjonalitet og på hvilken måte de kunne utvides med denne funksjonaliteten.
For å få funksjonaliteten inn i en begripelig innpakning var løsningen å tilby et enkelt grensesnitt som kunne benyttes fra utviklingsomgivelsen til designe en prototyp/program. Et slikt grensesnitt ville være et overbyggende lag over Bluetooth-funksjonaliteten som skjulte de tekniske detaljene til Bluetooth-stacken, kommunikasjonslaget mot Bluetooth.



Figur 2.3 Komponentens grensesnitt mot Bluetooth

Vi innså at brukergruppen ikke kunne forholde seg til Bluetoothstack'en direkte og laget derfor en overbygning som skjulte de tekniske detaljene ved teknologien. Ideen var å la designere kommunisere med teknologien via et grensesnitt som var lett forståelig og mest mulig tilpasset brukergruppens framgangsmåte og forestilling om teknologien.

3. Metode

I dette kapittelet vil jeg beskrive hvilke forskningsmetoder jeg har benyttet i forbindelse med hovedfagsoppgaven.

I forbindelse med hovedfagsoppgaven har jeg utført et litteraturstudium, utvikling av et grensesnitt mot ny teknologi, samt en enkel brukbarhetstest med observasjon og intervju.

3.1 Litteraturstudium

Arbeidet med hovedfagsoppgaven medførte lesing av litteratur. Etter at problemstilling og eventuelle hypoteser er utformet vil neste fase være litteraturstudiet, da det er viktig å få tak i hovedproblemstillingen innen fagfeltet (Hartvigsen, 1998).
Litteraturen besto av bøker, artiker og internettkilder som jeg fant gjennom søk i ACM Digital Library (Association of Computing Machinery), BIBSYS og på internett.
Jeg søkte etter relevant teori og informasjon omkring tema som blant annet brukergrensesnitt, prototyping, utviklingsomgivelser og Bluetooth.
Litteratursøket ble først og fremst gjort under Del I av oppgaven, for å finne teori som kunne være med i oppgaven og for å øke mine kunnskaper innen fagfeltet, men litteraturstudiet dreide seg også om å finne informasjon om Bluetooth-teknologien og teknologien/verktøyene jeg skulle benytte for å programmere den tekniske løsningen til Bluetooth-grensesnittet.
Det var nødvendig å sette seg inn i både teorien og teknologien allerede i startfasen av hovedfaget for å forstå hvordan jeg skulle angripe problemstillingen.

3.2 Utvikling

Det ble utviklet en løsning for å gjøre Bluetooth-teknologi tilgjengelig i en dertil egnet utviklingsomgivelse i samarbeid med Frank Werner Johansen.
Løsningen ble en ActiveX-komponent på mobilplattformen utviklet i utviklingsomgivelsen Embedded Visual C++ 3.0 og beregnet for bruk i Embedded Visual Basic 3.0.

Etter mye prøving og feiling for i det hele tatt å få ActiveX-komponenten til å fungere, ble grensesnittet utformet. Jeg hadde hovedansvar for det konseptuelle designet av grensesnittet, mens Frank var pådriver for implementasjonen.
Det ble gjort hyppig bruk av diskusjonsfora og internettsider som MSDN for å komme opp med en programmeringsmessig løsning. Dessuten studerte vi nøye dokumentasjonen til Bluetooth-API'en som komponenten benyttet samt at vi utvekslet epost med utviklerne av API'en vi benyttet.
Det ble eksperimentert mye med Bluetoothteknologien for å utvikle en så optimal løsning som mulig med hensyn til de begrensninger som lå i teknologien. For eksempel er ikke en Bluetooth-enhet søkbar når den selv er i søkemodus. Dette medførte at vi utviklet et intervallsøk der vi eksperimenterte med forskjellige søkeintervall (3 til 12 sekunder).
Det dukket altså opp utfordringer underveis. Ikke bare måtte de løses programmeringsteknisk, men komponenten måtte designes slik at brukeren ble skjermet mot en del tekniske kompleksiteter.

3.3 Test av komponenten

3.3.1 Observasjon og deltakelse i prototyping

Det ble avholdt fire forsøk der det ble testet hvordan det var å bruke Bluetooth-grensesnittet.

Forsøkene innebar at en forsøksleder hadde en konsultativ rolle under designet av en prototyp. Prototypen ble designet i Embedded Visual Basic på en ordinær PC, men testet ut på mobile enheter (PDA) (se kapittel 7 for mer om testopplegget).
Forsøkspersonene hadde omlag en time på seg til å få opp en funksjonell Bluetooth-prototyp og de forskjellige forsøkspersonene fikk ulik tilgang til dokumentasjon og andre hjelpemidler.

Et videokamera registrerte det som skjedde under forsøket. Videreopptaket ble senere gått igjennom og analysert for å kunne trekke ut de viktigste resultatene av forsøkene.

3.3.2 Intervju

Det ble avholdt et kort uformelt intervju etter hvert av forsøkene der forsøkspersoner ble spurt hvordan de opplevde Bluetooth-grensesnittet, samt hva de syntes var bra og hva de syntes manglet eller var uklart.
Intervjuene fungerte som supplement til designet av prototypene og meningen var å få mer tilbakemelding fra forsøkspersonene etter at "stresset" med prototypdesignet var lagt seg.

3.4 Kritikk av metoder og forsøkenes validitet

Da problemstillingen har vært å se på hvordan man kan presentere ny teknologi på en lettfattelig måte - med fokus på de kognitive egenskapene til målgruppen - inneholder studiet mange faktorer som ikke er målbare. Metoden som er brukt i dette studiet har derfor et sterkt kvalitativt preg. Det er for eksempel vanskelig å finne nøyaktige mål på hvor intuitive eller konsistente ulike programmeringsmetoder er eller hvorvidt en bestemt konseptuell modell gjør det lettere enn en annen modell å prototype funksjonalitet. Resultatene må tolkes i en eller annen sammenheng for å gi mening.
Testmaterialet er for begrenset til at testen kan gi noen entydige svar på problemstillingen til oppgaven, intervjuene var uformelle uten spesielle retningslinjer og sammensetningen av testpersoner var sannsynligvis for homogen til at den er representativ for en sammensatt gruppe designere. Dette er svakheter som man bør være klar over når man betrakter resultatene av testene. Resultatene har først og fremst verdi for å gi retningslinjer og råd for hvordan man kan legge til rette for prototyping av ny teknologi.

Det ble ikke gjennomført brukbarhetstester i ordets rette forstand, da forsøkene for eksempel medførte at forsøkspersonene fikk hjelp underveis i oppgaveløsningen og det ikke ble avholdt nok forsøk til å kunne trekke bestemte konklusjoner kun på grunnlag av testmaterialet. Det er mer en ekspertvurdering. Testene ga imidlertid nyttig tilbakemelding slik at jeg kunne gjøre tilpasninger til kvalitetskravene (se punkter nedenfor). Jeg har derfor valgt å la testene være en indikasjon på hvordan grensesnittet kan imøtekomme designernes krav, i stedet for å gi bestemte svar på hvordan grensesnittet skal utformes.

Testene ga en pekepinn på om grensesnittet som ble utviklet oppfylte kvalitetskravene om at :

- Grensesnittet skal gjøre det enkelt for designer å prototype brukergrensesnitt og funksjonalitet.
- Grensesnittet skal gi designeren rom for kreativitet og ideer for anvendelse.

Grensesnittet skal gi designerne muligheten til å konsentrere seg om oppgaven. Hensikten med å bruke Bluetooth-grensesnittet vil følgelig være å kunne designe brukergrensesnitt og demonstrere teknologiens funksjonalitet og få ideer om hvordan teknologien kan anvendes. Testpersonenes møte med grensesnittet i testen sa noe om hvordan disse kravene ble innfridd, og testresultatene kan gi innspill til hva det er viktig å være klar over når man utvikler et grensesnitt mot ny teknologi.
Det står mer om kvalitetskriteriene i problemstillingen i kapittel 2.

3.5 Avgrensning

- Jeg vil ikke se spesifikt på hvordan grensesnittet mot Bluetooth er implementert. Denne oppgaven retter heller fokuset mot anvendelsesaspektet ved tilrettelegging av ny teknologi.

- Jeg vil ikke gå inn på designteknikker og teorier rundt persepsjon og visuell representasjon. Det er først og fremst kognitive aspekter som blir diskutert.

- Denne hovedfagsoppgaven retter ikke fokus mot problemstillinger knyttet til mobil IT.

- Det er ikke rettet fokus mot sosiale og organisatoriske aspekter omkring design og programvare. Datastøttet samarbeid er for eksempel ikke omhandlet. Oppgaven ser på gruppeaspektet i forbindelse med designteam, men retter ikke søkelyset spesielt mot sosial kontekst.

4. Teoretisk bakgrunn

4.1 Abstraksjon

Abstraksjon er prosessen å redusere antall kjennetegn ved et fenomen ved å trekke ut det som antas å være vesentlig og utelate det som antas å være uvesentlig. Ved å abstrahere ut det essensielle ved et fenomen, utelater man samtidig det man anser at ikke er nødvendig å vite og reduserer dermed kompleksiteten.
Abstraksjon har gått igjen som en mye benyttet mekanisme i utviklingen fra lavnivå til høynivå programmeringsspråk.
Utviklingen gått gradvis fra å skjule detaljer på maskinvarenivå til å lage et strukturert programmeringsspråk på et høyere abstraksjonsnivå hvor man forholder seg til abstrakte begreper slik som moduler, klasser, datatyper etc.

4.2 Historisk perspektiv og programmeringspråk

4.2.1 Maskinspråk og assembler

Jeg vil nå se nærmere på utviklingen fra de første programmeringsspråkene til dagens høynivå språk.

Programmeringsspråkene hadde sin opprinnelse i maskinspråkene som oppsto på 50-tallet. Disse maskinspråkene baserte seg på manipulering av instruksjonssettet i datamaskinen og introduserte algoritmebegrepet, en kommandosekvens som manipulerer data for å løse et problem. Byggesteinene som algoritmen benytter for å løse problemet kalles *primitiver*. Det er programmeringsspråket som angir hvordan disse primitivene kan settes sammen til mer komplekse ideer. Maskinspråkene hadde primitiver på et nivå som tilsvarte instruksjonssettet til datamaskinen og å programmere på denne måten var forbeholdt de få.
Det kunne være en møysommelig prosess å finne feil og rette dem. Programmering ble således sett på som oppgaven å sette sammen en sekvens av maskinkodeinstruksjoner - *algoritmer* - for å få datamaskinen til å gjøre noe nyttig. Programmereren måtte forholde seg til et lavt nivå og skrev kode linje for linje fra å instruere datamaskinen. til å laste data inn i akkumulatoren, øke programtelleren, utføre en operasjon på dataene, lagre resultatet i minneceller og så videre (Wang, 2000).

Man måtte forholde seg til binære tallverdier og ha mye kunnskap om maskinvaren man programmerte mot. For å lage et program var det vanlig for programmerere å skrive koden i en slags 'pseudokode' og så oversette dette til maskinkode som datamaskinen kunne forstå. Dette var tidkrevende og det var lett å gjøre feil. Maskinkode var vanskelig å forholde seg til og det ble komplisert å lage større systemer og å vedlikeholde dem.

Det ble klart på 1950-tallet at man trengte hjelpemidler innen programmering og det ble utviklet ulike typer programmeringsspråk. Med disse språkene fikk man innført flere nye begreper og teknikker (Maus, 2002). Algoritmen skulle forsette å være et sentralt begrep med fremveksten av nye prosedyrale programmeringsspråk men da på et høyere abstraksjonsnivå og med større byggesteiner som støttet problemløsningen på en bedre måte.

Det første hjelpemiddelet innen programmering var assemblere, som hjalp til å få et skille mellom program og data, samt at programmeringsekvensene ble mer leselig. Man fikk navn

på operasjoner/instruksjoner slik som JUMP og STORE som før var representert med tallverdier i maskinkode. Dette var symbolsk kode som abstraherte vekk kompleksiteter og hevet produktiviteten. Man måtte imidlertid fortsatt forholde seg til registre og minneområder slik man gjorde i maskinkode. Med lavnivåinstruksjoner tok det lang tid å utvikle programmer, det var lett å gjøre feil og man måtte forholde seg til den spesifikke datamaskinen man programmerte mot.

4.2.2 Strukturell programmering og prosedyrale språk

På slutten av 1950-tallet ble det utviklet høynivåspråk for å gjøre det lettere å programmere. Det ble arbeidet med å utvikle et mer naturlig språk som støttet problemløsningen på en bedre måte og som fjernet avhengigheten av maskinvare. Algoritmer skulle ikke lenger behøve å bygges med enkeltinstruksjoner som primitiver. Man innførte et høyere abstraksjonsnivå ved at sekvenser av instruksjoner ble satt sammen i større enheter. I stedet laget programmereren algoritmer ved hjelp av større byggesteiner. Man trengte for eksempel ikke lenger forholde seg til i hvilket register dataene oppholdt seg, eller utføre flytting, lagring og innlasting i minnet.

Med FORTRAN og Algol, som var noen av de første høynivåspråkene, fikk man programmeringsspråk som tok i bruk prosedyrebegepet som gjorde det mulig å pakke en sekvens av aksjoner sammen i en enhet og gi denne et navn som beskriver aksjonssekvensen (Maus, 2002). Dette ga muligheter for å dele opp programmet i delprogrammer i stedet for å skrive programmet linje for linje, fra start til slutt.

Begrep som *instruksjoner* og *registre* ble abstrahert vekk slik at programmereren heller forholdt seg til en ”verden” bestående av *variabler*, *pekere*, *kontrollstrukturer* og *prosedyrer/metoder*.
Et oversettingsprogram – en kompilator, analyserte så disse høynivås primitivene, fant syntaksen og gjorde om til maskinkode. Dette lignet på måten en assembler oversatte fra mnemoniske instruksjoner til maskinkode, men mens assembleren tok enkeltsekvenser av instruksjoner, måtte kompilatoren sette sammen flere instruksjonssekvenser for å få utført det som programmereren hadde satt den til å gjøre, derav ordet kompilator.
Det ble altså brukt høyere nivås byggesteiner for å be maskinen om å utføre en sekvens av instruksjoner og disse instruksjonene ble kompilert sammen til et kjørbart program.

Målet om maskinvareuavhengighet innfridd da man hadde abstrahert vekk referanser til eksplisitte kjennetegn ved maskinvaren. Et program utviklet i et tredje generasjons språk kunne i prinsippet kompileres på en hvilken som helst datamaskin, men i praksis var det ikke så enkelt. Det var ulike ”dialekter” av samme språk på forskjellige maskiner og for eksempel var håndtering av I/O-operasjoner maskinvareavhengig (Brookshear, 2003). Noen tilpasninger måtte fortsatt gjøre for å kunne kjøre et program på forskjellige maskiner, så man kunne ikke snakke om full maskinvareuavhengighet, selv om det ble utviklet språkstandarder for å løse problemet.

Det som var et virkelig gjennombrudd var at programmereren nå kunne bygge opp uttrykk ved hjelp av naturlig språk. Med naturlig språk menes menneskespråk (som norsk, engelsk og russisk) som brukes for å uttrykke noe, i motsetning til maskinspråk hvor det brukes ”kunstige” kommandoer og uttrykk for å kommunisere med datamaskiner. Naturlig språk kan

sammenlignes med pseudokode, hvor det angis en handlingssekvens på "godt norsk" som danner grunnlaget for den ferdige koden skrevet i form av prosedyrer, kontrollstrukturer og variabler.
Høynivåspråkene utviklet seg til at programmeringsyntaksen fikk kjennetegn fra naturlig språk. Dette gjorde det lettere å programmere og det ble mulig å konsentrere seg om den oppgaven man skulle utføre uten å vite alt om hvordan datamaskinen var bygd opp. Man kunne nå programmere høynivå uttrykk slik som

tildel Totalkostnad verdien (Pris + Toll + Moms)

Dette fikk programmeringsverktøyutviklere til å drømme om å gjøre det mulig for mennesker å kommunisere med datamaskiner ved hjelp av abstrakte begrep i stedet for å måtte forholde seg til maskinkode-implementasjonen (Brookshear, 2003).
Problemene med maskinvareavhengigheter var altså ikke helt borte, men tredje generasjons programmeringsspråk var nå på et abstraksjonsnivå som var mer tilpasset menneskers måte å tenke på, og de støttet dermed programmeringsoppgaven bedre ved at man kunne programmere på en mer oppgaveorientert måte.

4.2.3 Byggesteiner i strukturell programmering

Variabler: For å referere til lokasjoner i minnet brukte maskinspråk og assembler numeriske minneadresser. I høyere nivås språk som C, Basic og Cobol brukes variabler, som er beskrivende navn (mnemoniske termer) som rommer ulike verdier. Variabler deklareres før bruk for å gi datamaskinen "beskjed" om hvilken datatype variablen har slik at den kan behandle dataene på riktig måte.
Konstanter: Tekstlig representasjoner av minneområder, som variabler, men som ikke kan forandre verdi.
Datatyper: Angir hvordan variabler kodes og hvilke operasjoner som kan gjøres på dem.
Datastrukturer: Variabler kan også assosieres med en bestemt datastruktur.
Arrayet er eksempel på en datastruktur, det er en rekke enkeltvariabler som det kan gjøres operasjoner på, for eksempel som å legge til eller slette et element (variabel) i arrayet.
Tilordning: Variabelverdier kan kopieres til andre variabler ved hjelp av tilordning. Dette tilsvarer å kopiere verdien i et minneområde til et annet.
Kontrollstrukturer: Uttrykk som forandrer utførelsessekvensen til et program. GOTO utrykket i Basic er en videreføring av assembleruttrykket JUMP som flyttet utførelsen fra et minneområde/register til et annet. Kontrollstrukturer som tilhører tredje generasjonsspråk er for eksempel IF-THEN-ELSE , SWITCH-CASE og WHILE-DO.
Sløyfestrukturer som FOR-DO brukes for å gjenta en utførelse flere ganger og "simulerer" en serie JUMPs i assembler.
Prosedyrer(metoder): er et sett instruksjoner for å utføre en oppgave. Sentral i imperative språk som C for å uttrykke algoritmer. Prosedyrer kan kalles fra ulike deler i programmet og "modulene" kommuniserer dermed ved hjelp av prosedyrekall. Prosedyren er sentral i tredje generasjons språk fordi de er relatert til algoritmebegrepet. Prosedyrer benyttes til å modularisere programmet ved å bygge opp algoritmer som løser hver sine delproblem. Prosedyrer har gjerne underprosedyrer. Slik blir problemet delt inn i mindre problem og er enklere å håndtere.

Parametre: er kjent fra assemblerkode som argument som følger med en instruksjon. Det kan sendes et eller flere argumenter med i et prosedyrekall. Disse argumentene har spesifikke verdier av visse datatyper/datastrukturer. Parametre benyttes i prosedyrer for å gjøre dem mer generelle og dermed øke utrykkskraften.

Med kontrollstrukturer, variabler, datastrukturer etc. rev man seg løs fra den ustrukturerte programmeringsmåten man hadde med maskinkode og assembler, og la grunnlaget for en strukturert programmeringsmåte.

Strukturert programmering ga muligheten for å splitte opp programkoden i kodeblokker eller prosedyrer som kan bli skrevet uten detaljert kunnskap om hvordan andre blokker fungerer. Dette førte til en "top-down" tilnærming der programmet ble stegvis forbedret gjennom en iterativ prosess. Programmereren kunne løse hvert enkelt delproblem for seg selv, og så knytte disse sammen for å løse selve hovedproblemet. Siden mange problemer har samme type delproblem så kunne man gjenbruke kode (Wang, 2000).
Slik ble det lagt til rette for at programmeringoppgaven ble mer i samsvar med den oppgaven som skulle gjøres. Programmeringsoppgaven ble mindre maskinspesifikk og mer orientert rundt oppgaven som skulle løses. Man fikk oversikt og orden i programkoden, noe som gjorde at man kunne lage programmer på en mer effektiv måte.

4.2.4 Innkapsling i moduler og abstrakte datatyper

Strukturert programmering var en effektiv teknikk for mindre programmer, men den var ikke effektiv nok for utvikling av store systemer.
Modulær programmering representerte en videreføring av den top-down oppdelingen i prosedyrer og datastrukturer som strukturelle språk hadde innført. Det nye ved *moduler* var at grensesnittet ble separert fra implementasjonen. Dette var en form for å gjemme bort detaljer, såkalt "information hiding". Det første modulære programmeringsspråket var Modula-2. Programkode og datastrukturer ble innkapslet i moduler som besto av to deler, en definisjonsdel som var det synlige grensesnittet mot andre moduler, og en implementasjonsdel som inneholdt koden til modulen. Dette gjorde det mulig for programmerere å forholde seg til grensesnittet når de skulle benytte moduler, og man kunne på en enkel måte endre implementasjonen uten å gjøre særlige endringer andre steder i programmet.

Denne ideen førte til nye datastrukturer som *abstrakte datatyper*. En abstrakt datatype er en samling metoder/operasjoner som manipulerer en samling dataobjekter.
Moduler og abstrakte datatyper førte til at data og dataoperasjoner ble innkapslet, og den abstrakte datatypen framsto som en enhet med egne data og metoder/operasjoner.

Ikke desto mindre følte man at noe manglet, at kanskje mange av de problemene man opplevde i tradisjonell prosedyral programmering kunne avhjelpes ved en mer helhetlig og balansert holdning til prosedyrer og datamodeller (Maus, 2002).

4.2.5 Objektorientert programmering

Begrepet om abstrakte datatyper ble utviklet videre og objektorientert programmering så dagens lys på 1960-tallet med bla. Simula. Det var likevel først på begynnelsen av 90-tallet at den objektorienterte tankegangen ble dominerende innen systemutvikling (Wang, 2000), da objektorientert design (OOD) fikk sitt virkelige gjennombrudd.

Hovedideen bak objektorientert programmering er at man tar abstrakte begreper fra den "virkelige" verden og lager en modell ut av disse. Begrepene er en slags beskrivelse uten innhold. For eksempel har ikke begrepet menneske øyenfargen blå, men det har en øyenfarge. Begrepet menneske har ikke noe konkret navn, men alle mennesker har et navn. Dette begrepet er beskrivelsen, eller oppskriften. Begrepet "finnes ikke" noe sted. Vi kan ikke peke på det eller ta på det. Det er en mental forestilling inne i folks hoder, som tilsynelatende er ganske lik. Når det lages en modell av den virkelige verden abstraherer man de kjennetegn ved begreper som er slik som de fleste ser det for seg.
I modellen kalles begrepene klasser. *Klassene* er abstraksjoner som beskriver interessante kjennetegn og handlinger til begrepene i den "virkelige" verden (Preece, 1994). De enkelte eksemplarene av begrepene kalles *objekter* og det er knyttet kjennetegn og tilstander til dem som kan manipuleres ved hjelp av de enkelte objektenes metoder.

Klassene er altså oppskriften på hvilken tilstand objektene kan være i og hvordan de kan oppføre seg, og objektene viser hvordan eksemplarene av den og den klassen faktisk ser ut og oppfører seg. Alle konkrete mennesker, som f.eks Arne, Per og Kari er objekter av begrepet, eller klassen, menneske. De er fysiske, eksisterende instanser av det abstrakte begrepet menneske. Objekter er altså ikke passive, men har *oppførsel* (prosedyrer/metoder) som aktivt kan gjøre ting med *dataene*. Det er objektene som er de naturlige enhetene som dataene og prosedyrene blir pakket inn i, og andre objekter forholder seg til et objekt via objektets grensesnitt - metodene. Slik sett blir objektene byggesteiner som i sin tur består av data og måter å behandle disse dataene på.

Innkapsling pakker data og operasjoner som kan gjøres på dataene inn i et objekt, akkurat som en abstrakt datatype. I objektorientert programmering blir ordet klasse benyttet for denne innkapslingen. Objekter er instanser av klassene. Det kan eksisterere flere instanser av den samme klassen, og med arv kan disse objektene ha data og metoder som er definert for klassens overklasse. I tillegg til innkapsling kom nye trekk som arv og polymorfisme.

Arv ga muligheten for at en klasses data og metoder kunne "kopieres" til andre klasser slik at disse slapp å deklarere disse dataene og metodene på nytt og gav mulighet til gjenbruk og utvidbarhet av kode.
Videre bidro polymorfisme til at forskjellige metoder kunne ha samme navn, noe som kunne redusere kompleksiteten og antall kodelinjer ytterligere(Wang, 2000).

Man kan kort og godt si at objektorientering var en ny måte å tenke på, hvor et system bestod av objekter som kunne interagere med hverandre og objektene hadde et velspesifisert grensesnitt. Det ga fokus på objekter som innkapsling av tilstand og oppførsel (Wang, 2000). Slik sett innenbærer objektorientering vel så mye analyse og design som programmering, men prinsippene bak objektorienteringen gjelder alle systemutviklingsfasene, fra analyse og design til selve programmeringen i en gitt utviklingsomgivelse.
Objektorientering innebar altså vel så mye å finne fram til ulike objekter og relasjoner mellom dem, som å implementere hvilke data og metoder objektene skal ha (Wang, 2000).

4.2.6 Komponenter som byggesteiner

Komponent-orientert programmering er et resultat av den objektorienterte tankegangen, og baserer seg på utvikling og gjenbruk av komponenter, prekompilerte objekter med programkode bestående av bla. metoder og data.
Gjenbruk av programkode i komponenter medfører at man kan lage egne komponenter eller benytte andres komponenter for å lage applikasjoner. I stedet for å gjenbruke skrevet kode, gjenbruker man prekompilert programkode i form av fysiske filer. Dette gjør at programkoden er fullstendig skjult, slik at implementasjonsdetaljene ikke er synlige for brukeren av komponenten. Brukeren forholder seg kun til komponentens grensesnitt.
Mens ordinære objekter er hvitboks-objekter med synlig programkode tilgjengelig, er komponenter svartboks-objekter der kun grensesnittet er synlig.

I tillegg til at det ble lettere å gjenbruke funksjonalitet gjorde komponenter at man kunne modularisere programvare i håndgripelige deler. Dessuten ga det muligheten for å benytte komponenter i andre programmeringsspråk eller utviklingsomgivelser enn komponenten var utviklet i.

4.2.7 Dynamic Link Library (DLL)

En DLL består av en samling metoder og variabler og skiller seg fra en vanlig kjørbar fil ved at den ikke er frittstående, men brukes av andre applikasjoner. DLL'ens innhold blir hentet inn ved dynamisk lenking. Ved dynamisk lenking lastes prekompilert programkode bestående av metoder og variabler inn i minnet, adskilt fra den kjørbare fila til applikasjonen som benytter programkoden.
Disse metodene og variablene eksponeres gjennom et veldefinert grensesnitt til applikasjoner som trenger dem. Slik kan en DLL være et middel for å oppnå abstraksjon og gjenbruk. Implementasjonsdetaljer skjules i DLL'en og flere applikasjoner kan gjenbruke den samme DLL'en.

Dette gir en helt ny mulighet til å modularisere applikasjoner, slik at funksjonalitet kan bli gjenbrukt og oppdatert på en lettere måte. Implementasjonen av metodene på DLL'en kan endres uten at applikasjonen som benytter dem trenger å endres, bare metodenes grensesnitt og semantikk forblir uendret. Med DLL'er lastes bare de metodene og variablene inn i minnet som applikasjonen trenger, noe som kan redusere bruken av minne. Dette kan være spesielt nyttig hvis applikasjonen ligger på en datamaskin med begrensede ressurser, som for eksempel en PDA, en mobiltelefon eller en annen mobil enhet.

Å benytte Dynamic Link Libraries kan også være problematisk. For eksempel oppstår det problemer hvis flere DLL'er har samme navn, navnet på DLL'en endres eller dens tilhørende metoder endres, legges til eller fjernes, noe strenge regler for grensesnittets oppbygning kan gjøre noe med.

4.2.8 COM modellen

COM-teknologien gjør det mulig for komponenter utviklet av forskjellige programutviklere å samhandle på en pålitelig og kontrollert måte ved at det defineres et standardisert grensesnitt.

Dette gjør det lett for programutviklere å benytte komponenter fra flere forskjellige programutviklere og til å få dem til å fungere sammen.

COM er utviklet av Microsoft og danner grunnlaget for alle komponenter som er laget for Windows-plattformen. Hensikten var at COM skulle sette en standard for binær gjenbruk på Windows-plattformen, det vil si gjenbruk av programkode og data i form av dynamiske bibliotek.

COM benytter dynamisk linking av prekompilert programkode, i likhet med ordinære DLL'er. I tillegg definerer COM nødvendige regler for oppbyggingen av komponenter og deres tilhørende grensesnitt. For eksempel sier COM-modellen at alle komponenter skal utstyres med en egen ID.
Regler for oppbyggingen av komponenter er viktige for:

- å gjøre komponenter kompatible på tvers av programmeringsspråk og plattformer
- å løse versjonsproblematikken
- å gjøre det lettere å utveksle programkode mellom utviklere.

Når utviklere av COM-objekter tvinges til å følge visse regler nå de lager komponentene, blir det lettere og mer forutsigbart å benytte dem. Grensesnittet blir en slags kontrakt mellom COM-objektet og applikasjoner eller COM-objekter som benytter det. Hvordan COM-objektet løser oppgaven sin spiller ingen rolle, så lenge det gjør det som det lover det skal gjøre. Det at man faktisk får komponentene til å virke er avgjørende.

Det er samspillet mellom de som bygger komponenter og de som benytter dem som senker terskelen for å prøve ut ny teknologi i utviklingsomgivelser (Wang, 2000). Det er altså to hovedaktiviteter knyttet til komponertorientert programmering:

- Bygging av gjenbrukbare komponenter
- Bruk av komponenter i design av programvare

Fokuset i denne oppgaven vil være mer rettet mot bruksaspektet, og mindre rundt hvordan komponenten er bygd opp.

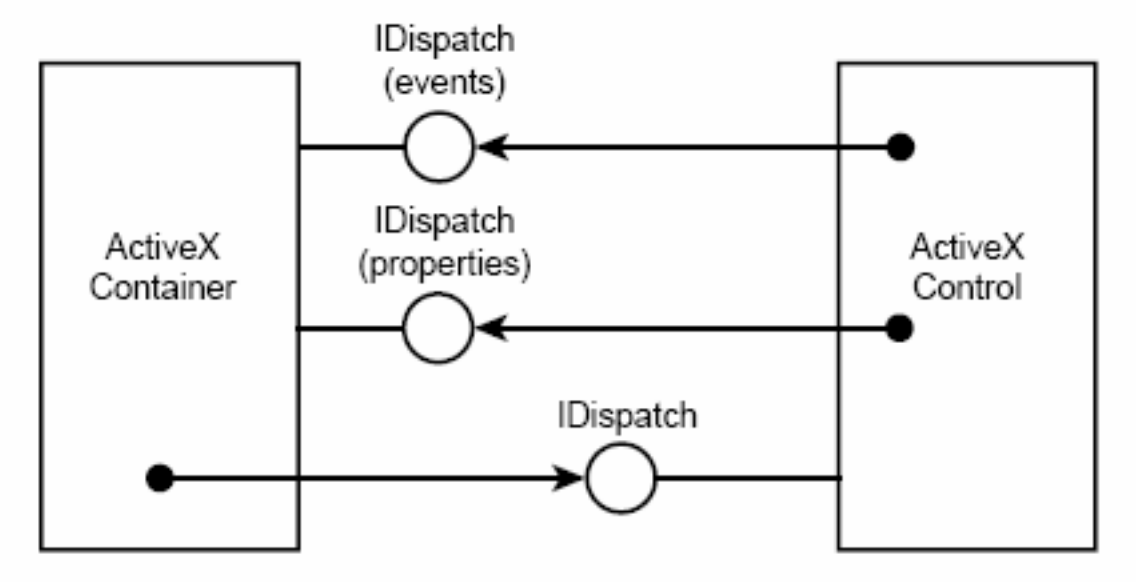
4.2.9 Komponent-teknologien ActiveX

ActiveX-teknologien ble opprinnelig utviklet av Microsoft for å gjøre det lettere å integrere elementer fra ulike Microsoft-applikasjoner.
ActiveX-komponenter er vanligvis grafiske objekter som benyttes i grensesnittet til applikasjoner. De omtales også som ActiveX-kontroller (styreanretninger). Disse programmeres og kompileres vanligvis i programmeringsspråket C++.

ActiveX kan sies å være en sammensmelting av COM og OLE-teknologien.
ActiveX bygger på Object Linking and Embedding (OLE) ved at den benytter OLE-Automation. OLE ble utviklet for å gjøre det mulig for vanlige brukere å lage sammensatte dokumenter som inneholdt elementer fra andre applikasjoner. Man kunne på denne måten "bake" et dokumentelement inn i hoveddokumentet ved å kalle elementets grensesnitt. For eksempel kunne et Word-dokument et inneholde et element fra PowerPoint eller Excel.

En applikasjon som benytter ActiveX-komponenter kalles en kontroll-container.
Visual Basic er en av de mest benyttede containerne for ActiveX komponenter, og bruker ActiveX for å gjenbruke grafiske elementer og funksjonalitet.
Andre kontroll-containere kan være Microsoft-applikasjoner som Word, Excel og Internet Explorer, samt en del andre applikasjoner, blant annet utviklingsomgivelsen Borland Delphi.

Figuren nedenfor viser en komponent (ActiveX-control) som eksponerer sitt grensesnitt til en applikasjon (ActiveX-container). Applikasjonen får da tilgang til komponentens funksjonalitet kun gjennom grensesnittet.



Figur 4.1 ActiveX-grensesnitt

Hvis komponenten er synlig, vil den inngå som del av det visuelle grensesnittet og funksjonaliteten som ligger i den kan programmeres ved å referere til komponentens grensesnitt i programkoden. De grafiske elementene kan ha mer eller mindre funksjonalitet, og kan være alt fra en knapp til en større applikasjon med et bestemt grensesnitt.

En ActiveX-komponent trenger ikke nødvendigvis å være synlig i brukergrensesnittet, den kan være usynlig og kun tilby funksjonalitet. Hvis komponenten er usynlig, må designeren selv bygge opp det visuelle grensesnittet, i tillegg til å referere til komponentens metoder i programkoden. Dette viser at ActiveX gir fleksibilitet til å lage mange ulike komponenter som er velegnet for forskjellige bruksområder.

Jeg vil senere i oppgaven se på hvordan man kan bruke en ActiveX komponent til å få tilgang på Bluetooth-teknologi i en utviklingsomgivelse. For at det skal være mulig å hurtig prototype Bluetooth-teknologien, må ActiveX-grensesnittet være enkelt å benytte. Med dette som utgangspunkt vil jeg se på utviklingen innen fagfeltet menneske-maskin-interaksjon og design av prototyper i utviklingsomgivelser.

4.3 Menneske-maskin-interaksjon (MMI)

4.3.1 Menneske-maskin-interaksjon som fagfelt

Menneske-maskin interaksjon er et ungt fagfelt. Fagfeltet representerer teori for å forstå informasjonsteknologi og og praktiske framgangsmåter for å utvikle datasystemer ut fra denne forståelsen. Allerede fra 1960-tallet har nyvinninger innen ulike forskning-og utviklingsområder lagt grunnen for MMI, men først med forskningssenteret Xerox PARC på 1970-tallet ble disse nyvinningene satt ut i praksis. Menneskers kognitive egenskaper ble lagt under lupen og fokus ble rettet mot menneskene som skulle anvende teknologi. Dette ga grunnlag for usability-prinsipper som står sentralt innen MMI.
Man kan si at betydelig pionerarbeid ble gjort med Xerox PARC og andre offentlig støttede prosjekter på 1970-tallet, som fagfeltet har bygget videre på for å utvikle måter å lage brukervennlige datasystemer på.
Etter dette ble det gjort en rekke andre framskritt innen fagfeltet som konferansen National Bureau of Standards i 1982 og lignende konferanser og workshops på samme tid (Carroll, 2002).

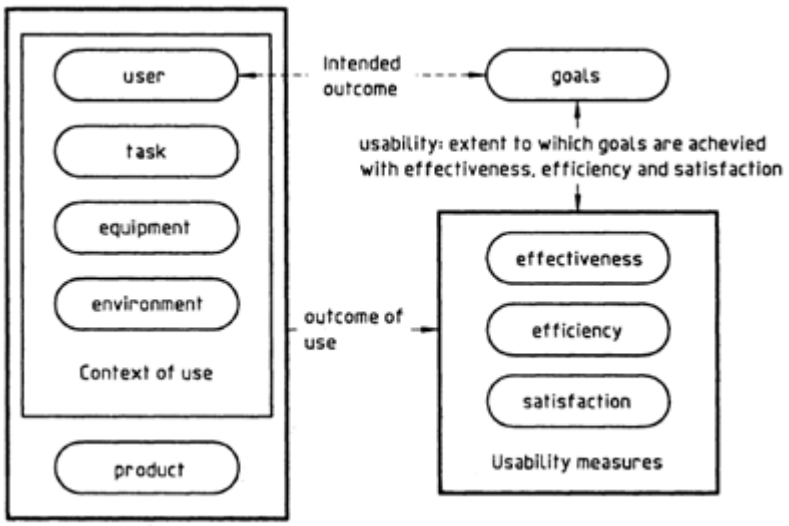
Å utvikle brukergrensesnittverktøy dreier seg om å benytte seg av systemutviklingsmetoder, å utnytte grafikkmuligheter, om interaksjonsteknikker og om å bygge opp begreper og metaforer (Carroll, 2002). HCI handler altså ikke bare om systemutviklingsteori, men også å anvende disse kunnskapene og teoriene til å tilrettelegge datasystemer slik at det er mulig å bruke dem til å få utført oppgaver i ulike sammenhenger. (Carroll, 2002). Med et vel så sterkt fokus på bruksaspektet som på den tekniske siden ved systemene er *usability* et sentralt begrep innen MMI. Det er derfor naturlig å bringe på det rene hva usability er i MMI-sammenheng.

4.3.2 Usability (brukbarhet/brukskvalitet)

ISO's definisjon av usability :

"Usability is the effectiveness, efficiency and satisfaction with which specified users can achieve specified goals in a particular environment"
(ISO 9241.11, 1998)

Usability (brukbarhet) beskriver kort sagt hvor enkelt og tilfredsstillende det er for en bruker å anvende noe for å oppnå et gitt mål.
Figur 4.2 illustrerer de ulike aspektene ved usability.



Figur 4.2 Et Usability-rammeverk (ISO 9241.11, 1998)

For å fastslå brukbarheten til et design er det nødvendig å identifisere brukerens mål og å analysere/dekomponere aspektene *environment(brukskonteks), effectiveness*, *efficiency*, *satisfaction* slik at aspektene kan måles og verifiseres (ISO 9241.11, 1998).
Figuren overfor viser det rammeverket som er knyttet til usability.

Målet vil si det brukeren vil oppnå ved å benytte produktet, for eksempel å få to enheter til å kommunisere.

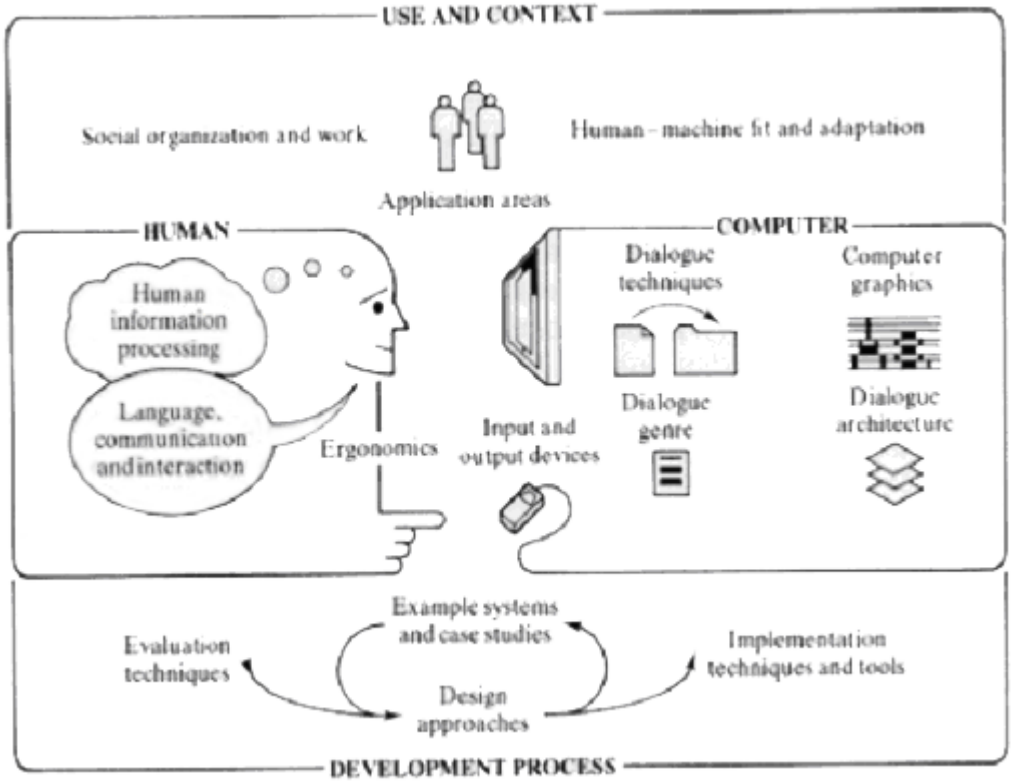
Environment (brukskonteksten) vil si de personlige egenskapene brukeren har (human factors), den oppgaven han skal utføre(task), utstyret han har til rådighet og de fysiske omgivelser rundt brukeren.

Effectiveness er et mål på hvor presist og fullstendig brukerens mål blir oppnådd.

E*fficiency* er et mål på hvor mye som skal til for å oppnå målet i form av anstrengelse, penger, tid og utstyr.

S*atisfaction* sier noe om hva brukerne synes om produktet og om de finner det behagelig.

Usability/brukskvaliteten er dermed, som man ser i figur 4.2, hvor godt man klarer å ivareta hensynet til effectiveness, efficiency og satisfaction for å oppnå brukerens mål.
Usability/brukskvaliteten bestemmes altså ut fra i hvilken grad brukerens mål ble oppnådd og hvor mye innsats som skulle til.



Figur 4.3 Bruk og kontekst (Preece, 1994)

I forbindelse med oppgavens problemstilling er usability først og fremst relevant for å forstå hvilke kvalitetskrav designeren stiller til grensesnittet/designverktøyet.

Designerens målsetning med å benytte grensesnittet kan være å få laget en prototyp med skjermbilder og funksjonalitet enkelt og hurtig. Han vil lage prototypen for å få demonstrert aspekter ved teknologien og dermed få ideer om hvordan teknologien kan anvendes og hvordan brukergrensesnittet kan tilpasses de teknologiske aspektene.

Hensikten med designverktøyet (ActiveX-grensesnittet) som jeg har laget i forbindelse med hovedfagsoppgaven er å se på hvordan man best kan legge til rette for at designeren kan lage funksjonelle prototyper, og å vurdere designers nytte av å få demonstrert funksjonaliteten til teknologien. Jeg ser i den forbindelse på hva som er viktig for å imøtekomme designers krav til brukskvalitet/usability.

4.3.3 Hovedtråder i framveksten av fagfeltet

I følge Carroll har det vært fire underliggende hovedtråder som har bidratt til framveksten av fagfeltet MMI. Disse er :

- Nye metoder for utvikling av programvare fra systemutviklingsteori
- Psykologiske aspekter ved utvikling av programvare fra læringspsykologi
- Utvikling av ny brukergrensesnittprogramvare
- Modeller og teoretisk rammeverk fra forskning på kognitiv psykologi

Jeg vil i de neste kapitlene ta for meg de fire ulike trådene og se på hvilken betydning de har hatt for framveksten av fagfeltet og brukergrensesnittdesign.

4.3.4 Nye metoder for utvikling av programvare

På 1960-tallet la framskritt innen maskinvare til rette for nye programmer som krevde mye større og mer komplisert programvare enn før. Dette forverret problemene med pålitelige og ineffektive datasystemer og kostnads-og tidsoverskridelser. Dette har blitt omtalt som ”programvarekrisen”. Det tvang seg fram et økt fokus på en mer metodisk tilnærmelse til utvikling av datasystemer, og nye metoder for design og utvikling ble sentrale innen elektronisk databehandling (Carroll, 2002).

Dette var starten på systemutvikling som fagdisiplin, med fokus på flere ulike faser i utviklingen av programvare.
Systemutviklingsprosessen gikk fra analyse, kravspesifikasjon og design til implementering og evaluering, der programvaren ble testet opptil flere ganger underveis i utviklingsprosessen. En slik faseinndelt systemutviklingsprosess går under betegnelsen fossefallsmodellen. Man innså imidlertid at det var nødvendig med flere iterasjoner etter hvert som systemet tok form, der man gjennomførte faser i prosessen på nytt for å gjøre forbedringer.

Den konvensjonelle fossefallsmodellen, der programmet blir spesifisert, implementert og testet i ulike separate faser var utilstrekkelig. I konkrete systemutviklingsprosjekter vil disse fasene ville disse fasene nødvendigvis gå om hverandre (Myers, 1994). For å utvikle produkter som var mest mulig i samsvar med det brukerne eller markedet generelt ville ha, måtte man kunne gjøre større eller mindre endringer i designet/utformingen underveis. Det ble utviklet prototyper, uferdige versjoner eller utkast av de ferdige systemene, som så ble testet på aktuelle brukere.

4.3.5 Psykologiske aspekter ved utvikling av programvare

Man innså at for å lage større og mer kompliserte systemer måtte man ha flere og bedre skolerte programmerere som fikk utført oppgavene mer effektive. Til da hadde programmering vært forbeholdt de få, og assemblykode, datidens programmeringsspråk,

krevde detaljkunnskap om datamaskinens oppbygning, som nevnt i kapittel 4.2. Hvis man gjorde det enklere å programmere, ville man kunne gjøre noe med dette problemet.
Det ble økt interesse rundt programmeringsoppgaven og kunnskap fra lingvistikk ble brukt for å finne ut hvordan programmerere kunne kommunisere med datamaskiner. Språklig syntaks og semantikk ble studert og forsøkt overført til programmeringsspråk. På 60-tallet kom nye programmeringsspråk som Algol og på 1970-tallet kom objektorienterte språk som Smalltalk og Simula. Disse tok i bruk språklige mekanismer for å gjøre det enklere og mer naturlig å programmere.
Abstraksjonsnivået ble løftet opp på et nivå som passet mer naturlig til problemløsningen (se kapittel 4.2.5).
Fokuset ble altså rettet mot programmereren og programmeringsoppgaven, fordi man innså viktigheten av en mer effektiv interaksjon mellom menneske og maskin for å kunne lage mer kompliserte systemer.

I løpet av 1970-tallet ble det dannet ulike forskningsinstitusjoner for programvare og de første brukbarhetslaboratorier ble opprettet. Disse benyttet fastlagte metodiske retningslinjer for forskning (se kapittel 4.3.1 og 4.3.4).

4.3.6 Utvikling av ny brukergrensesnittprogramvare

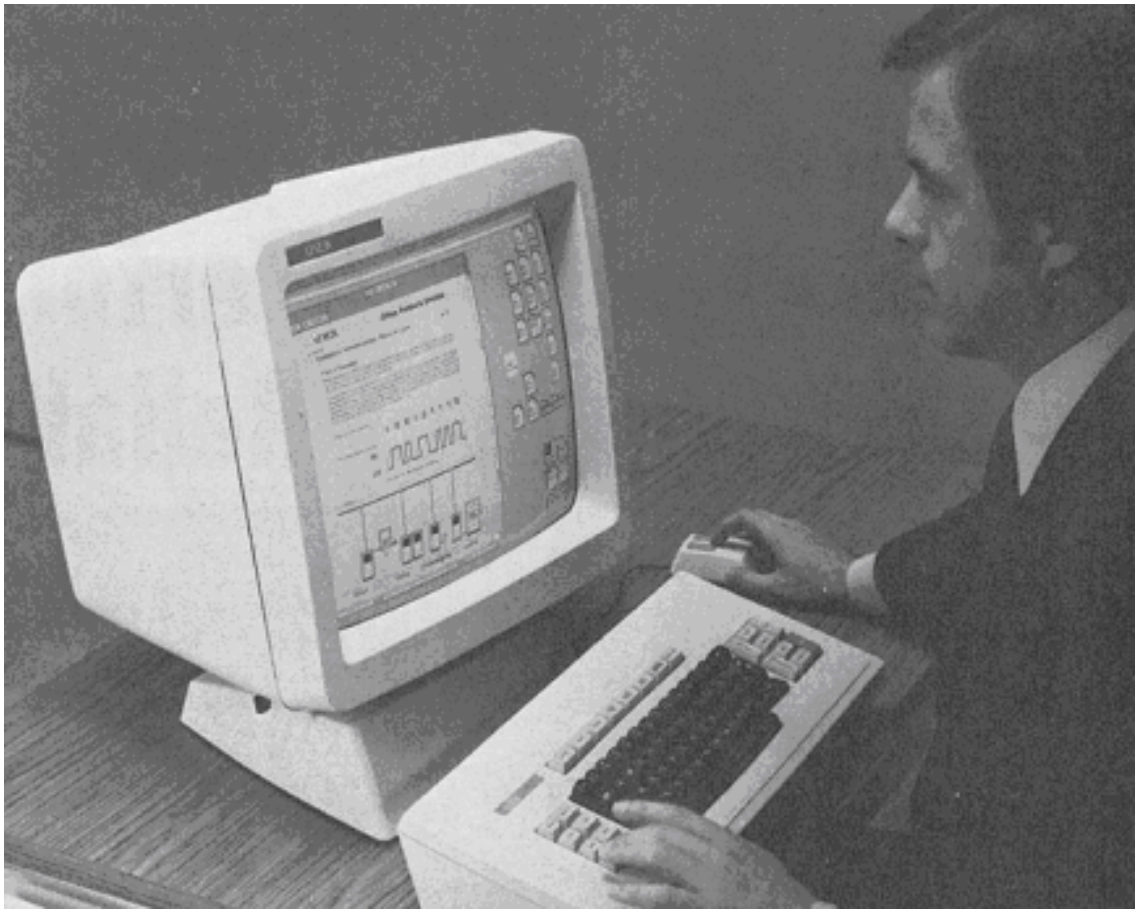
Tidlige visjoner om å gjøre elektronisk databehandling tilgjengelig for den ”vanlige” bruker ble satt ut i livet gjennom samarbeidsprosjekter mellom universiteter og næringsliv.

I 1970 etablerte Xerox forskningssenteret PARC i Palo Alto, USA for å lage framtidens dataløsninger for elektronisk kontorstøtte. Xerox var (og er) en ledende produsent innen kontorutstyr som skrivere, faxmaskiner og andre kontorautomatiseringsutstyr.

Xerox PARC tok ibruk nye systemutviklingsmetoder som innebar utvikling av prototyper og brukbarhetstester. Etter to års forskning og utvikling presenterte PARC kontordatamaskinen Alto. Den hadde bitmapgrafikk, noe som gjorde det mulig å ha grafiske brukegrensesnittelementer som vinduer, ikoner, dialogbokser osv. på skjermen. For brukeren så datamaskinens brukergrensesnitt ut som et skrivebord der ikoner og ”widgets” ble manipulert direkte ved hjelp av en nyvinning kalt mus. Ikoner på skjermen – skrivebordsdokumenter, kunne maksimeres til egne vindu hvor brukeren kunne gjøre operasjoner på grafiske elementer som tekst, grafikk, tabeller osv. Dette kom til å danne grunnlaget for brukergrensesnittet til mange av dagens operativsystemer og applikasjoner.

Ideen var at datasystemet skulle kunne brukes av vanlige personer og at disse skulle kunne få utført sine oppgaver uten å være spesielt interessert i datamaskiner eller ha noen særlig kunnskap om hvordan de var bygd opp. Derfor var det en viktig målsetning at datamaskinen var så ”usynlig” og så lett å lære seg å bruke som overhode mulig, slik at brukeren slapp å forholde seg til hvordan datamaskinen virket, og bare kunne konsentrere seg om å bruke den til utføre sine arbeidsoppgaver.
Mange av de viktigste interaksjonsteknikkene som anvendes i brukergrensesnitt med direktemanipulasjon, for eksempel måten skjermobjekter og tekst blir valgt, åpnet og manipulert på, ble utforsket på Xerox PARC på 1970-tallet (Myers, 1998).
Et viktig bidrag fra forskningssenteret PARC var Bravo, en teksteditor beregnet for Altomaskinen, hvor tekststørrelser, teksttyper og andre tekstlige atributter kunne redigeres på

direkte på skjermen. Bravo og tegneprogrammet Draw, også fra Xerox PARC, var opprinnelsen til begrepet ”What you see is what you get” (WYSIWYG).
Bravo og Draw utnyttet Altos nye bitmapskjerm. Men selv om skjermen kunne deles opp slik at ulike dokumenter eller deldokumenter kunne redigeres samtidig, hadde ikke teksteditoren på langt nær de samme egenskapene som det vi ser i dagens windows-systemer. Den banet imidlertid vei for senere grafikk og tekstbehandling i programmerings – og prototypingsomgivelser.



Figur 4.4 Xerox Star fra 1981

I 1981 kom Xerox Star, bygd på prinsippene og erfaringene fra Alto og den første datamaskinen beregnet for personlig bruk. Mens forgjengeren Alto var resultatet av et forskningsprosjekt, ble Star lansert som et kommersielt produkt med en overkommelig pris, maskinen kostet $16,500 med ferdig installert programvare.

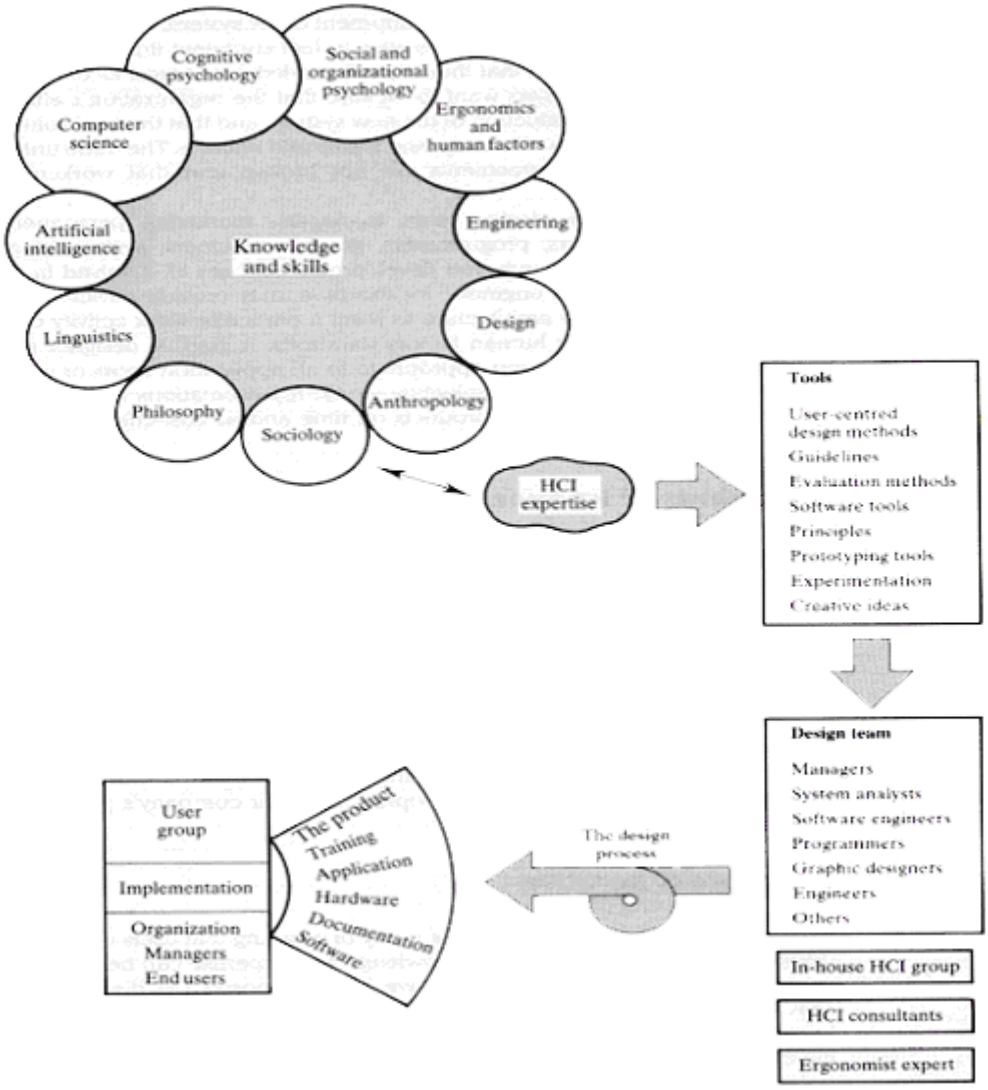
Forskningssenteret brukte Alto til å utvikle en interaktiv programmeringsomgivelse som kunne tilby brukere av maskinen de nødvendige byggesteinene for å bygge sine egne programmer. Denne programmeringsomgivelsen fikk navnet Smalltalk og basert seg på objektorientert tankegang der grafiske elementer på skjermen ble betraktet som objekter og kunne programmeres. Målet var å vise at man kunne programmere datamaskiner ved hjelp av

brukergrensesnittets "språk" (The Xerox Star: A Retrospective, 2004). At man på Alto - og stormaskinene kunne bruke en programmeringsomgivelse som Smalltalk til å programmere grensesnittet demonstrerte fordelene ved grafiske bitmap skjermer, mulighet for ulike applikasjoner i ulike vinduer og mus som arbeidsredskap. Man abstraherte vekk datamaskin-spesifikke begreper til fordel for å la brukeren utføre oppgaven sin kun ved å forholde seg til grafiske objekter på skjermen.
Smalltalk fremmet forestillingen om personlige datamaskiner, banet vei for integrerte og interaktive programmeringsomgivelser, og tok i bruk begreper innen objektorientert systemutvikling som til da kun hadde vært til stede i mindre grad i utviklingsomgivelser (The Xerox Star: A Retrospective, 2004).

Å programmere og benytte disse objektene ved hjelp av direkte manipulasjon ble to sider av samme sak. Man kunne styre og programmere maskinen ved hjelp av et brukergrensesnitt som var lettfattelig og intuitivt. Dessuten ble funksjonaliteten og brukergrensesnittet adskilt. Brukerne utførte sine arbeidsoperasjons mot grensesnittet og ikke direkte mot operativsystemet. For programmererne sin del ble det flere adskilte lag, brukergrensesnittet og funksjonaliteten til applikasjonen var i hvert sitt lag, et prinsipp som programmererne kunne dra nytte av og som gjorde det enklere å programmere (Carroll, 2002).

4.3.7 Kognitive modeller og teoretisk rammeverk

På slutten av 1970-tallet vokste kognitiv psykologi sammen med elementer fra lingvistikk, filosofi, psykologi og informatikk til et tverrfaglig prosjekt som var med å danne grunnlaget for MMI som fagfelt. Et prinsipp innen kognitiv forskning var at tverrfaglig forskning skulle komme til nytte ved at reelle håndfaste problemer ble løst. Den opprinnelige visjonen til MMI som anvendt forskning var at metoder og teori fra kognitiv psykologi skulle ha betydning for utvikling av programvare (Carroll, 2002).
Forskningsprosjektet Xerox PARC er et godt eksempel på en slik tverrfaglighet. Mange av nyvinningene innen MMI har vært preget av forskning som er gjort ved universiteter og forskningslaboratorier. Det ble sprøytet offentlige midler inn i forskningsprosjektet Xerox PARC. PARC var et samarbeid mellom universitetsmiljøer, forskningsavdelingen til Xerox og "kommersielle interesser". Uten denne satsningen ville Star med sine nyvinnende interaksjonsteknikker sannsynligvis ikke blitt realisert (Myers, 1998).



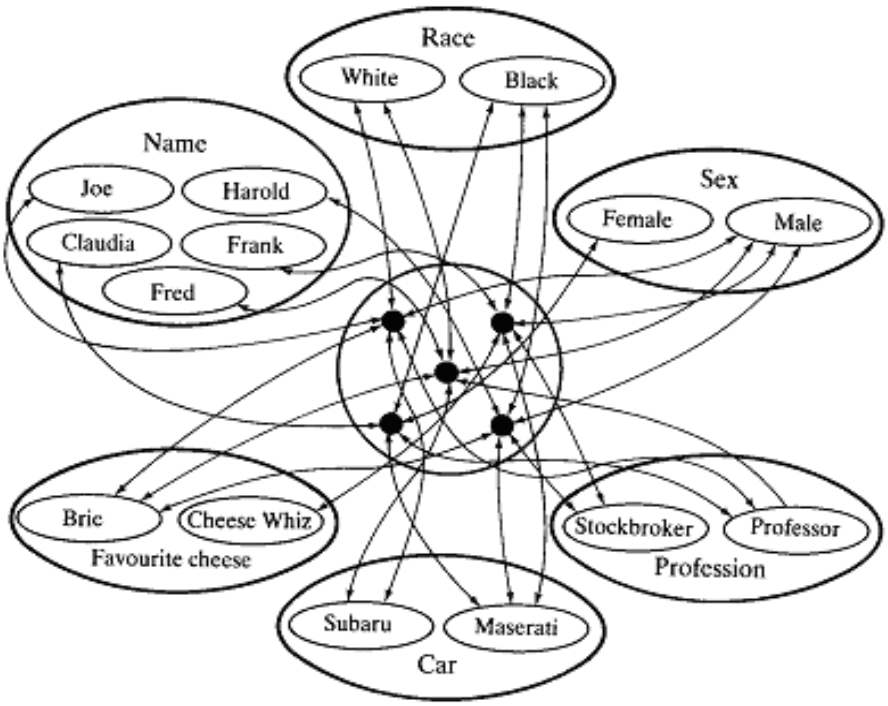
Figur 4.5 Den tverrfaglige designprosessen (Preece, 1994)

Prosjektet tok modeller og teoretisk rammeverk utviklet ved universitetsmiljøer og brukte denne kunnskapen til å utvikle programvare bygd på prinsipper om brukervennlighet og anvendelse. De kommersielle aktørene i Xerox var intereressert i brukervennlige løsninger slik at de kunne få solgt så mange datamaskiner som mulig. Hvis det skulle la seg gjøre å selge kontordatamaskiner i stort volum, måtte de satse på den store gruppen av mulige brukere rundt om på kontorene som ikke hadde så mye kunnskap om datamaskiner. For å få potensielle brukere til å greie å bruke maskinene måtte de anvende kunnskap om hvordan den enkelte bruker tenkte og hvordan de tok teknologien i bruk.
Forskning på kognitiv psykologi og 'Human factors' var interessant i så måte, og dannet et teoretisk grunnlag for måten Xerox PARC brukergrensesnittet til Alto/Star var bygd opp på.

Kunnskapsmodeller var resultat av denne tverrfaglige forskningen med utgangspunkt i kognitiv forskning. Kognitiv forskning har i denne sammenheng sett på hvordan kunnskap blir representert i minnet og hvordan kunnskapen som en person har tilegnet seg senere kommer til nytte i problemløsning. Det er enighet om at kunnskapen blir strukturert på en

eller annen måte. Informasjon blir husket ved at personen assosierer ulik kunnskap med hverandre ut fra hvilken semantisk betydning informasjonen har. På hvilken måte informasjonen blir representert i minnet er det imidlertid delte meninger om.

Konneksjonistisk tilnærmelse sier at informasjon som det er en semantisk sammenheng mellom grupperes i egne noder etter assosiasjon. Forbindelsen mellom ulike grupper av informasjon som hører sammen blir i sin tur assosiert med andre grupper, slik at oppstår et nettverk av noder – et semantisk nettverk.



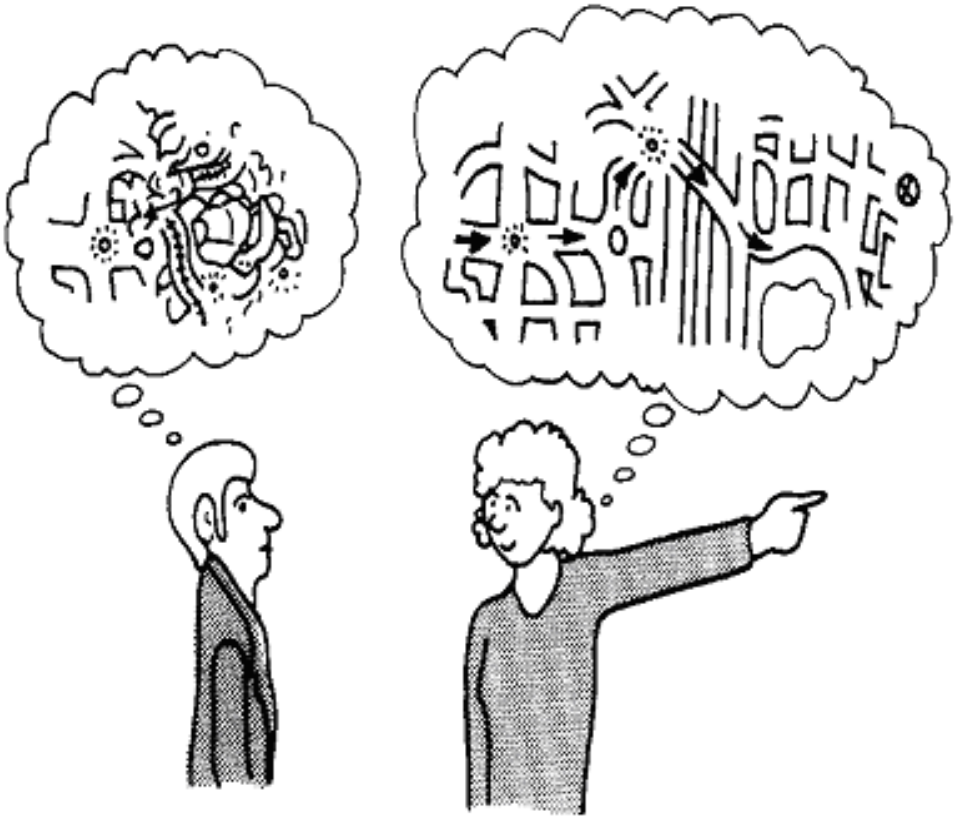
Figur 4.6 Semantisk nettverk (Preece, 1994)

En lignende teoretisk tilnærming til kunnskapsrepresentasjon er begrepet om kunnskapsskjema; et nettverk av generell kunnskap på grunnlag av tidligere erfaring, som blir memorert ved at bestemte erfaringer og handlinger blir gjentatt flere ganger. I likhet med den konneksjonistiske modellen blir kunnskap også i følge denne modellen organisert etter assosiasjon, men det fokuseres mer på hva man gjør i den og den sammenhengen og i hvilken rekkefølge. Kunnskap blir altså representert i form av regler –skjemaer, som assosieres med visse situasjoner. For eksempel kan en handletur i en forretning assosieres med å planlegge hva eller hvor man vil handle, ha på seg passende klær, dra til forretningen, gå rundt på utkikk etter varer, betale og så komme seg hjem igjen mer eller mindre fullesset med tunge poser.

4.4 Konseptuelle modeller og bygging av metaforer

4.4.1 Mentale modeller

For å kunne vite hvordan de skal bruke ting eller for å forstå hvordan tingene virker, kan det være til hjelp å danne seg indre bilder av disse tingene.
Slike bilder kan være basert på tidligere erfaringer med disse tingene eller med lignende ting. Hvis det ikke foreligger noen forestillinger fra før av om hvordan en ting er, bygges det gjerne opp et nytt bilde av tingen etter hvert som man får mer erfaring med den.
Slike indre bilder av ting kalles mentale modeller og er sentralt i fagfeltet MMI for å forstå hvordan folk forholder seg til og samhandler med datasystemer.
Mentale modeller kan defineres som den indre modellen folk har om seg selv, andre folk, omgivelsene og ting de interagerer med (Norman, 1988).
Teorien om mentale modeller er sentral i MMI og er tatt fra kognitiv psykologi.



Figur 4.7 Mentale modeller (Preece, 1994)

Som kunnskapsskjema sier mentale modeller at det dannes et nettverk av generell kunnskap på grunnlag av tidligere erfaringer. Disse erfaringene blir gjort fortløpende og blir representert i minnet ved at informasjon blir forbundet med hverandre og lagt i grupper. Modellene kan endre seg dynamisk etter hvert som det blir gjort nye erfaringer.
Folk har en tendens til å løse problemer ved å benytte kunnskap om lignende problemer hvis det er likhet mellom den tidligere kunnskapen og det problemet som skal løses. En mental modell dannet fra tidligere erfaringer kan altså benyttes, noe som kan lette problemløsningen.

Mentale modeller kommer ofte til nytte når man møter noe nytt som man må gjøre en antagelse om. Da kan det være til hjelp å ha dannet seg et indre bilde av en lignende fenomen og så forholde seg til det nye fenomenet som om det er dette man kjenner til.

4.4.2 Strukturelle og funksjonelle mentale modeller

Det er to typer mentale modeller som benyttes for å internalisere kunnskap.
Strukturelle mentale modeller beskriver den informasjonen en person trenger for å vite hvordan noe er bygd opp. Fordelen med dette er at hvis man vet hvordan noe er bygd opp, kan man forstå hvordan man bruker systemet på ulike måter, og ikke bare knyttet til en bestemt sammenheng. Det blir dessuten enklere å forutsi hvilken virkningen av enhver rekkefølge av handling som utføres på systemet. Ulempen er at det kan kreve en veldig stor innsats å danne seg en modell av hvordan noe er bygd opp.

Funksjonelle mentale modeller innebærer på den andre side prosedyral kunnskap. De beskriver hvordan man går fram for å løse en oppgave. Dette kan sammenlignes med kunnskapsskjemaer, der kunnskap blir internalisert i minnet som en ”oppskrift” på hvordan man gjør ting.

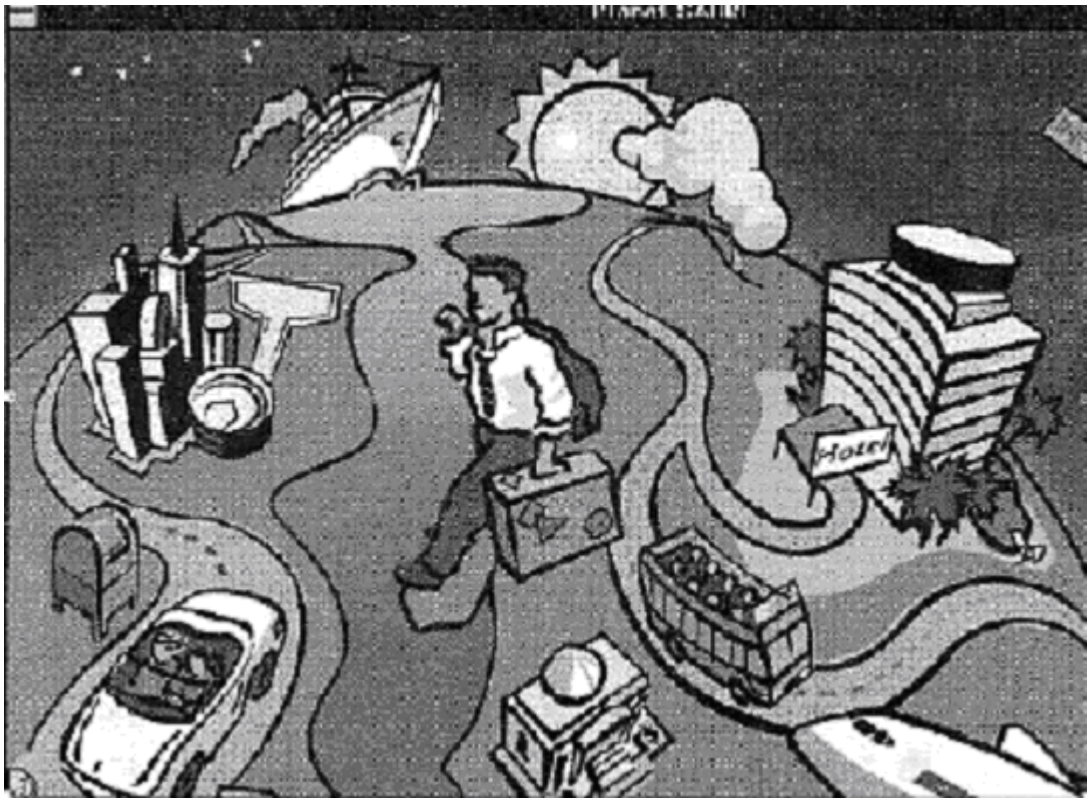
Kort fortalt kan man si at mens en strukturell modell beskriver *hvordan* et system virker, nøyer en funksjonell modell seg med å beskrive *hva* systemet gjør.

Fordelen med funksjonelle mentale modeller er at modellene kan dannes ut fra eksisterende kunnskap om et fenomen og at de er lette å lære. En ulempe er at de er kontekst-avhengige. Strukturelle modeller kan lettere overføres mellom ulike kontekster (Khella, 2002).

Når en person setter seg inn i en teknologi benytter han gjerne både strukturelle og mentale modeller. Han forholder seg for eksempel til en bruksanvisning – som gir funksjonell og prosedyral kunnskap, samtidig som han gjennom erfaring med en lignende teknologi har en viss forståelse av hvordan teknologien er bygd opp.

4.4.3 Metaforer

Metaforer er begrep som brukes i overført betydning, altså ord eller uttrykk som som får en ny betydning når de blir ”lånt” til en annen sammenheng enn de opprinnelig var ment for.
Å bruke metaforer kan også beskrives som det å snakke i bilder. De fleste språk er fulle av slike billedlige uttrykk og i skjønnlitteratur, poesi og sang er det ganske vanlig å bruke ord og uttrykk som egentlig har en annen betydning.
Metaforene er som oftest abstrakte begrep som har en betydning som folk flest kjenner til. Det er derfor ofte nærliggende å uttrykke noe ved hjelp av metaforer som det ellers hadde vært vanskelig å få sagt fordi fenomenet er for abstrakt eller er ukjent.



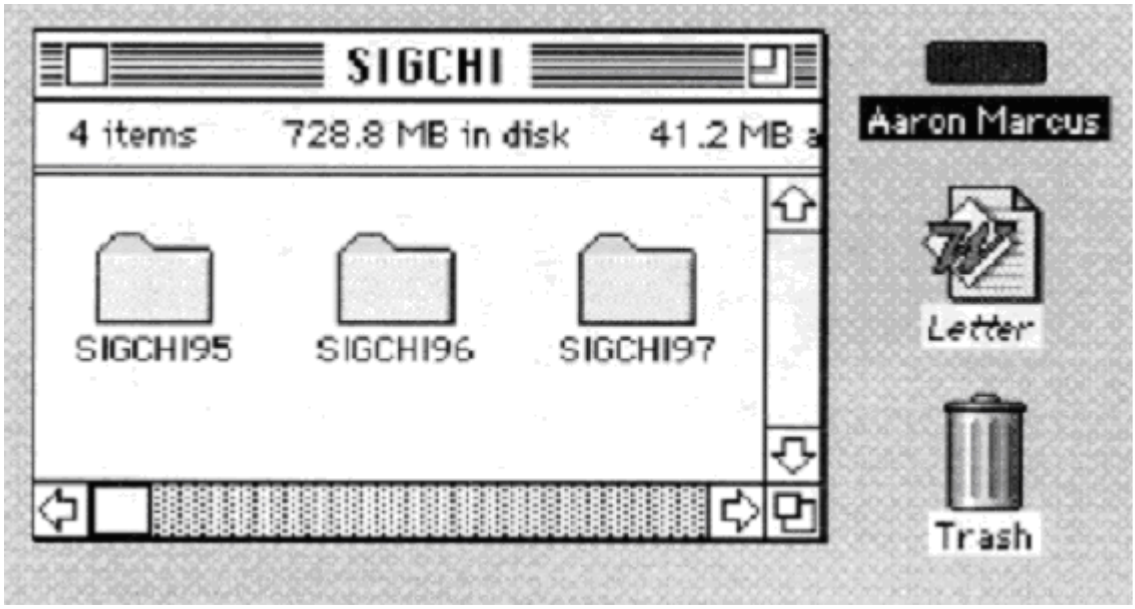
Figur 4.8 Bruk av metaforer

4.4.4 Grensesnittmetaforer

En av de første som benyttet metaforer i brukergrensesnittet var Xerox som med sin Star maskin innførte skrivebordsmetaforen (desktop metaphor).
Selv om Star teknisk sett var resultat av nyvinninger innen maskinvare, datagrafikk og IO-utstyr, var det ideene bak skrivebordsmetaforen som var mest banebrytende for utviklingen innen brukergrensesnitt. Da forskningssenteret PARC utviklet Alto tidlig på 1970-tallet – forløperen til Star, begynte utviklerne å se for seg betydningen av å skape en tydelig og forståelig modell som kunne danne utgangspunktet for designet (Winograd, 1995).

Selve brukergrensesnittet til Star var bygd på en metafor der man ga brukerne en ”illusjon” av at skjermen var et skrivebord på et kontor. Star var ment å være et hjelpemiddel for kontorautomatisering og tanken var derfor å skape en grensesnittmetafor som for brukeren av maskinen opplevdes som de fysiske gjenstandene på et kontor. Derfor så utviklerne for seg at skrivebordsmetaforen kunne være passende. Begrep som filer og filsystem, maskinvare osv. var gjemt bort og erstattet av elementer som mapper og søppelbøtte, i form av grafiske ikoner på skjermen. Man utførte oppgaver ved å peke og klikke på skrivebordsikoner med mus som arbeidsredskap. Utviklerne så for seg at mapper, papirer og andre ”nyttegjenstander” var metaforer for ikonene på skjermen. Behandling av skjermelementer ved hjelp av mus kalles direkte manipulasjon og kan sies å være en metafor i seg selv. Brukeren manipulerer skjermelementer og er ”lykkelig” uvitende om det som skjer rent teknisk på maskinen, filbehandling, input/output redskap osv. Operativsystemet tar seg av dette uten at brukeren er klar over hva som egentlig skjer og det å peking og klikking er en metafor på å bruke hendene for å manipulere skrivebordet.

Prinsippet om direkte manipulasjon er en integrert del av skrivebordsmetaforen.
At brukerne av maskinen allerede har et grunnlag fra det "virkelige liv" for å forstå hvordan de behandler skrivebordselementer ved å ta og flytte på dem, gjør det lettere å få dem til å lære å bruke datamaskinen og gjør det mulig å utføre arbeidsoppgavene ved hjelp av datamaskin mer effektivt (Carroll, 2002).



Figur 4.9 Et grafisk brukergrensesnitt med skrivebordsmetafor

Det som skiller grensesnittmetaforer fra vanlige metaforer er at mens vanlige metaforer lar brukeren assosiere begrep på andre områder med noe i systemet, er grensesnittmetaforen selve illusjonen som brukerens mentale modell bygger på. For eksempel er kontoret som brukes i Star en grensesnittmetafor, og ikke bare en vanlig metafor, fordi metaforen *er* selve modellen som blir skapt. Ideen er at begrepet i den opprinnelige sammenhengen og det "adopterte" metaforiske begrepet smelter sammen til et begrep (Preece, 1994). Kontorbegrepet som brukeren kjenner fra sine erfaringer med kontorarbeid og begrepet slik han opplever det når han bruker datamaskinen blir til *en* entitet, og utgjør begrepet som er grunnleggende for illusjonen brukeren har av at han driver "kontorarbeid". Egentlig sitter han jo bare foran en dataskjerm med mus og tastatur.

Å finne passende metaforer kan være vanskelig når man skal designe et system. Det er viktig å utnytte brukerens kjennskap til familiære begreper, eller i det minste å være konsistent når man presenterer metaforer. Jo flere begrep og metaforer, jo verre kan det bli å for brukeren å få oversikt. Spesielt kan det skape forvirring hvis det eksisterer flere lignende begreper som beskriver den samme tingen.
Det er altså viktig at brukere ikke bare forstår de enkelte begrepene, men at de forstår den sammenhengen mellom begrepene og hvordan de kommer til uttrykk i praksis. De trenger et oversiktlig bilde av hvordan de bruker teknologien.

4.4.5 Konseptuelle modeller

Konseptuelle modeller er overordnede beskrivelser av hvordan noe virker eller er organisert. Når mennesker interagerer med omgivelsene sammenligner de gjerne med tidligere erfaringer som gjør det lettere å løse et problem eller forholde seg til et nytt fenomen. Den erfaringen de får med gitte fenomener danner grunnlaget for den mentale modellen de benytter når de forholder seg til noe nytt (se kapittel 4.4.1.).
Når selve forestillingen eller modellen av noe bygger på metaforiske sammenligninger er dette typisk en konseptuell modell. Skrivebordsmetaforen er et eksempel på en metafor som også er en konseptuell modell. Den inneholder ulike begreper som brukere kjenner fra andre sammenhenger.

Konseptuelle modeller beskriver :

1. Den viktigste designmetaforen og andre viktige metaforer og analogier brukt i designet

2. Alle begrep som systemet eksponerer til brukere

3. Forholdet mellom disse begrepene, eks.: ei mappe inneholder dokumenter

4. Sammenhengen mellom begrepene og deres fysiske representasjon, altså oversettelsen (mapping) fra abstrakte begrep slik brukere og designere ser det for seg, til hvordan de uttrykkes på skjermen eller i datasystemet som f.eks widgets, filsystem.

(Johnson og Henderson, 2002, s. 26)

En annen definisjon av konseptuelle modeller:

Konseptuelle modeller er den generelle betegnelsen som beskriver hvilken forståelse brukere og designere har av systemet og hvilke begrep de danner seg
(Preece, 1994, s. 151).

Hovedutfordringen for designere er å utforme systemet slik at det gir en konsistent begrepsdanning – en designmodell, og å legge til rette for at brukeren kan danne seg en mental modell av systemet – en brukermodell, som er i samsvar med denne designmodellen (Norman, 1986).
Det er viktig at grensesnittmetaforen og andre viktige begrep som designerne bruker er forståelige, klare og entydige slik at brukerne lettere kan danne seg en oppfatning av systemet som stemmer overens med det ”imaget” designerne ønsker å presentere.

For at brukerne skal kunne forstå hvordan de kan benytte de ulike begrepene/ objektene i systemet, bør designerne utforme en konseptuell modell som passer godt overens med den mentale modellen av systemet de tror brukerne har.
Det er av avgjørende betydning at den konseptuelle modellen sier brukeren noe om hvordan han kan benytte seg av de ulike begrepene som presenteres – den fysiske mappingen bør være klargjørende (jvf. Johnson og Hendersons punkt 4 for konseptuelle modeller). Dessuten bør det komme fram hvor generell eller spesifikk de enkelte begrepene/objektene i modellen er, slik at brukerne for eksempel vet om en metode er aktuell å bruke eller om den har et for

begrenset bruksområde. Det er viktig å synliggjøre i designet hvilke begrensinger og muligheter som ligger i begrepene som utgjør brukers mentale modell.
Slik kan brukerne få en formening om hvordan de forbinder/mapper begrepene i modellen med det de faktisk kan gjøre fysisk med dem på skjermen.

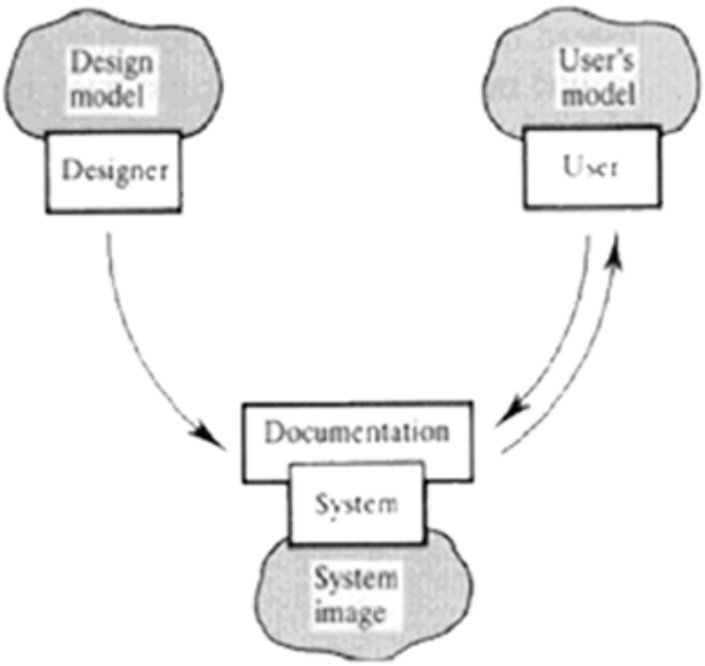
4.4.6 Den konseptuelle modellen i Macintosh-grensesnittet

Macintosh-grensesnittet har siden den første Macintosh-maskinen kom på markedet (på 1980-tallet) bestått av brukergrensesnittobjekter som kunne håndteres direkte ved hjelp av en mus, basert på prinsippene bak grensesnittet til Xerox Star. Ideen er at folk kan kommunisere med deres omgivelser gjennom fysiske handlinger som gir umiddelbar respons i grensesnittet. Disse objektene, slik som for eksempel knapper, scrollbars og vinduer, har et standard utseende og reagerer på bestemte handlinger med mus-verktøyet (TOG on Interface, 1992). Macintosh har altså tatt i bruk prinsippene fra Xerox PARC som dreier seg om å bygge systemets grensesnitt rundt en konsistent konseptuell modell. Denne modellen danner grunnlaget for å tilby et forutsigbart og konsistent designspråk. En viktig grunn til at Mac ble så populær er at man fikk systemet til å virke enkelt å bruke gjennom et uniformt grensesnitt (Winograd, 1995).
Brukerne bør altså danne seg en overordnet oppfatning av et system før de bruker det, på samme måte som designere bør utforme en overordnet konseptuell modell før de tar fatt på designdetaljene. Videre så er det om å gjøre at de skjønner hva deres forståelse av systemet konkret innebærer for den oppgaven de skal løse. Det kan sies å være en god 'mapping' mellom det konseptuelle og fysiske designet hvis den virker naturlig og intuitiv for brukere (Preece, 1994). På denne måten kan designet virke selvforklarende slik at det oppmuntrer til å gjøre de riktige tingene (affordance) og hindrer brukere fra å gjøre handlinger som ikke designer ikke vil at de skal gjøre (constraints). For å lykke med dette er det viktig med et felles designspråk. Systemet blir forutsigelig og brukeren kan dra nytte av ervervet erfaring gjennom uniformitet i et konsistent grensesnitt.

4.4.7 Å kommunisere konseptuelle modeller

Det er gjennom brukergrensesnittet at designerne kommuniserer den konseptuelle modellen. Brukergrensesnittet er oversettelsen mellom abstrakte begrep i den konseptuelle modellen og den konkrete manifestasjonen av den - i form av knapper mapper og ikoner (Johnson og Henderson, 2002). Begrepene og metaforene er fysisk representert ved blant annet widgets, ikoner og vinduer. Brukerne interagerer med systemet gjennom dets look-and-feel og det som skjer når de for eksempel trykker på den og den knappen, samt gjennom å lese dokumentasjon (særlig hvis de står fast). Dette kaller Norman for "system image" (Norman, 1986). Dette "imaget" sier noe om hva brukeren kan gjøre med de forskjellige objektene i systemet, samt - hvis det er nødvendig – i hvilken rekkefølge ting skal gjøres. Hva som skjer (hendelsene) når brukeren interagerer med objektene - altså den konkrete manifestasjonen av begrepene i modellen, er også del av "system image".
Det er derfor viktig å være klar på overgangen fra konseptuelt design til fysisk design, slik at brukergrensesnittdesignet gjenspeiler den konseptuelle modellen så nøyaktig og konsist som mulig, og dermed gjør det enklere for brukeren å danne seg et riktig bilde av hvordan han kan

bruke systemet. Et system som er designet slik at det er forutsigbart hva som skjer når man gjør operasjoner på det, vil således være lettere å lære og mer behagelig og intuitiv å bruke.



Figur 4.10 Konseptuell modell (Preece, 1994)

4.4.8 Hensikten med konseptuelle modeller

Designeren har oppgaven med å tenke ut en konseptuell modell som gir mening for brukerne, ut fra den arbeidsoppgaven de har tenkt å utføre (problemområdet). Den konseptuelle modellen kan derfor danne grunnlag for den mentale modellen til brukerne, men dette trenger ikke nødvendigvis være hovedhensikten. Det kan også være nyttig å betrakte en konseptuell modell som et designhjelpemiddel - en måte for designere å rydde opp i tanker på og finslipe ideer før de begynner å utforme designet.

Den konseptuelle modellen tjener altså flere hensikter. Modellen har en viktig kommunikativ rolle, både overfor designerne selv, framtidige brukere av et system og andre folk som er tilknyttet utviklingsprosjektet, og kan være et hjelpemiddel for å synliggjøre designet og de designvalgene som kan tas.

I praksis er det imidlertid ikke alltid at brukeren tar til seg alle elementene av designer-modellen, slik at han utnytter alle muligheter som systemet gir. Brukers forståelse av systemet kan være begrenset i forhold til det designeren har sett for seg at bruker kan gjøres med systemet.
Ved for eksempel å lese dokumentasjon, sammenligne med lignende erfaringer eller snakke med andre brukere av et system, kan brukere danne seg en mentalt modell av hvordan det fungerer eller hvordan de kan bruke systemet. Dette gjør det mulig for dem å forutsi hvordan systemet oppfører seg og dra nytte av denne kunnskapen ved senere anledninger. Hvis designere tar seg bryet å utforme en konseptuell modell for systemet *før* de designer

brukergrensesnittet, vil brukere lettere kunne danne seg et bilde av systemet. Dessuten vil den modellen de danner seg av systemet bli mer lik det designerne hadde sett for seg.
En konseptuell modell er derfor et idealisert bilde av hvordan systemet skal være, det vil si den modellen designerne håper brukerne skal ta til seg (Johnson og Henderson, 2002).

I motsetning til grunnprinsippet i iterativ systemutvikling, der man gjennomgår ulike faser i prosessen etter hvert som det er behov for det og der rekkefølgen på fasene ikke alltid er så avgjørende, bør den konseptuelle modellen helst være på plass før designet utformes. Godt grensesnittdesign innledes med en enkel og tydelig konseptuell modell som utgjør ryggraden i designet (Johnson og Henderson, 2002). En godt gjennomtenkt og konsis konseptuell modell burde kunne ligge støtt uten for mange forandringer og være en basis for videre arbeid med systemets brukergrensesnitt. Det kan imidlertid være at designet må gjøres om underveis og at dette må føre til endringer i den konseptuelle modellen. Hvis for eksempel interaksjonsdesignet endres fra en halvgrafisk eller tekstbasert løsning til en løsning basert på bruk av ikoner og direkte manipulasjon kan man stå med en konseptuell bruksmodell som er ulik det den var i utgangspunktet.
Men tidlig utforming av en konseptuell modell for design er likevel hensiktsmessig.
Det kan også være at man ved å fokusere på den konseptuelle modellen avdekker svakheter ved designet som ellers ikke hadde blitt rettet opp før senere. Da spiller det mindre rolle at man faktisk må tilpasse den grunnleggende bruksmodellen.

Jo større datasystemene er, jo flere begrep og metaforer vil brukeren sannsynligvis måtte forholde seg til. Da blir det enda viktigere å utforme en gjennomført og helhetlig konseptuell modell slik at man kan sette begrepene i sammenheng med hverandre og ikke ha en samling løsrevne begreper som hver for seg gir en viss mening, men ikke gir noe bilde av hvordan man skal forholde seg til systemet sett under ett.

4.4.9 Objektorientert design og konseptuelle modeller

I utviklingen av Star begynte designerne tidlig med å bygge en tydelig og forståelig konseptuell modell. I stedet for først å bestemme maskinens funksjonalitet for så å lage brukergrensesnittet, ble psykologer og designere engasjert i prosjektet fra starten av. Man erkjente hvor viktig brukernes konseptuelle modell var for designet av systemet. Alt annet var underordnet det å lage en modell som var tydelig, åpenbar og håndgripelig. Den viktigste rollen til designet var altså å lage en konsekvent ”virtualitet” med en lettforståelig struktur eller modell (Winograd, 1995).
Hjelpemidler for å designe slike virtualiteter har ofte vært basert på objektorienterte modeller, der objektklassene mer har reflektert brukerens perspektiv enn å gjenspeile implementasjons-hensyn (Winograd, 1995). Smalltalk var et objektorientert språk som lot systemutvikleren abstrahere begreper fra den virkelige verden og bruke disse som klasser i datamodelleringen, som nevnt i kapittel 4.3.6.

Det er grunn til å merke seg at i nyere systemutviklingsmetodologi er det mer fokus på objektorientert design enn objektorientert programmering (Winograd, 1995).
Det er likhetstrekk mellom objektorientert design og konseptuelle modeller.
Begge er abstraksjoner fra virkeligheten eller kan ha selvstendig eksistens i datasystemet.
Bruk av Smalltalk til å programmere Star/Altos brukergrensesnitt viste at objektorientert design har et passende abstraksjonsnivå/mekanisme for å støtte de konseptuelle modellene

som ligger i brukergrensesnittet. Selv om objektorientert design (OOD) ikke trenger å være den eneste metoden for konseptuelt design, har OOD den fordelen at det betrakter abstrakte begreper/klasser og skjuler implementasjonsdetaljer på samme måte som at brukergrensesnittet skal skjule det uvesentlige og eksponere til brukerne det de trenger å vite for å utføre arbeidsoppgaven sin. Dette gjør OOD til en effektiv metode for å arbeide med grafiske brukergrensesnitt.

4.5 Designprosessen og prototyping

4.5.1 Design av programvare

I engelsk-engelsk ordbok(Clue for Windows versjon 5.1) defineres design som :

"*Design er prosessen med å planlegge formen til et nytt produkt. Med designet til et framstilt produkt forstås produktets form eller måten det ble laget på*".

Design framstilles altså både som utformingen av et produkt og som prosessen med å lage produktet.

Jeg vil ta for meg design i forbindelse med utvikling av programvare. Her kan designet både forstås som utformingen til programvaren og måten det utvikles på, men når jeg viser til måten produktet framstilles på vil jeg omtale det som designprosessen. På samme måte vil jeg omtale produktets utforming som designet.

Aktivitetene til en programvaredesigner omfatter de tradisjonelle systemutviklings- og programmeringsoppgavene som kravspesifikasjon, ren programmering og debugging, men omfatter i tillegg brukersentrert design i mer eller mindre grad.

Innen programvareutvikling innebærer brukersentrert design fokus på mennesker, deres omgivelser og arbeidsoppgaver, og hvordan teknologi kan bli designet og anvendt slik at det er best mulig tilpasset de som bruker denne teknologien.

ISO sier følgende om brukersentrert design:

"*Human-centred design is an approach to interactive system development that focuses specifically on making systems usable. It is a multi-disciplinary activity which incorporates human factors and ergonomics knowledge and techniques. The application of human factors and ergonomics to interactive systems design enhances effectiveness and efficiency, improves human working conditions, and counteracts possible adverse effects of use on human health, safety and performance. Applying ergonomics to the design of systems involves taking account of human capabilities, skills, limitations and needs. Human-centred systems support users and motivate them to learn. The benefits can include increased productivity, enhanced quality of work, reductions in support and training costs, and improved user satisfaction.*"

(ISO (International Organization for Standardization) 13407, 1999)

Design, usability (brukbarhet) og human factors henger altså sammen. Et godt design gir et produkt som tilfredsstiller brukernes ønsker, øker effektiviteten, øker sikkerheten osv.

Ved brukersentrert design gjennomgår man en hurtig designprosess med flere iterasjoner og omfattende evaluering underveis. Evalueringen danner grunnlag for å drive designprosessen framover ved at man da får nyttig tilbakemelding som danner grunnlag for endringer i designet – redesign. Evalueringsmetoder kan være eksperiment og testing, observasjon og/eller intervju med brukere, spørreskjema osv. Ofte blir flere enn en metode brukt i en designprosess.

Det kan delta folk med ulik bakgrunn i designprosessen, da design krever mange ulike kompetansefelt (ISO 13407, 1999). Det kan for eksempel være at programmerere, grafiske designere, fageksperter og psykologer jobber sammen med designet i et tverrfaglig prosjekt. Brukere er gjerne involvert i designprosessen, enten med rollen som aktiv part i prosjekt-teamet eller blir involvert i testing av testbare prototyper underveis i prosessen. Videre kan en programmerer/systemutvikler delta både i designprosessen, i implementeringen og testen/evalueringen.

Utforming av designet er en kreativ prosess, men krever en metodisk tilnærming, for eksempel innebærer iterativ brukersentrert design en mer eller mindre formell framstilling av designet (Winograd, 1995). Det kan framstilles prototyper som testes ut på brukere for å få nyttig tilbakemelding. Framstillingen bør være presis nok til å fange inn de mest karakteristiske egenskapene til systemet, men bør samtidig være så enkel at den er oversiktig og hindrer forvirring og sammenblanding (Preece, 1994). Det er mange måter å modellere designet på, og ulike designmodeller brukes til ulike formål, men det benyttes oftest mer eller mindre formelle designmetoder for å få en viss retning og struktur på designprosessen. Arbeidet med designet inngår jo som del av systemutviklingsprosessen og krever en metodisk tilnærming.

Det er kan være vanskelig å få formidlet de tekniske og konseptuelle aspektene ved designet til brukere og folk tilknyttet designprosessen. Generelt vil folk ha noe konkret å forholde seg til, og hvis man kan demonstrere det ferdige designet i en konkret prototyp, kan de som får presentert prototypen gi tilbakemelding på hvordan de opplever designet. Slik tilbakemelding gir verdifull informasjon til evalueringen. Utvikling av prototyper kan derfor være et viktig hjelpemiddel for å kommunisere designet slik at det kan evalueres.

4.5.2 En bredere tilnærming til systemutvikling og design

En vesentlig forskjell mellom tradisjonell systemutvikling (software engineering) og tilnærmelsen til programvaredesign er at systemutvikling nesten utelukkende fokuserer på det som skal bygges og hvordan man skal gjøre dette, mens design fokuserer på i hvilken sammenheng produktet/systemet skal brukes. Tradisjonelt har det vært slik at når en dataingeniør sier at noe virker så mener han at det er robust, pålitelig og tilfredsstiller andre funksjonelle spesifikasjoner som måtte settes.

Programvaresystemer består gjerne av tusenvis av prosedyrer og mange avhengigheter innad i systemet. Å designe programvare er derfor en komplisert oppgave og designere trenger så mye støtte i designprosessen som mulig (Preece, 1994). Det er begrenset hvor mye informasjon designere kan forholde seg til uten å bruke noen hjelpemidler, og deltakere i designprosessen kan ha vidt forskjellige designferdigheter. Det som trengs er å tilrettelegge

slik at man støtter designprosessen best mulig, fra ideer og begrepesdanning og til implementering av designet.
Det gjelder å legge til rette for at designere kan konsentrere seg om det de er gode til – å designe programvare. For å få til dette må designverktøyet skjule en del kompleksiteter slik som for eksempel algoritmeeffektivitet, operativsystemspesifikke detaljer og andre datatekniske problemer.

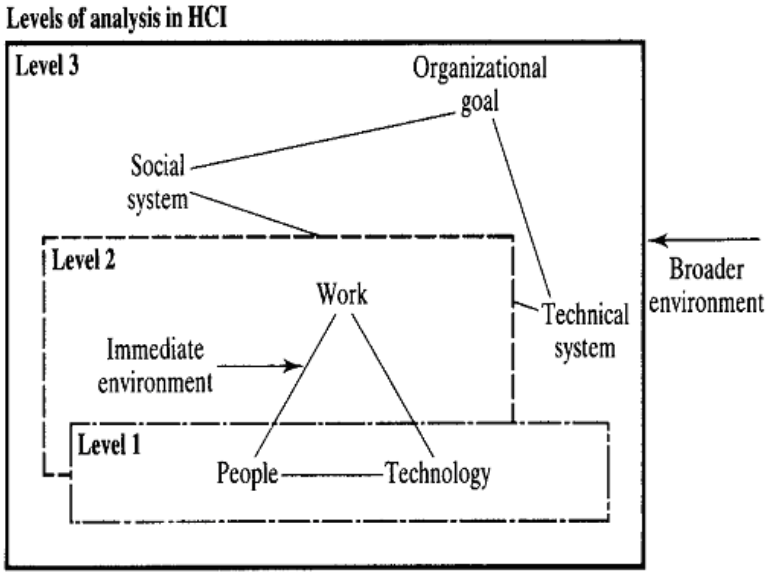
Hensyn til funksjonelle krav er også viktig for en designer, men han vil gjerne i tillegg se på hvordan programvaren virker i en bredere sammenheng. Interessen rettet seg mot de verdier potensielle brukere har og hvilke behov som skal tilfredsstilles. Programvaren brukes gjerne i en viss arbeidsomgivelse som igjen er preget av forskjellige sosiale mekanismer i bedriften og i samfunnet for øvrig, som for eksempel bedriftskultur, sosialt hierarki osv. Slike sosiologiske mekanismer er utenfor oppgavens rekkevidde, men det er viktig å være klar over at de finnes.

Figuren nedenfor viser den bredere konteksten som designere forholder seg til.
I tillegg vil tenkte brukere av prototypen benytte teknologien i en bredere sammenheng.
Både designere og brukere forholder seg til teknologien, jvf. Nivå 1 i figuren.

Designere vil lage prototypen i en viss arbeidssammenheng, der prosjektleder og andre i utviklingsteamet eller tilknyttet prosjektet tilsvarer 'Work' på på nivå 2.
De større organisatoriske rammene, bedriftskulturen, bedriftsprofil osv. tilsvarer nivå 3.

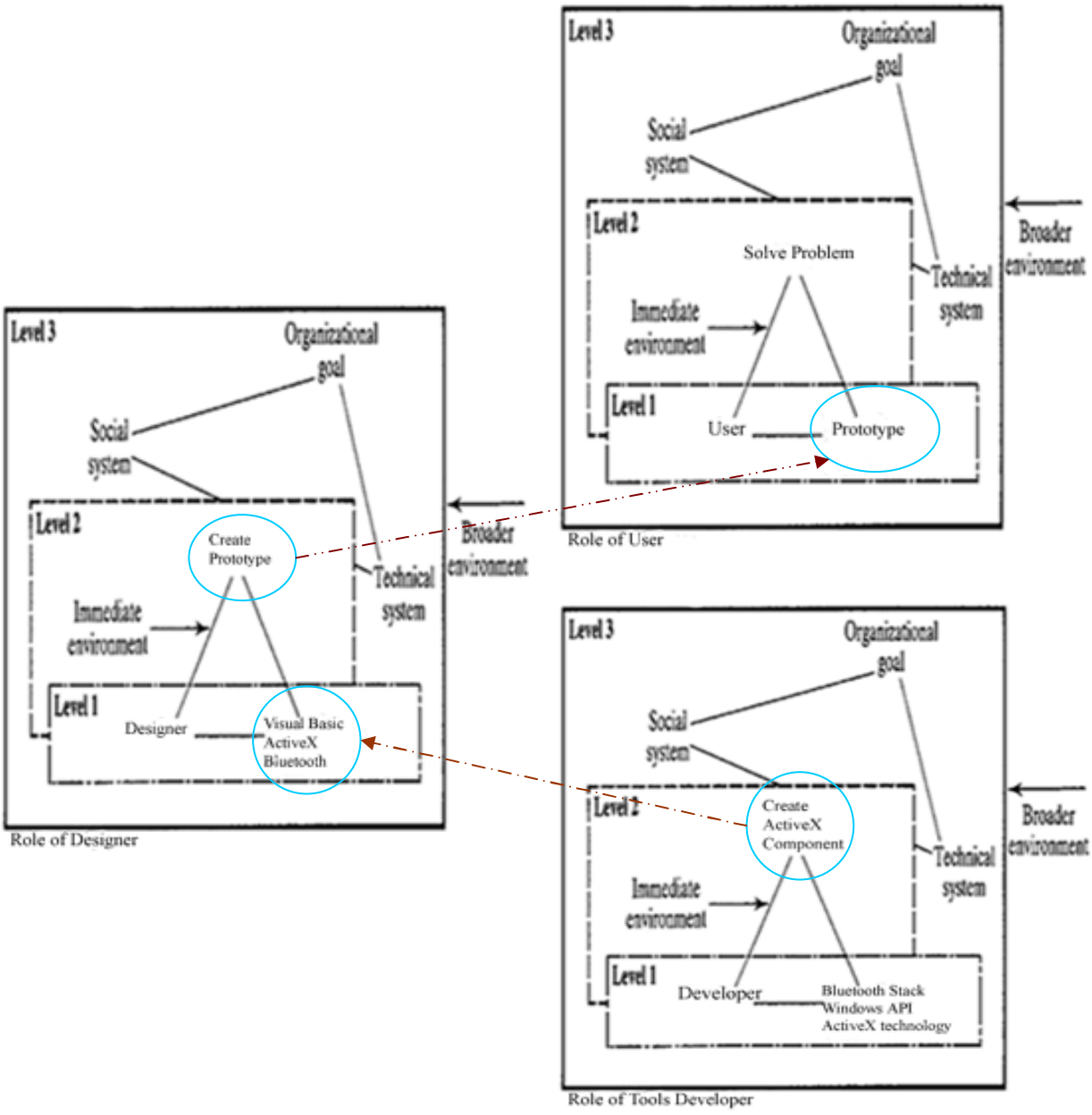
For brukernes del tilsvarer nivå 2 den spesifikke arbeids-eller brukssammenhengen som prototypen er beregnet for. Nivå 3 tilsvarer den bredere konteksten teknologien benyttes i, for eksempel til å fremme målsetning om bruk av datautstyr til bedre kommunikasjon.

Designeren må imidlertid planlegge designet ut fra både sine egne omgivelser og brukere av prototypen sine omgivelser.



Figur 4.11 MMI i kontekst (Preece, 1994)

Sammenhengen mellom utvikling av designverktøy, design av prototypen og bruk av prototypen er vist i figur 4.12. Figuren er "instansiert" for verktøyutvikler, designer og bruker. Utvikleren lager et designverktøy – i vårt tilfellet en ActiveX komponent Designer benytter i sin tur verktøyet i designet av prototypen, som så presenteres for brukeren.



Figur 4.12 Sammenhengen designverktøy, design og bruk av design

Designprosessen kan være forskjellig alt etter hvilken sammenheng det designes i og hva som designes. Det kan være ulike typer programvare med svært forskjellige egenskaper, det deltar gjerne forskjellige personer med ulik bakgrunn og ulike erfaringer i et prosjekt som designprosessen inngår i, og det benyttes gjerne vidt forskjellige ressurser og verktøy i ulike designprosesser. Designprosessen vil også være avhengig av hvilken målgruppe produktet er

beregnet på. Ulike brukergrupper vil legge vekt på ulike aspekter ved produktet og vil ha hver sine krav. Dessuten vil det ha betydning hvilken sammenheng produktet skal brukes i.

Prosessen med å designe ett programvareprodukt kan altså kreve helt andre ressurser, fagpersoner eller designmetoder enn designet av et annet produkt og designerne må gjerne forholde seg både til sine egne omgivelser og brukernes omgivelser.

4.5.3 Hvorfor lage prototyper?

Det er viktig for både bruker og designer å kunne se for seg hvordan et program eller system vil bli, hvilke designvalg som skal tas og hva brukere kan gjøres med programmet. Å demonstrere designet for brukere, design-teamet og andre involverte tidlig i designprosessen kan være klargjørende og gjøre at det endelige produktet blir slik som forventet. Tanken er at designere av et system kan få anledning til å prøve ut sine ulike ideer - og å få kontinuerlig tilbakemelding fra brukere. Til dette kan utvikling av prototyper være et viktig hjelpemiddel.

Det kan være hensiktsmessig å ha gjennomgått de konseptuelle sidene ved designet før selve designprosessen tar til. Dette kan gi større rom for kreativitet, man kan få belyst flere alternative designvalg og blir ikke heftet av funksjonelle og tekniske beskrankinger.

Ved å benytte prototyping ønsker man å oppmuntre til en mer effektiv systemutvikling. Ved å bryte komplekse og uklare problemer ned i mindre og lettere begripelige deler, der prototypen demonstrerer de enkelte aspektene ved designet, kan man fjerne en del av usikkerheten knyttet til designet. En slik tilnærming kan være til hjelp for å tilvirke, og senere raffinere, et produkt som møter de forventninger brukere eller markedet måtte ha til det (Carr og Verner, 1995). Designere og systemutviklere kan samle nyttig informasjon om produktet til å løse uklarheter rundt hvorvidt designet er i samsvar med de krav brukerne og samarbeidspartnere stiller.
Prototyping benyttes til å gi informasjon om nødvendig funksjonalitet, hvordan systemet og dets bestandsdeler skal representeres, antyde hvilken brukerstøtte og hjelp som brukere krever og gi en pekepinn på hvordan brukergrensesnittet skal fremstå (Preece, 1994).

4.5.4 Prototyper kommuniserer designet

Det er viktig å ha klart for seg hva som er formålet med prototypen. Hva som skal prototypes og hvem prototypen er myntet på er sentrale spørsmål som bør besvares. En prototyp har gjerne et bredt publikum, noe som kan gjøre det vanskelig å få svar på alle interessante designproblemene.
Overfor brukere kommuniseres prototypen for å få tilbakemelding på designet. Innad i designteamet evaluerer designere ulike designvalg ved å demonstrere ulike prototyper for alternative design. Overfor investorer, prosjektledere og andre parter tilknyttet de involverte organisasjonene kan prototypen for eksempel kommunisere framdrift og retning i utviklingsprosjektet (Houde og Hill, 1997).

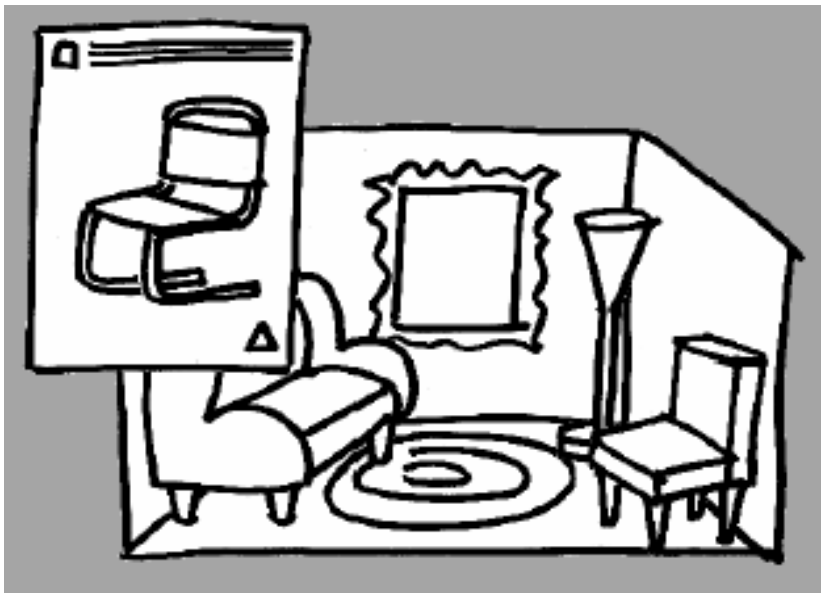
Prototypen skal gi brukerne en oppfatning av hvordan designet er og ulike designvalg som kan være aktuelle. Brukertester av prototyper gir i sin tur viktig informasjon om designet imøtekommer de kravene som stilles til systemet.
Spesielt i brukersentrert design er interaksjonen med brukere i designprosessen viktig og ved hjelp av prototyper kan man formidle egenskaper ved designet til brukere.
Prototyping benyttes i forbindelse med en iterativ designprosess. Prototyper kan utvikles underveis i prosessen og gi nyttig informasjon om produktet som utvikles. Framstilling av en prototyp og testingen av den på brukere utgjør gjerne en iterasjon i utviklingsprosessen.
Det må i de fleste tilfeller gjøres større eller mindre endringer av designet underveis, gjerne ut fra tilbakemelding fra brukere. Gjennom gjentatte redesign forbedres designet slik at systemet møter de forventninger og krav som designere, utviklere og brukere stiller til det.
I denne sammenheng har prototypen en kommunikativ rolle overfor de involverte partene i systemutviklingsprosessen. Når prototypen er blitt laget er utfordringen å formidle den til de ulike "tilhørerne".

I tillegg til at prototypen gir viktig informasjon om designet til brukere, kommuniserer den designet til de ulike medlemmene i design-teamet og til de involverte organisasjonene forøvrig. Prototyper gir dessuten ikke bare nyttig tilbakemelding på designet, men er også et viktig medium for informasjon, interaksjon, integrasjon og samarbeid (Winograd, 1995).

Design av programvare innebærer samarbeid mellom ulike fagfolk. For eksempel kan et prosjekt kreve kunnskapene til en programmerer, en interaksjonsdesigner, en industriell designer og en prosjektleder. De ulike aktørene i prosjektet kan ha forskjellig oppfatning av prototypen. Å velge riktig prototypingsteknikk til et såpass bredt publikum kan være krevende (Houde og Hill, 1997). I tillegg til et sammensatt design-team og andre personer tilknyttet prosjektet, er gjerne brukergruppen som systemet er ment for relativt uensartet.

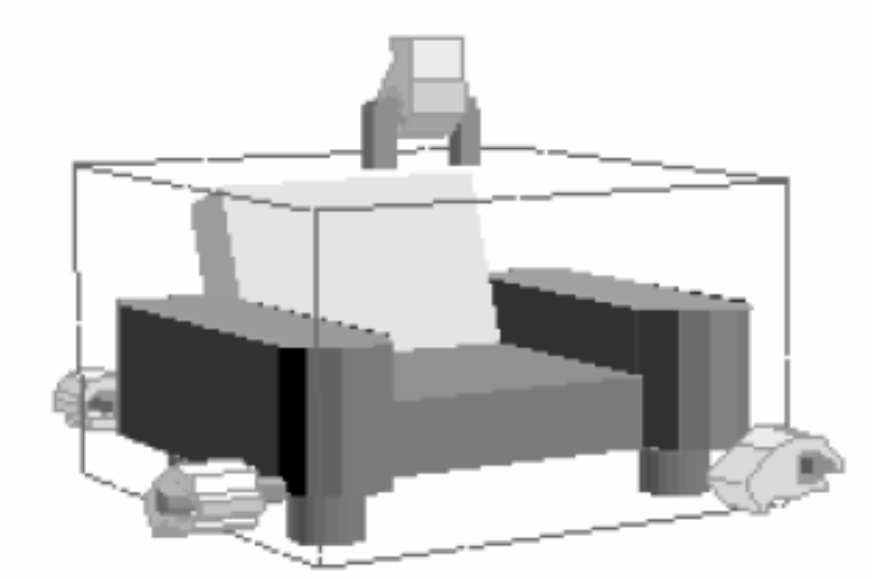
En prototyp vil derfor gjerne fokusere på deler av et system og overser andre deler og vil være forskjellig fra det ferdige produktet med hensyn til størrelse, pålitelighet og funksjonalitet (Preece, 1994). Hvilken prototypingsteknikk som velges vil avhenge av hva som skal prototypes, altså med formålet med prototypingen.

Det kan være at det velges en *rolle-prototyp* hvis man vil undersøke hva en bestemt teknologi betyr for brukere og hvilken rolle den spiller for brukere i en gitt sammenheng, uten at man får noe særlig inntrykk av utseende til brukergrensesnittet eller funksjonaliteten. Det er kun hva brukergrensesnittet og funksjonaliteten betyr for brukeren som blir undersøkt. Sketcher og papirprototyper kan da gi verdifull informasjon om hvordan designet skal utformes for å tilby de konseptuelle ønskene til brukere, designer og oppdragsgiver. Rolleprototypen kan lages svært hurtig og likevel gi nyttig kunnskap om konseptuelle aspekter ved designet.



Figur 4.13 Rolleprototyp for en 3D interiørplanlegger (Houde og Hill, 1997)

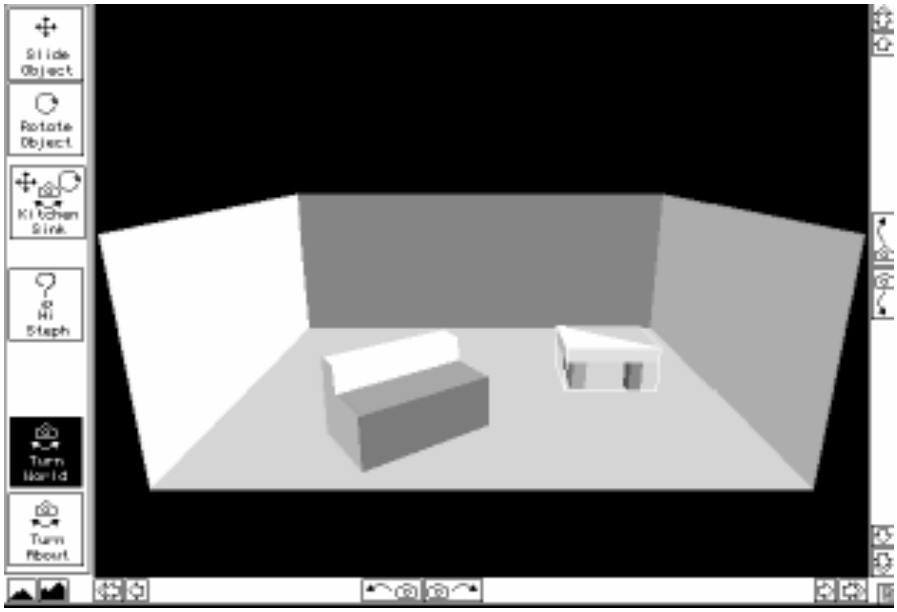
Hvis man derimot ønsker å undersøke ulike framstillinger av brukergrensesnittet kan en *look-and-feel-prototyp* gi et mer realistisk bilde av dette. En slik prototyp simulerer interaksjonen i brukergrensesnittet, slik som ulike valgmuligheter og rekkefølge på operasjoner i grensesnittet. Prototypen gir dessuten svar på hvordan designet oppleves av bruker, hvor behagelig og brukervennlig det føles. Look-and-feel-prototypen belyser ulike framstillinger av systemet, altså den visuelle framtoningen og hvordan det føles.
Men look-and-feel gir ikke noen svar på hvilken betydning de ulike elementene i systemet har, slik rolle-prototyper gjør.



Figur 4.14 Look-and-feel-prototyp for en 3D interiørplanlegger (Houde og Hill, 1997)

Noen prototyper bygges primært for å besvare tekniske spørsmål rundt hvordan noe faktisk virker. Slike prototyper kalles *implementasjonsprototyper* og kan lages for å gi et inntrykk av funksjonaliteten til et system. En implementasjonsprototyp tar gjerne for seg et begrenset antall tekniske aspekter ved systemet og demonstrere disse relativt grundig. Tekniske aspekter ved bestemte deler av funksjonaliteten til et system blir dermed grundig undersøkt i en slik prototyp. Dette er dermed en vertikal prototypingsteknikk, i motsetning til rolle-og look-and-feel-prototyper, som primært innebærer horisontal prototyping. Rolleprototyper illustrerer hvordan systemet kan brukes og hvilken betydning ulike begrep har, og tar gjerne også for seg bredere aspekter ved systemet.
Look-and-feel-prototyper gir en horisontal tilnærming til prototypingen, da det viser det generelle brukergrensesnittet, men ikke går i dybden med hensyn til hvordan systemet er bygd opp.

Implementasjonsprototyper kan brukes innad i et designteam som et middel for eksperimentering, for å demonstrere muligheter og begrensninger ved teknologien til andre samarbeidspartnere(ledere, investorer etc.) og for å få tilbakemelding fra brukere på hvor godt teknologien virker (Houde og Hill, 1997). Slike egenskaper ved teknologien kan ellers være vanskelig å simulere ved hjelp av bare rolle-prototyper eller look-and-feel- prototyper.



Figur 4.15 Implementasjonsprototyp for en 3D interiørplanlegger (Houde og Hill, 1997)

Det er viktig å være tydelig på hvilke designspørsmål som skal løses når man skal velge hvilken eller hvilke typer prototyper som skal bygges. Det kan være vel så hensiktsmessig å fokusere på formålet med prototypingen som på hvilke prototypingsverktøy som benyttes.

En prototypingteknikk kan passe i en sammenheng, mens en annen er mer hensiktsmessig i en annen sammenheng. Det er dessuten sannsynlig at flere typer prototypingsteknikker brukes i samme designprosjekt. En prototyp kan ha et bredt sammensatt publikum og skal formidle mange ulike aspekter ved et system. Det kan derfor være naturlig med flere ulike prototypingteknikker som gir komplimentær informasjon til bruk i evalueringen av designet.

5. Utviklingsomgivelser og prototypingsverktøy

5.1 Hvorfor er det viktig å støtte designprosessen?

Nær sagt alle utviklingsomgivelser har verktøy for å lage brukergrensesnittet og disse er som regel basert på et grafisk grensesnitt.
Verktøy for å støtte utvikling av brukergrensesnittet er viktig fordi det reduserer den mengden kode utviklere/designere trenger å skrive for å lage brukergrensesnittet og gjør dem i stand til å få opp brukergrensesnittet raskere. Dette i sin tur muliggjør hurtigere prototyping og derfor flere iterasjoner av designprosessen, som er en avgjørende faktor for å lage brukergrensesnitt med høy kvalitet (Myers, 2000). Man kan sammenligne dette med utforming av keramiske produkter. Designeren har mulighet til å forme leiren til den har fått den formen som er ønskelig. Denne reflekterende omgangen med "byggematerialet" er nøkkelen til en effektiv designprosess. Interaktivitet er viktig i dagens interaksjonsintensive utviklingsomgivelser som tilrettelegger for fortløpende uttesting. Både grensesnittet og den underliggende funksjonaliteten til programvare blir utviklet inkrementelt gjennom samhandling med fremtidige brukere i en iterativ og brukersentrert designprosess (Winograd, 1995).

Videre kommer det at en designers vurderinger i tidlig designfase er opportunistisk og foranderlig. Det innebærer at i stedet for å følge en fastlagt rekkefølge fra abstrakte begrep og til mer og mer konkrete løsninger, hopper designeren gjerne mellom ulike abstraksjonsnivå etter hvert som han får mer innsikt i problemstillingen (Preece, 1994). Dette gir store utfordringer når man skal lage verktøy for design og prototyping. I testing av prototyper eller annen evaluering kan nye opplysninger dukke opp på bakgrunn av tilbakemeldinger fra brukere. Dette krever sannsynligvis nye endringer som må testes ut og evalueres.

Å kommunisere og teste ut ideer er også en sentral del av design. Ved hjelp av prototyper i forskjellige deler av designprosessen kan man få testet ut på brukere om designet virker bra, men også lufte ut tanker og ideer rundt designet i prosjektet/mellom aktører i prosjekt.
I tillegg til en høy grad av interaktivitet er det derfor et mål at utviklingsomgivelsen legger til rette for utforskning av ulike design og muligheter ved et produkt. Utviklingsomgivelsen skal være et hjelpemiddel for å fremme kreative ideer og for å formidle/ kommunisere disse ideene slik at designerne selv, brukere eller andre kan ha noe konkret å evaluere. Dette er også grunnen til at prototyping er så viktig. Prototyper gir ikke bare tilbakemelding til utviklerne, men er også et hjelpemiddel for kommunikasjon og samarbeid (Winograd, 1995).

Utover det å få programvaren til å fungere, må moderne interaktive utviklingsomgivelser gi støtte til hele designprosessen og det som det innebærer. Sammenhengen som programvaren designes i ser ut til å bli viktigere, slik at utviklingsomgivelsen i enda større grad må integreres i omgivelsene og gi støtte utover det programmeringsmessige.
Man må altså ta høyde for at utviklingsomgivelsen vil brukes i ulike sammenhenger og til ulike formål.

5.2 Etablering av et allment brukergrensesnittspråk

Utviklingsomgivelser har gått i den retning at de kan brukes til flere ting enn å utvikle fullstendig programvare. De kan også benyttes til å lage prototyper. Et formål med prototyper er å demonstrere til "publikum" hvordan designet skal se ut og fungere, slik at man får evaluert designet. Utviklingsomgivelsene idag tillater at man kan lage prototyper med mer eller mindre funksjonalitet. Utviklingsomgivelsene har forskjellig grad av muligheter når det gjelder grensesnittbygging og funksjonalitet, og det må vurderes hvilke muligheter den enkelte har når man skal velge utviklingsomgivelse.
En tradisjonell utviklingsomgivelse fokuserer på å få prototypen til å gjøre de rette tingene, men sett ut fra et designperspektiv så fokuseres det på å at prototypen skal formidle designet. Hovedpoenget med en prototype er den kommunikative rollen den spiller (Winograd, 1995).

I systemutviklingsperspektiv så har programmene blitt større og mer omfattende. Dette har medført at ulike konseptuelle modeller og verktøy er blitt introdusert for å lette prosessen med å utvikle applikasjoner. Trenden har gått i den retning at kompleksiteter har blitt abstrahert vekk og programmeren har fått nye konsepter på høyere abstraksjonsnivå å forholde seg til, fra maskinkode til "naturlige språk". Ved hjelp av parsere og kompilatorere ble det "naturlige språket" gjort om til maskinkode.
For å hjelpe programmerene ble ulike verktøy som støttet opp under programvareutvikling samlet i en utviklingsomgivelse.

Nær sagt alle applikasjoner i dag består av window-managers, verktøysamlinger og grensesnittbyggere som har røtter tilbake til forskning og utvikling innen menneske-maskin interaksjon og brukergrensesnitt på 1970 og 1980-tallet.
Mens 1970 og 1980-tallet var preget av eksperimentering med ulike input - og interaksjons-modeller, har dette stabilisert seg rundt PC'er med mus og tastatur som inpututstyr og et grafisk brukergrensesnitt på en stor fargeskjerm (Myers, 2000). Skrivebordsmetaforen, som nevnt tidligere, ble introdusert av Xerox PARC og har spredd seg til Macintosh, Windows og andre plattformer, og de fleste applikasjoner på de tilhørende plattformene.

Denne stabiliteten har altså medført en konsistent og uniform interaksjonsform basert på velkjente metaforer. Denne uniformiteten har gjort det mulig for brukere å benytte opp kunnskaper og ferdigheter på en viss plattform eller applikasjon og så gjenbruke dette i andre plattformer eller applikasjoner.

Moderne utviklingsomgivelser har på samme måte innarbeidet en konsistent og uniform interaksjonsform. Utviklingsomgivelsen tilbyr et visuelt grensesnitt mot brukeren som består av kjente begreper og metaforer for å gjøre det lettere for utviklere å få utført sine oppgaver. Et tydelig kjennetegn hos moderne utviklingsomgivelser er således den visuelle utformingen.

Utviklingsomgivelser som Visual Basic, Director og Supercard har grensesnittbyggere for å lage skjermbilder med knapper, vinduer og andre objekter og gjør utviklere i stand til å knytte funksjonalitet til disse objektene ved hjelp av et programmeringsspråk. I tillegg til programmeringsspråkets metodebiblioteker, legges det til rette for å kunne benytte ferdigkompilerte komponenter som utvider mulighetene til å lage funksjonalitet.

5.3 Grensesnittbyggere

En grensesnittbygger (interface builder) er et interaktivt verktøy som gir mulighet til å bygge opp et grensesnitt hurtig ved hjelp av grafiske komponenter.

Grafiske verktøy som grensesnittbyggere gjør det mulig for utvikleren å lage brukegrensesnitt uten å bruke for mye tid og krefter på å få opp brukergrensesnittet. Dette har ofte vært en ekstra terskel i utviklingen av programvare og har gjerne innebæret at utvikleren har måttet kunne grafikkprogrammering eller har måttet benytte visse grafikkbiblioteker. Slik har utvikleren måttet sette seg inn i en del teknisk kompliserte ting før han har kunnet ta fatt på den egentlige oppgaven – å utforme selve programmet. Å utvikle grensesnittobjektene fra scratch på "windows manager"-nivå er ofte både tidkrevende og langtekkelig og gjør det praktisk talt umulig å få til et konsistent brukergrensesnitt. For å fri utviklere fra slike kompleksiteter - og å dermed lette utviklingsprosessen, er det hensiktsmessig å benytte grensesnittbyggere. Disse bygger en abstraksjon over "windows manager"-nivået og tilbyr en samling interaktive komponenter og et rammeverk for å håndtere komponentenes grensesnitt (Myers, 2000).

Grensesnittbyggere er ment å skulle gjøre utviklingsomgivelser mer responsive og interaktive. De må kunne reagere fortløpende på endringer som utvikleren gjør i koden, i grensesnittet eller i andre deler av utviklingsomgivelsen. Til dette er WYSIWYG-prinsippet (What-you-see-is-what-you-get) verdifullt. Et grensesnittvindu vil da kunne gi utvikleren en indikasjon på hvordan programmet blir seende ut når det blir kjørt.
De fleste utviklingsomgivelser har en slik type grensesnittbygger, slik at utvikler kan teste ut hvordan endringer han har gjort i koden eller direkte i grensesnittobjektene blir seende ut i look-and-feel.

En viktig årsak til at grensesnittbyggere har hatt slik suksess er at de uttrykker grafiske begreper ved hjelp av grafiske metoder. Ved å la grensesnittbyggeren ta seg av grafiske aspekter ved implementeringen av brukergrensesnittet er utforming av brukergrensesnitt gjort tilgjengelig for de som har en viss programmeringskunnskap, men som ikke er rene programmerere. Dette har gjort det mulig for fageksperter å tilpasse grensesnittet til deres oppgaver og for grafiske designere å bli mer involvert i utformingen av grensesnitt.
Programmerere har også tjent på dette ved at de har brukt mye mindre tid og krefter på å lage grensesnittet og har kunnet konsentrere seg om sine egentlige oppgaver.
Grensesnittbyggere har resultert i en lavere terskel for å komme i gang med å benytte en utviklingsomgivelse, og en flatere læringskurve (Myers, 2000).

Man har kunnet skille brukergrensesnittet fra funksjonaliteten under og dermed fått en bedre struktur på utviklingsoppgavene og en bedre arbeidsdeling mellom deltakere i utviklingsprosessen. Grafisk designere har for eksempel kunnet konsentrere seg bedre om det utseendemessige ved designet og programmerere har fått frigjort konsentrasjon til å lage funksjonalitet, samtidig som det har blitt lettere å kommunisere og samarbeide om designet.

Nyvinningene innen interaksjonsteknikker ga grunnlaget for grensesnittbyggere slik man kjenner dem fra prototypingsomgivelser som Hypercard, Visual Basic og Flash.
Brukergrensesnittet kan utformes ved å velge og flytte grafiske objekter ved hjelp av en mus hovedsakelig fra en verktøylinje og direkte over til det grafiske grensesnittvinduet.
Dette er en interaksjonsform som har blitt både populær og innarbeidet. Faktisk bygger de aller fleste Windows- og Mac- applikasjoner på denne grensesnittmetaforen. Brukeren peker

og klikker på verktøyene og flytter enten objekter fra verktøylinja over til grensesnittvinduet eller finner verktøy på verktøylinja som brukes til å manipulere objekter på formen. I tegneprogram har dette lenge vært en utbredt skjermmetafor.
Utviklingsomgivelser som Flash, Visual Basic, Supercard osv har for lengst adoptert denne måten å la brukeren arbeide på. Brukeren kan ta med seg kunnskapene fra grensesnittbyggeren i en utviklingsomgivelse og over til en ny uten for mange problemer, da de fleste utviklingsomgivelsene bygger på lignende metaforer og har standardiserte GUI-objekter – et felles designspråk.
Den tidligste utviklingsomgivelsen som hadde en grensesnittbygger var Apples Hypercard som la grunnlaget for visuell komponentbasert programmering og rapid application development (RAD) (Durham, 2001).

5.4 Programmeringsmuligheter

5.4.1 Generelle programmeringsmuligheter og brukskrav

Grensesnittbyggere er nyttige for å bygge opp brukergrensesnittet, men utviklingsomgivelsen må også legge til rette for programmering av funksjonalitet til det grensesnittet som presenteres. Til dette tilbyr de fleste utviklingsomgivelser mer eller mindre avanserte programmeringsmuligheter.
Det er en del fellestrekk ved moderne utviklingsomgivelser når det gjelder programmerbarhet, og det stilles krav til at utviklingsomgivelsene kan brukes av personer med forskjellig bakgrunn og kompetanse i ulike sammenhenger. Det er visse programmeringsmuligheter en moderne utviklingsomgivelse vanligvis bør ha.

Ofte blir programmering av funksjonaliteten bak grensesnittobjekter omtalt som scripting. Jeg vil for enkelhets skyld bruke begrepet programmering om både scripting og programmering generelt, siden det for utvikleren oppleves som to sider av samme sak.

Utviklingsomgivelser har som oftest et eget vindu med en teksteditor som brukes til å programmere kodelinjer. Her kan det programmeres funksjonalitet knyttet til handlinger som utføres på grensesnittobjektene som er bygd i grensesnittbyggeren eller det kan være annen kode som ikke er direkte knyttet til grensesnittobjektene.

Et hvert programmeringsspråk med en viss utbredelse innebærer generelle kontrollstrukturer som variabler, løkker og betingelser. Selv om en del programmeringsspråk sies å ha lav terskel og ikke utelukkende er beregnet for eksperter, blir disse språkene oftest brukt til mer avanserte ting enn det opprinnelig var meningen, eller det dukker opp flere krav til funksjonalitet etter hvert som et program blir utviklet. Fullstendige programmerings-muligheter er derfor nødvendig (Myers, 2000).

Utviklingsomgivelser har ofte et ganske generelt anvendelsesområde og benyttes av utviklere med forskjellig programmeringserfaring, ferdigheter og bakgrunn. Derfor blir programmeringspråk ofte en avveining mellom flere hensyn. Syntaksen bør være enkel og konsistent slik at det er enkelt å få til ting i programmeringsspråket og lett å lære. Hvis utviklingsomgivelsen skal være velegnet for prototyping og iterativt design må programmeringsspråket ha et abstraksjonsnivå som er tilpasset de programmerings- og designoppgavene som skal løses.

Samtidig kan hensynet til enkelhet være til hinder for hvor mye det er mulig å få til ved hjelp av språket. For noen utviklere kan programmeringsspråket aldri få nok muligheter, mens andre har behov for enkelhet og oversikt. Dette ser ut til å være et dilemma som det er vanskelig å finne noen enkel løsning på.

Siden programmeringsmulighetene er integrert med grensesnittbyggeren – for eksempel funksjonalitet knyttet til GUI-objektene, bør språket gjøre det lett å håndtere de grafiske objektene. Mye av programmeringen vil jo kunne være tilknyttet interaksjonsdesignet, slik som widgets for input og output av data, feilmeldinger, menyer osv.

Utviklingsomgivelsen bør tilby gode debuggingsmuligheter. Feilmeldinger må være fornuftige og gi mening for utvikler. Når programmene som lages blir større, øker utfordringene selv om modularisering og forbedrede utviklingsmetoder gjør det lettere å holde oversikten.
Samarbeid mellom flere applikasjoner kompliserer dette ytterligere. Hvis en utviklingsomgivelse importerer metoder fra et annet programmeringsspråk, kan det være vanskelig å finne feil, da de importerte metodene ikke lar seg debugge på samme måte som utviklingsomgivelsens kode.

Programmering mot andre typer datamaskiner enn vanlige stasjonære PC'er - som PDA'er og mobiltelefoner, gjør i tillegg at man pånytt må ta stilling til maskinvarerelaterte problem-stillinger som begrenset minnekapasistet, nye input/output-verktøy osv.

Etter hvert som systemutviklingsfeltet har utviklet seg, har fokuset skiftet fra å være rettet mot algoritmer og datastrukturer og til å sette søkelys på de kognitive strukturen til menneskene som lager programmene. En god utviklingsomgivelse støtter de prinsipper og metoder som gjør utviklere mer produktive (Winograd, 1995). I tillegg til at programmeringsspråkene skal være forståelige, har utviklingen gått i retning av å støtte systemutviklingsprosessen i sin helhet. Design og programmering må ses i sammenheng og utviklingsomgivelsens rolle blir å integrere disse aktivitetene.

Moderne utviklingsomgivelsen har større krav til interaktivitet og må kunne respondere fornuftig på det utvikleren gjør i omgivelsen.
Utviklingsomgivelsen skal legge tekniske til rette for at utvikleren skal få utført sine oppgaver på best mulig måte. Utvikleren skal raskt kunne prøve ut noe, se hvordan resultatet ble, gjøre eventuelle endringer og så se hvordan resultatet ble. For at utvikleren skal kunne prøve ut ideer og få fortløpende tilbakemelding, stilles det store krav til utviklingsomgivelsen. Utvikler må fortløpende kunne se resultatet av endringer i programmet.

De første utviklingsverktøyene ble utviklet ved hjelp av interpreterte språk. Xerx PARC's Smalltalk ga muligheten til å hurtig å prototype ulike ideer til brukergrensesnitt og enkelt gjøre endringer underveis. Smalltalk hadde dessuten et lite verktøysett.
Med framveksten av kompilerte språk som C og C++ forsvant mye av interaktiviteten og den hurtige responsen. (Myers, 2000).

5.4.2 Prosedyrale, objektorienterte og hendelsesorienterte språk

Programmeringsspråk som brukes i utviklingsomgivelser er gjerne klassiske prosedyrale, objektorienterte eller hendelsesorienterte språk. Felles for språkene er at de gir utvikler mulighet til å definere funksjonalitet i –og bak brukergrensesnittet.

- I prosedyrale språk fastslår utvikleren ulike delproblemer og samler disse i prosedyrer.
- I rene objektorienterte språk, som i Jbuilder, implementeres grensesnitt og funksjonalitet som klasser.
- Hendelsesspråk (event languages) tar utgangspunkt i ulike events (hendelser) knyttet til handlinger gjort av brukere på grensesnittobjekter eller andre hendelser som ikke er knyttet til selve grensesnittet, men som inntrer som følge av at et objekt har endret tilstand - at et tidsur slår inn eller lignende.

De fleste utviklingsomgivelser bygger på prinsippet med grensesnittbyggere og benytter mekanismen hvor hendelser som for eksempel peking og klikking i et vindu "vekker" event-håndterere som så utfører en eller annen bestemt hendelse. Slike event-mekanismer benyttes i de aller fleste moderne utviklingsomgivelser, men ikke alle har vesentlige innslag av eventer i selve programmeringsspråket.
Hendelses – og objektorienterte språk har en del felles kjenntegn, som bruk av objekter og klasser, innkapsling, modularisering osv. En utviklingsomgivelse kan ha både objektorienterte og hendelsesorienterte trekk. Disse språkene har oftest prosedyrale kjennetegn blant annnet ved at de fortsatt baserer seg på at delproblemer blir lagt i metoder/prosedyrer, selv om tankegangen er objekt – eller event orientert. De finnes i dag få språk som utelukkende er prosedyrale og ikke har tatt til seg noen trekk fra objekt – eller hendelsesbaserte språk. Språkene kan således være vanskelig å sette i absolutte kategorier, men er "hybrider" mellom prosedyral, objektorientert og hendelsesorientert tankegang.

Hendelsesorienterte programmereringsspråk fokuserer imidlertid mer på hendelsesaspektet enn andre typer språk. Hendelsesaspektet er grunnleggende for måten å programmere på og er gjerne innbakt i den konseptuelle modellen.
Hendelsesorienterte språk er nært knyttet til grafiske interaksjonsteknikker og har blitt populære fordi de passer godt sammen med grafiske brukergrensesnitt og prinsippet om direkte manipulasjon. Det virker intuitivt for utviklere at visse typer handlinger utløser bestemte hendelser. Hendelsesorienterte språk kan sies å støtte opp om en modalløs interaksjonsform, sammen med prinsippet om direkte manipulasjon. Brukeren slipper altså å gjøre handlinger i en fastlagt rekkefølge.

Flash og Visual Basic er eksempler på omgivelser som benytter eventer i den konseptuelle modellen.
Programmeringsspråket til disse utviklingsomgivelsene har forøvrig både prosedyrale og objektorienterte egenskaper, i tillegg til de hendelsesorienterte. Poenget er ikke at språket nødvendigvis skal følge en fastlagt tankegang/orientering, men at språket gjør det lett for ulike utviklere/ designere å uttrykke løsninger på forskjellige problemer og ellers føyer seg inn i utviklingsomgivelsens øvrige konseptuelle modell.

5.5 Bruk av komponenter

Flere av dagens utviklingsomgivelser legger til rette for ”arbeidsdeling” mellom applikasjoner. Til dette kreves det en standardisering slik at applikasjonene kan ”snakke sammen” uten for store hindre. En måte en slik arbeidsdeling kan skje på er ved utvidelse av utviklingsomgivelsen vha komponenter eller importerte biblioteker. En utviklingsomgivelse kan for eksempel benytte grensesnittet til en komponent skrevet og kompilert i annet programmeringsspråk.

Ideen om å lage applikasjoner ved å kombinere separate skrevne og kompilerte komponenter dynamisk ble først demonstrert i datasystemet ”Andrew” fra Carnegie Mellon University IT Center (Carnegie Mellon, 2004). Ulike komponenter kontrollerte hvert sitt rektangel på skjermen og i disse rektanglene kunne det i sin tur være andre komponenter. For eksempel kunne en tegning inne i et tekstdokument bli kontrollert av en tegnekomponent som var uavhengig av tekst-editor komponenten. Denne ideen ble senere tatt i bruk av Microsoft’s OLE (Object Linking and Embedding) og ActiveX, Sun’s JavaBeans og Apple’s OpenDoc og var et gjennombrudd for å gjøre det enklere og mer effektivt å utvikle applikasjoner og integrere grafiske elementer i ulike programmer.

En av grunnene til at komponentmodellen ble en suksess er at den rettet seg mot et viktig aspekt ved design av programvare: hvordan man kunne modularisere programvare i håndterbare deler, samtidig som man tilbød gode muligheter og fleksibilitet for de som skulle benytte programvaren (Myers, 2002).

Komponenter som inkluderes i utviklingsomgivelser for å utvide funksjonaliteten kalles plug-ins eller extensions. Extension innebærer bruk av ferdige komponenter som tilbys som prekompilerte binærfiler. Dette gir en arbeidsdeling der komponentene kan være ferdigutviklet av et tredjeparts leverandør, og så tilbudt for bruk i en bestemt utviklingsomgivelse som grafiske eller funksjonelle verktøy.

5.6 Utviklingsomgivelser som kan benyttes til prototyping

5.6.1 Generelt om utviklingsomgivelsene

Jeg vil nå ta for meg ulike utviklingsomgivelser som kan benyttes til prototyping. Jeg har benyttet betegnelsen utviklingsomgivelse i stedet for prototypingsverktøy fordi utviklingsomgivelse er en mer generell betegnelse på et verktøy som kan benyttes for å lage programvare med funksjonalitet. Prototypingsverktøy er i mange tilfeller verktøy som kun fokuserer på det visuelle aspektet ved en applikasjon eller visse domenespesifikke områder.

Min innfallsvinkel er å se på muligheter for å kunne gjøre ny teknologi tilgjengelig, og da må utviklingsomgivelsen gi støtte for uttesting av funksjonalitet i tillegg til visuelle aspekter.

Jeg vil se på hvilke muligheter hver enkelt utviklingsomgivelse har for utvidelse av funksjonaliteten, slik at det kan lages et grensesnitt mot Bluetooth.
Jeg tar for meg Director, Flash, Supercard, Jbuilder og Visual Basic, og ser hvorvidt de ulike utviklingsomgivelsene kan brukes til prototyping av ny teknologi (se kap. 2 om problemstilling for definisjon av hva som ligger i ny teknologi og hvem målgruppen er).

Det som er felles er at det i utgangspunktet er relativt enkelt å bruke dem til å lage grafisk brukergrensesnitt og at de tilbyr utvidet funksjonalitet slik at ulike teknologier kan gjøres tilgjengelige i utviklingsomgivelsene. Disse utviklingsomgivelsene er egnet til prototyping og støtter opp om designprosessen. Selv om Jbuilder markedsføres som en omgivelse for integrering av industrielle applikasjoner, er JBuilder også profilert som et hjelpemiddel for å lette designprosessen og oppnå hurtig utvikling av prototyper.

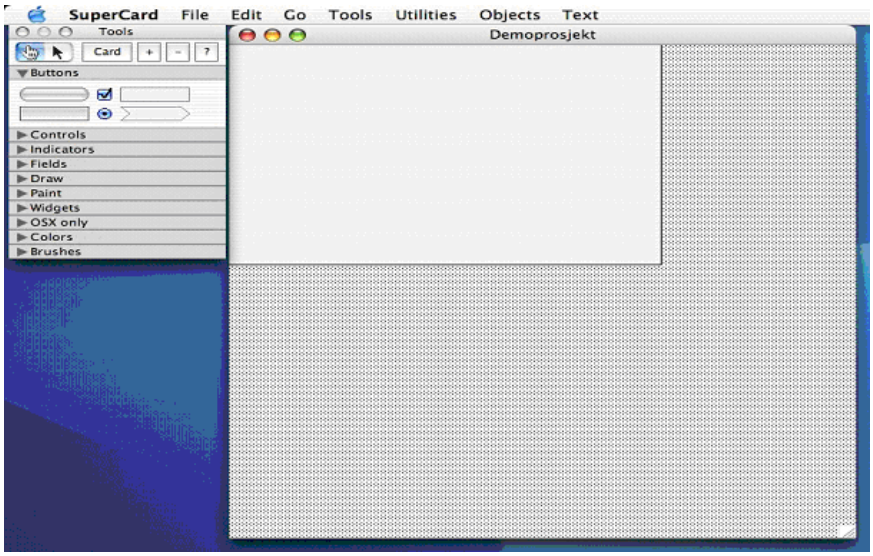
Utviklingsomgivelsene er imidlertid tilegnet til ulike formål, fra Flash som er mye brukt til å lage web-applikasjoner, til Director som er spesielt egnet til design av multimedia-applikasjoner, og til Jbuilder som brukes til utvikling av teknisk avanserte applikasjoner på Java-plattformen.

5.6.2 Supercard

Supercard er en utviklingsomgivelse som er beregnet for å lage interaktive multimedia applikasjoner og støtter elementer som grafikk, lyd, tekst, video og hypertekst.
Supercard baserer seg på metaforer som bakgrunn, stack og kort slik som sin forgjenger.

Man kan enten lage selvstendige applikasjoner (stand-alone) eller prosjekter som kan kjøres ved hjelp av en Supercard-player. Selvstendige Supercard-applikasjoner kan kun kjøres på Macintosh plattformen.

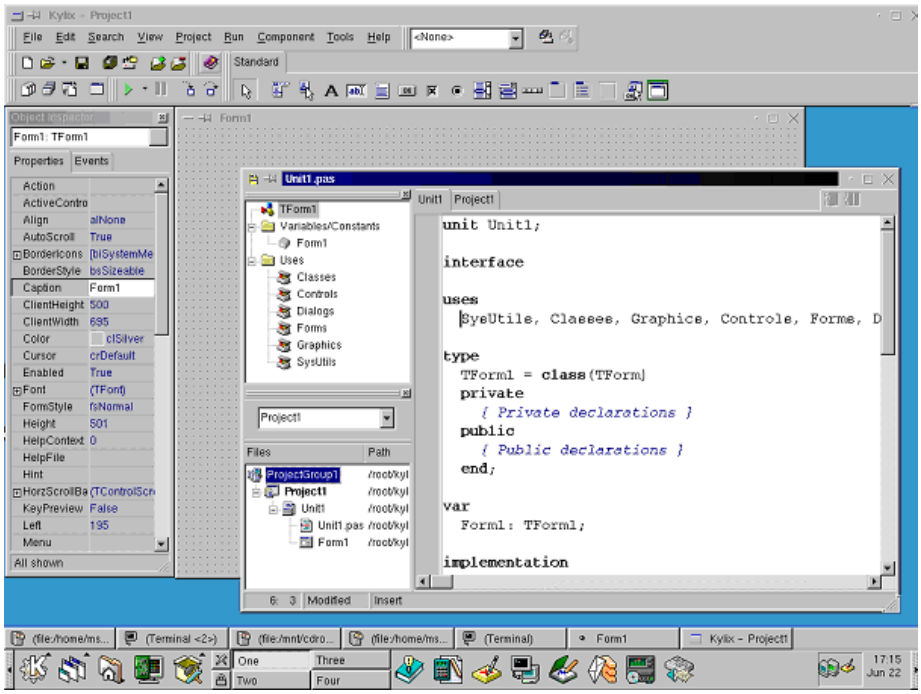
Mediaelementene kan manipuleres ved hjelp av programmeringsspråket SuperTalk, som er et basert på 'naturlig-språk-prinsippet'.
SuperTalk er i likhet med Visual Basic et hendelsesbasert og objektorientert språk som har en stor mengde multimedia-hendelser tilgjengelig for prototyping av multimedia-programmer og andre grafikk –og lyd intensive applikasjoner.
Supercard tilbyr standard grensesnittobjekter som kan legges ut på kort eller på bakgrunner i design-editoren og man kan også designe egne grensesnittobjekter. Stack'en er en framtredende metafor og er betegnelsen for hele applikasjonen som lages. Objekter - som for eksempel kortene i stack'en, kan ha egne metoder og egenskaper i tillegg til de som arves fra overklassen stack.
Supercard støtter en utvidbarhetsarkitektur kjent som XCMD grensesnittet, som gir brukere anledning til å legge til utvidet funksjonalitet og nye kommandoer til SuperTalk skriptspråket.



Figur 5.1 Hovedelementene i Supercard er en design-editor, en kode-editor og en verktøylinje

5.6.3 JBuilder

JBuilder er en utviklingsomgivelse som er basert på Java, et rent objekt-orientert programmeringsspråk. Jbuilder benytter den ordinære "form"-metaforen som Visual Basic benytter seg av der formene bygges opp ved å dra grafiske objekter over fra verktøylinja.



Figur 5.2 JBuilder med design-editor, kode-editor og verktøylinje som sentrale elementer

Komponentene som benyttes i det grafiske brukergrensesnittet kalles JavaBeans. De kan være synlige eller usynlige, slik som ActiveX komponenter. JBuilder er egnet for utviklere som

ønsker å lage applikasjoner som kan kjøres på mange plattformer. Java-applikasjoner og JavaBeans kjøres på en Java Virtual Machine og er derfor plattform-uavhengige.
Som i Visual Basic er det mulig å gjenbruke komponenter, men Jbuilder gir ikke like stor fleksibilitet til å lage et grafisk brukergrensesnitt som Macromedia Director.

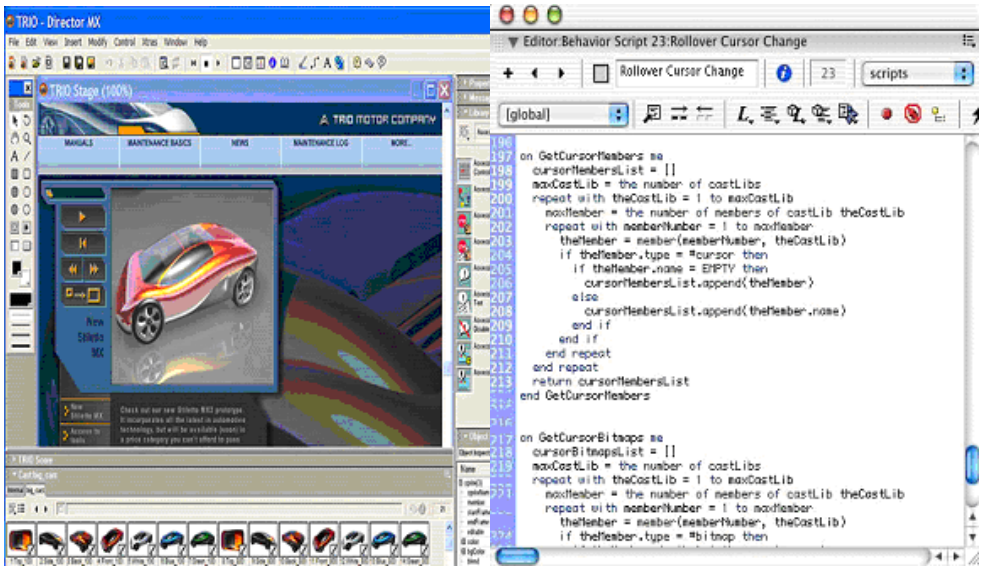
5.6.4 Director

Director er en utviklingsomgivelse som er beregnet for å lage interaktive multimedia applikasjoner. Director lar brukere integrere flere ulike multimedia-elementer som grafikk, lyd, tekst, video, hypertekst og 3D modeller i en og samme applikasjon. Man kan kontrollere hvordan disse elementene skal oppføre seg og hvordan de skal presenteres.
Applikasjoner laget i Director blir lagret i et bestemt format som heter Shockwave. Man trenger en shockwave-spiller for å kjøre en Director applikasjon.

Director forsøker å gi designeren en illusjon av å være filmregissør når han jobber med multimedia-elementer. Hovedmetaforen er filminnspilling og andre metaforer som scener, roller, tidslinje osv. brukes til å støtte opp om denne illusjonen.

Director tilbyr stor fleksibilitet når det gjelder å designe brukergrensesnitt, da det er enkelt å integrere grafiske multimediaelementer utviklet i andre program eller internt i Director ved å bruke verktøy som håndterer dette.

Macromedia Director har et programmeringsspråk (scripting) som kalles Lingo. Språket er inspirert av Supertalk, språket som brukes i Supercard. I Director 2004 MX kan også Javascript benyttes.



Figur 5.3 Hovedelementene i Director er en design-editor, en kode-editor og en verktøylinje

I tillegg til dette kan utviklingsomgivelsen utvides ved hjelp av Xtra-komponenter, da Director har en plugin-arkitektur. Denne arkitekturen gir mulighet til enten å utvide utviklingsomgivelsens funksjonalitet ved at man lager verktøy som kan operere mot ulike

elementer i utviklingsomgivelsen, eller til å utvide funksjonaliteten til applikasjonen man har laget (produktet).
Dette gir rom for tredje-parts utviklere til å lage ny funksjonalitet som kan benyttes av folk som designer Director applikasjoner. Xtra plugins må ligge i en bestemt plugins-katalog hvor de leses av utviklingsomgivelsen under oppstart og presenteres i en meny. For å utvikle en Xtra-plugin trenger man en Xtra API (Application Programming Interface).

5.6.5 Flash

Flash er en utviklingsomgivelse som er beregnet for å lage grafiske grensesnitt for Web. Flash har utviklet seg til å støtte ulike multimedia-elementer som grafikk, lyd, video og hypertekst.

I likhet med Director benytter Flash filminnspilling som metafor. Tidslinjen er særlig sentral som metafor i Flash som utviklingsomgivelse. I Flash MX 2004 tilbys imidlertid "form"-metaforen som et alternativ, for utviklere som er vant med Visual Basic.

Macromedia Flash er velegnet for å designe grafiske brukergrensesnitt på grunn av den fleksibiliteten den gir ved å la brukeren designe sine egne grafiske komponenter. Disse grafiske komponentene kan manipuleres ved hjelp av skriptspråket Actionscript. Flash kan importere en god del multimedia-elementer som er utviklet i andre program slik at man kan lage kraftige multimedia-applikasjoner. Flash er velegnet for horisontal prototyping, men har også en god del funksjonalitet rettet mot internett-teknologi slik som javascript, databaser, XML og videostreaming.

Figur 5.4 Hovedelementene i Flash er en design-editor, en kode-editor, en tidslinje, en verktøylinje og et komponentpanel med grafiske komponenter.

Tidslinjen benyttes for å lage animasjoner og tilstander i Flash-applikasjoner. I design-editoren kan man legge ut multimedia-elementer i tillegg til å lage egne grafiske objekter. Hvert grafiske objekt har en egen tidslinje og kan inneholde egne Actionscripts.

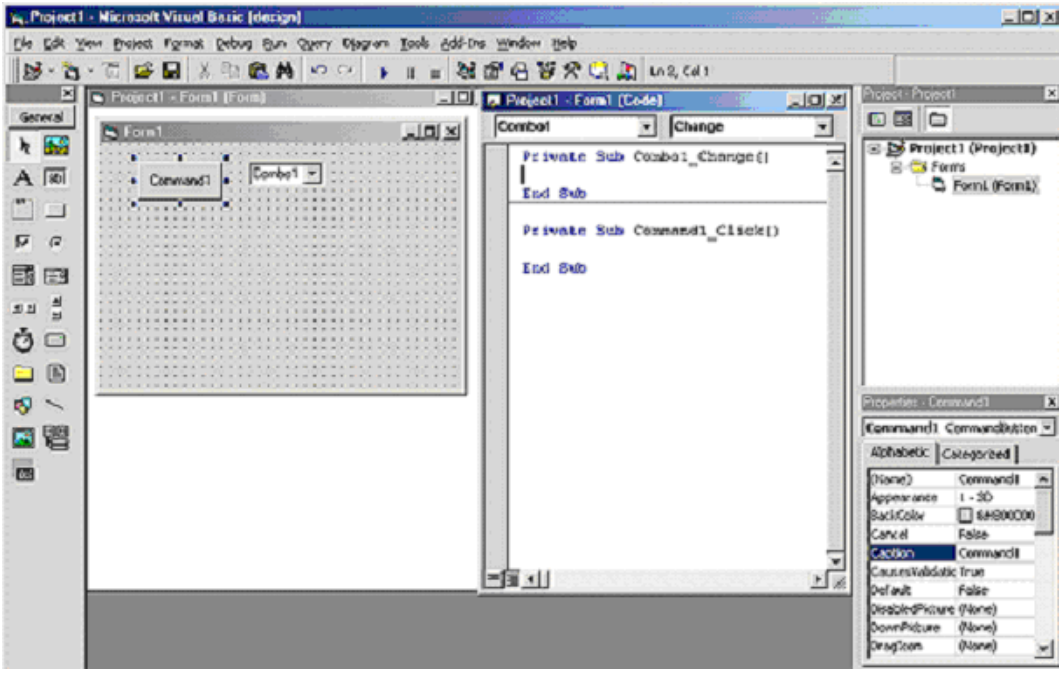
Applikasjoner laget i Flash blir lagret i et bestemt format som heter SWF. Man trenger en Flash Player for å kjøre en Flash-applikasjon. Flash applikasjonen kan enten være en selvstendig applikasjon (stand-alone) eller integrert i en Web-browser hvor Flash Player er installert. Det gis mulighet for å integrere Flash-player og SWF-fila i ei kjørbar fil. Flash Player finnes for Macintosh og Windows/PocketPC plattformen.

Da jeg jobbet med oppgaven tilbød ikke Flash noen muligheter til å utvide funksjonaliteten slik at man for eksempel kan prøve ut radioteknologi, men det var ventet at det ville komme.

5.6.6 Visual Basic

Visual Basic er opprinnelig et hendelses - og objektorientert programmeringspråk som ble utviklet av Microsoft som et verktøy for at utviklere på windows-plattformen skulle kunne utvikle egne applikasjoner med grafiske grensesnitt. Visual Basic er imidlertid mer enn et programmeringspråk og er en utviklingsomgivelse som blant annet tilbyr følgende verktøy :

- Design-editor (design av det grafiske brukergrensesnittet). Denne editoren representeres ved ulike 'forms' og bygger på WYSIWYG-prinsippet.
- Kode-editor. Her defineres hva som skal skje når det for eksempel trykkes på en knapp.
- Verktøylinje som rommer standard eller egendefinerte grensesnittobjekter.



Figur 5.5 Visual Basic med designeditor, kodeeditor og verktøylinje som sentrale elementer

Visual Basic gir mulighet til å lage fullt funksjonelle applikasjoner, i tillegg at den egner seg til å lage prototyper. Visual Basic er ikke like fleksibel som Director til å designe grafiske brukergrensesnitt, siden utviklingsomgivelsen baserer seg på gjenbruk av ferdige komponenter laget av andre og ikke er spesielt beregnet for å lage grafiske brukergrensesnitt og integrere multimedia-elementer slik som Director er.
Visual Basic er likevel et kraftig prototypingsverktøy som kan brukes i mange ulike sammenhenger og som gjør det enkelt for designere å benytte elementer fra andre applikasjoner når de utvikler prototyper.

I Visual Basic kan man benytte predefinerte komponenter slik som knapper, tekstbokser og scrollbars som kan legges i verktøylinja og brukes til å bygge opp brukergrensesnittet med. Disse komponentene kan gis oppførsel basert på events (hendelser) (Brookshear, 2003). I kodeeditoren angir man hva som skal gjøres når en bestemt event inntreffer - som for eksempel et musklikk.

ActiveX er en type komponent som er mye brukt i Visual Basic. ActiveX benyttes gjerne i det visuelle brukergrensesnittet, men kan også være usynlig og kun inneholde metoder.
ActiveX kan gi Visual Basic-utvikleren tilgang til utvidet funksjonalitet og til å integrere ulike grafiske elementer i en applikasjon.

Visual Basic støtter også 'Dynamic Link Libraries' (DLL).
I Visual basic koden kan man da benytte seg av data og funksjonalitet som ligger i DLL'en. Man må på forhånd vite metodenavnene og variablene som skal benyttes. Mens Visual Basic framstiller ActiveX-komponenter i brukergrensesnittet når de har blitt lastet inn, kan DLL'er kun refereres til direkte i programkoden (kode-editoren).
Vanlige DLL'er tilbyr ikke egenskaper som hendelser og meldinger, og er i så måte ikke så fleksible som ActiveX.

Del II

Komponent og test

6. Bluetooth-komponenten

6.1 Innledning

Oppgaven var å gjøre Bluetooth-teknologi tilgjengelig i en utviklingsomgivelse, og det var derfor nærliggende å velge den utviklingsomgivelsen som var best egnet til dette formålet. Grensesnittet mot Bluetooth-teknologien skulle være så enkelt som mulig, så det var viktig at utviklingsomgivelsen hadde et rammeverk rundt seg som gjorde det mulig å tilby et slikt grensesnitt.

Jeg vil i følgende kapitler gjøre rede for hvilke valg av utviklingsomgivelse og grensesnitt mot Bluetooth som ble gjort og hvordan denne løsningen ble seende ut.
Jeg vil videre diskutere ulike aspekter ved grensesnittet ut fra komponentens egenskaper.

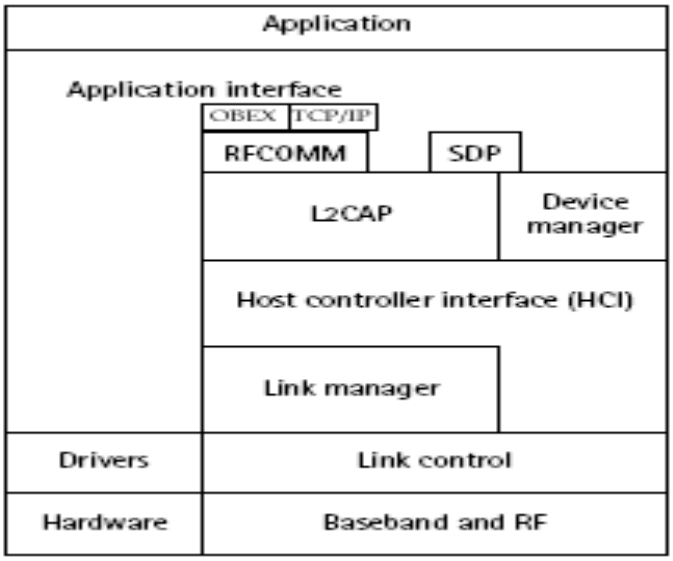
6.2 Bluetooth-teknologiens kompleksitet

Grensesnittet fungerer som en abstraksjonsmekanisme som utgjør en overbygning over Bluetooth-funksjonaliteten slik at den kan prototypes i en utviklingsomgivelse.

Jeg presenterer bare et utvalg av den funksjonaliteten som ligger i Bluetooth-teknologien, men designeren vil i det minste kunne få et realistisk møte med en del sentrale aspekter ved teknologien i sine prototyper der komponenten benyttes.
For å illustrere kompleksiteten til Bluetooth vil jeg nedenfor se litt nærmere på de tekniske spesifikasjonene som ligger til grunn for funksjonaliteten som komponenten presenterer et forenklet grensesnitt mot. Det var blant annet tekniske spesifikasjoner på dette nivået jeg måtte forholde meg til da jeg tok fatt på oppgaven med å abstrahere viktige Bluetooth-begrep.

Bluetooth er en trådløs kommunikasjonsteknologi for å utveksle data mellom mobile enheter og for å erstatte kabler mellom stasjonære og mobile enheter, eksempelvis mellom mobiltelefoner og head-sets (for mer om Bluetooth se kapittel 2.3 og 6.2).
Det er ulike begreper knyttet til Bluetooth-teknologien som er sentrale for å forstå teknologien. Bluetooth er imidlertid relativt komplisert og utilgjengelig for de som ønsker å prøve ut funksjonaliteten i egne programmer/prototyper. Jeg har valgt å abstrahere ut bestemte aspekter ved teknologien som jeg mener det er viktig for en designer å forholde seg til og ha kjennskap til når han skal prototype Bluetooth i en utviklingsomgivelse.

Bluetooth-teknologien utgjør ulike lag i en Bluetooth-protokollstack.
For å vise den kompleksiteten som designeren slipper å forholde seg til når han benytter grensesnittet, men som utgjør funksjonaliteten som skal demonstreres i prototypen, vil jeg nå se kort på hvordan Bluetooth-stack'en er bygd opp. Deretter vil jeg dra ut begreper som jeg syntes det var viktig å abstrahere ut og framstille i grensesnittet.



Figur 6.1 Versjon 1.1 av Bluetooth Specification Protocol-stack'en

De nederste lagene er essensielle for Bluetooth-teknologien og er basis for de andre protokollene og tjenestene i stack'en :

- **RF** (Radio Frequency) og **baseband** som sender og mottar de digitale radiosignalene som inneholder data eller tale.
- **Link-kontroll** enhet som prosesserer de sendte/mottatte dataene. Bluetooth-enheter kan sende og motta signaler på samme tid og dataene blir overført i form av datapakker som blant annet inneholder informasjon om hvor de er sendt fra og hvor de skal sendes.
- **Link Manager** som setter opp og administrerer forbindelser mellom Bluetooth-enheter.

I laget over disse protokollene ligger Host Controller Interface, som er grensesnittet mot Bluetooth-maskinvaren. Den tilbyr et grensesnitt til Link Manager og maskinvarens registre.

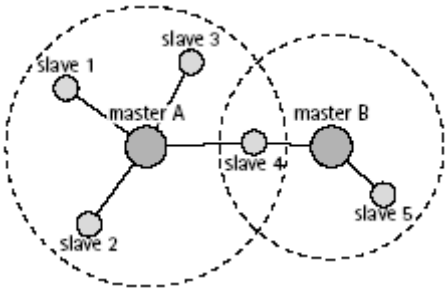
- **L2CAP** (Logical Link and Adaption Protocol) benytter dette grensesnittet til å tilby forbindelsesorienterte og forbindelsesløse datatjenester til protokollene i øvre del av stack'en. L2CAP gir muligheter for protokoll-multipleksing, segmentering og reassemblering og etablerer kontakt med andre kommunikasjonsprotokoller slik som f.eks SDP (Service Discovery Protocol) og RFCOMM.

- **RFCOMM** (Radio Frequency Communication Protocol) gir støtte for serieport-emulering over L2CAP protokollen og støtter opptil 60 koblinger samtidig mellom to enheter. RFCOMM gir grunnlaget for øvre nivås protokoller som TCP/IP og OBEX.

- **OBEX** (Object Exchange Protocol) er en kommunikasjonsprotokoll for å gjøre det lettere å utveksle binære objekter mellom Bluetooth-enheter.

- **SDP** (Service Discovery Protocol) gjør det mulig for applikasjoner å oppdage hvilke tjenester som er tilgjengelig og bestemme egenskapene til disse tjenestene. I omgivelser der Bluetooth benyttes varierer det hvilke tjenester som er tilgjengelige, siden Bluetooth innebærer spontane adhoc-forbindelser mellom mobile enheter med ulike egenskaper. Kalenderen til en PDA kan for eksempel bli synkronisert automatisk med en PC når den kommer innen rekkevidde av den.

Det lanseres fortløpende nye versjoner av denne spesifikasjonen, da Bluetooth er en relativt ny teknologi (lansert i 1998) og det utvikles Bluetooth stack'er for forskjellige plattformer som skal samsvare med spesifikasjonene i stack'en, noe som kompliserer det ytterligere.

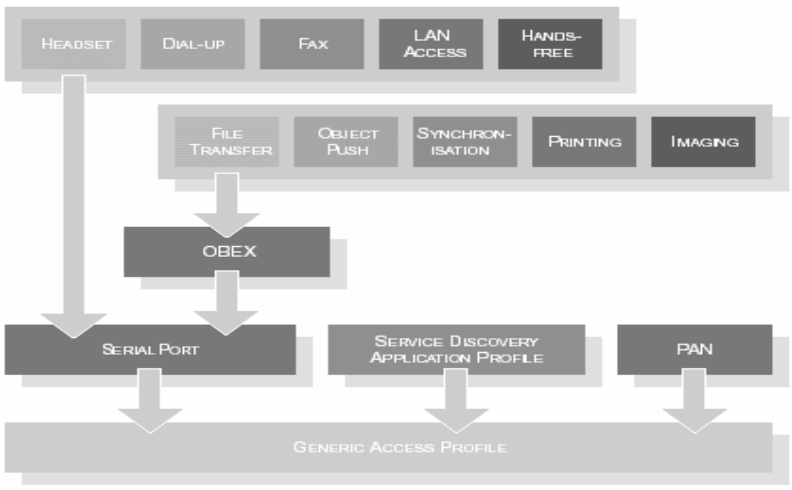


Figur 6.2 Piconet og scatternet

I tillegg til punkt-til-punkt forbindelse tilbyr Bluetooth-teknologien multipunktforbindelse mellom enheter. En samling enheter som er forbundet på en slik måte kalles et 'piconet'. Piconet kan i sin tur knyttes sammen til et 'scatternet'.
Siden komponenten jeg har utviklet ikke tilbyr piconetfunksjonalitet nøyer jeg meg med å si at i et piconet er den enheten som starter forbindelsen master mens de andre opp til sju enhetene er slaver.

Det er knyttet ulike bruksområder til Bluetoothkommunikasjon. Enhetene kan tilhøre forskjellige enhetsklasser slik som Fax, headset, PC og LAN.
Enheter av en bestemt enhetsklasse tilbyr gjerne omtrent de samme tjenestene og inngår i en eller flere profiler. Bluetooth-teknologien benytter profiler som spesifiserer hvordan bluetooth protokollstack'en skal brukes i forhold til bruksområdene for enhetene og enhetsklassene. Profilene definerer hvilke tjenester som kan utføres. Det er viktig at de aktuelle enhetene støtter profilen for det bestemte bruksområdet for at enhetene skal kunne kommunisere.

Eksempel på profiler er Dial-up Networking (DUN), Lan Access Point(LAP) og Fax. Antallet profiler vil sannsynligvis øke når nye bruksområder og applikasjoner oppstår.



Figur 6.3 Bluetooth-Profiler

Det er ikke tatt hensyn til profiler i komponenten siden den bare benyttes på PDA'er (av en bestemt enhetsklasse). Men siden det er knyttet bestemte bruksområder til PDA'ene som benyttes, og komponent-grensesnittet har metoder for å finne hvilke tjenester som tilbys på enhetene, vil profilbegrepet likevel spille en rolle. Hvis PDA'ene skal kommunisere med andre enhetstyper som f.eks. en Fax eller en PC (dongle) blir spesifikke bruksområder/profiler relevant.

Det er altså ikke formålet med grensesnittet å få demonstrert alle mulige tekniske aspekter ved Bluetooth, men heller å demonstrere den funksjonaliteten som designer synes er interessant. Dette kapittelet illustrerer behovet for et forenklet grensensitt for å verne designeren mot de komplekse tekniske detaljene. For å kunne lage prototyper som demonstrerer hvordan teknologien virker, må abstraksjonsnivået tilpasses designerens måte å forestille seg teknologien på.

6.3 Abstraherte Bluetoothbegreper

Når man står overfor en ny teknologi er det gjerne naturlig å spørre seg hvilke begrep som kan knyttes til teknologien. Nedenfor presenterer jeg forslag til begrep som kan abstraheres ut og inngå i en konseptuell modell av Bluetooth. De abstraherte begrepene er markert med **fet skrift**.

- Noe av det første en person som skal sette seg inn i Bluetooth tenker på kan være **kommunikasjon** og **enheter** (devices). Det kan deretter være naturlig å spørre seg hvilke **typer enheter** og hva slags kommunikasjon det er snakk om, siden det finnes flere typer enheter som godt kan kommunisere seg i mellom på forskjellige måter. Bluetooth kommuniserer **trådløst** mellom enheter ved hjelp av **radiobølger.**

- Enhetene identifiseres ved en MAC-adresse og har egne **enhetsnavn**, såkalte friendly names.

- Bluetooth er beregnet for kommunikasjon innen en begrenset radius (10 meter), så **rekkevidde** og **signalstyrke** kan derfor sies å være sentrale aspekter ved teknologien.

- Det er også vesentlig hvilken **informasjon** som skal kommuniseres.
  Bluetooth kan blant annet benyttes til å **utveksle objekter** som bilder, lydfiler og tekst.

- **Søk** etter enheter er et annet sentralt begrep i Bluetooth. Man kan få enheter til å søke etter andre enheter innen 10 meters avstand og lagre funnede enhetene på den søkende enheten. Fra **enhetslista** kan man så velge den eller de enhetene man vil utveksle informasjon med.

- Enhetene kan tilby ulike **tjenester**, slik som å sende filer eller tekst, opprette en internettforbindelse osv.

- Et annet viktig begrep er **profiler**. Bluetooth er ment å skulle støtte ulike bruksområder, såkalte Bluetooth-profiler. I en profil tilbyr de ulike kommuniserende enhetene forskjellige tjenester til hverandre.
  Head-set er eksempel på en slik profil. Det innebærer gjerne kommunikasjon mellom et headset og en mobil enhet (mobiltelefon, PDA etc.). Lyd representert ved en **datastrøm** sendes fra den mobile enheten til head-set'et.
  Profilbegrepet er imidlertid ikke tatt med i grensesnittet (se kap. 6.2), men er representert av en metode som sjekker tjenesten til en gitt enhet. Hvilken tjeneste enheten tilbyr, kan si noe om hvilken enhetsklasse den tilhører og hvilke profiler den støtter.

Bluetoothbegrep	**Abstrahert begrep**	**Grensesnitt (API mot eVB)**
Metoder:		
Enhetsøk i frekvensområdet etter tilgjengelige enheter	**Søk**	**StartSok ()** **StoppSok ()** **StartStoppSok (tid)**
Enheter	**Enheter**	**Enhet /enhetsnavn**
EnhetsID og MAC-adresser	**Enhet/enhetsnavn**	**OppgiMittNavn()** **EndreMittNavn(mittenhetsnavn)** **OppgiNavnTilknyttetEnhet()**
Enheter knyttes sammen og blir utilgjengelige for andre enheter: Bonding	**Bonding**	**OpprettBond(pinkode, enhet)**
OBEX-protokoll (Object exchange)	**Utveksling av informasjonsobjekter**	**SendFil( )** **SendBusinessCard(enhet)** **MottaBusinessCard( )** **UtveksleBusinessCard(enhet)**
Signalstyrke	**Signalstyrke**	**OppgiSignalstyrke(enhet)**
Service Discovery Protocol (SDP)	**Sjekke tjeneste**	**Sjekktjeneste(tjeneste, enhet)**
Enhetsklasse (device class)		
Profiler		
Serieportforbindelse Comport	**Sende tekststreng mellom to enheter**	**SendTekst( )** **LesTekst( )**
Autentisering	**Autentisering ved pinkode**	Parameteren **Pinkode** i bla. **OpprettBond()**
Events:		
Søk er startet	**Søk er startet**	**_SokStartet ()**
Søk er avsluttet	**Søk er avsluttet**	**_SokStoppet ()**
Enhet funnet i søk: OnDeviceFound		**_EnhetFunnet (enhet)**
Tekst kommer inn i buffer via serieport/comport	**Tekst mottas**	**_TekstMottatt ()**
OBEX-fil sendes	**Filforsendelse pågår**	**_OnSendFileProgress**
OBEX-fil er sendt	**Fil sendt over**	**_OnSendFileComplete**

Figur 6.4 Abstraksjon av Bluetooth-begrep i grensesnitt

Det var begreper som dette jeg tok utgangspunkt i da jeg laget et grensesnitt mot Bluetooth. Til venstre i tabellen er Bluetoothbegrepene slik jeg måtte forholde meg til dem da jeg fikk oppgaven å lage et forenklet grensesnitt mot Bluetooth. I midten er (endel av) disse begrepene forsøkt forenklet. I høyre kolonne ser vi hvordan begrepene vises i grensesnittet. Det er dette designeren forholder seg til.
Jeg anså at det var viktig at grensesnittet ble presentert på en slik måte at begrepene kom klart fram, men at den nødvendige funksjonaliteten samtidig var tilgjengelig.

6.4 En konseptuell modell av Bluetooth-teknologien

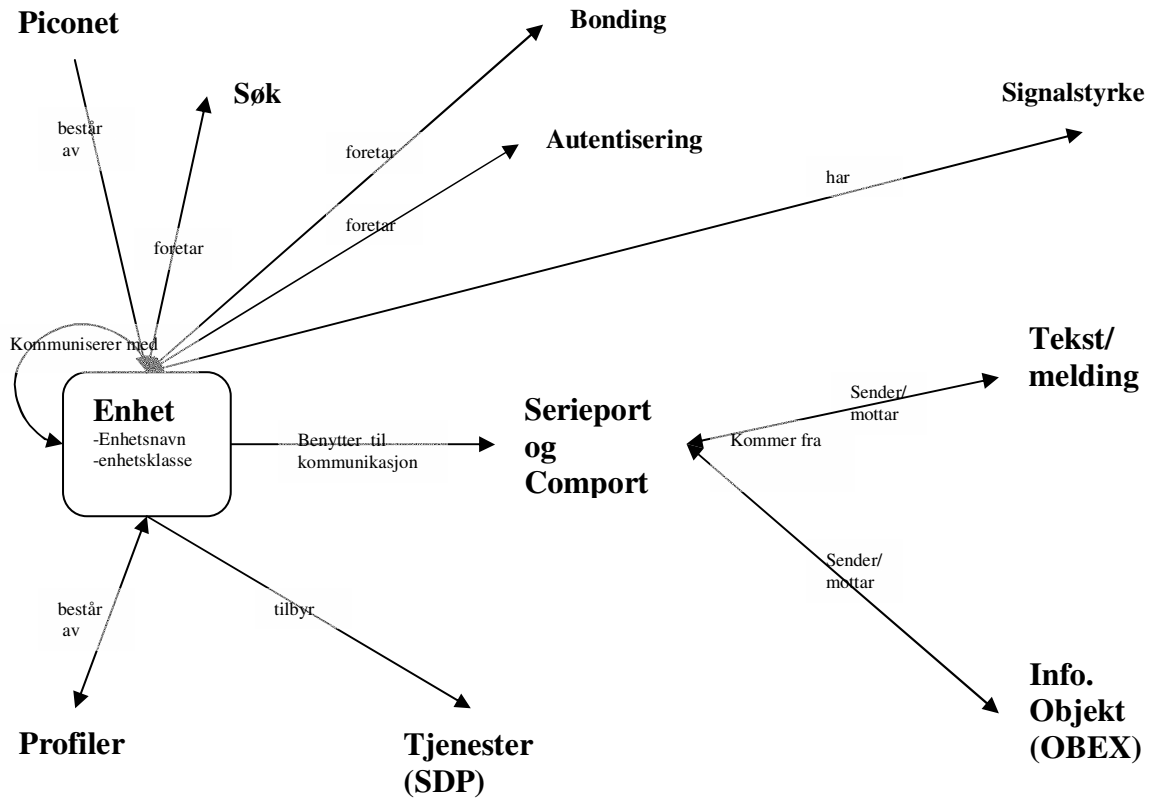
For å forstå Bluetooth er det hensiktsmessig å få på plass et begrepsapparat knyttet til aspekter ved teknologien og å begripe hvilken betydning de ulike begrepene har for anvendelsen av teknologien.
Jeg ville presentere Bluetoothteknologien i en forenklet innpakning som kommuniserte de begrepene som jeg mente designeren hadde nytte av å ha kjennskap til.
Grensesnittet skulle reflektere en konseptuell modell som ga designeren et utgangspunkt for å forstå teknologien.

En konseptuell modell består ifølge Johnson og Henderson av sentrale begreper, forholdet mellom begrepene, den overordnede designmetaforen samt andre relaterte metaforer, sammenhengen mellom de abstrakte begrepene og hvordan disse kommer til uttrykk i interaksjonsdesignet (Johnson og Henderson, 2002). Den konseptuelle modellen til Bluetooth er det overordnede 'imaget' jeg vil formidle til de som skal sette seg inn i teknologien. Når jeg presenterer grensesnittet for designeren vil jeg at han skal forbinde teknologien med visse begreper som til sammen utgjør den konseptuelle modellen (se kapittel 4.4).

Jeg vil nedenfor se på Bluetooth-grensensittet med hensyn til Johnson og Hendersons punkter for konseptuelle modeller.

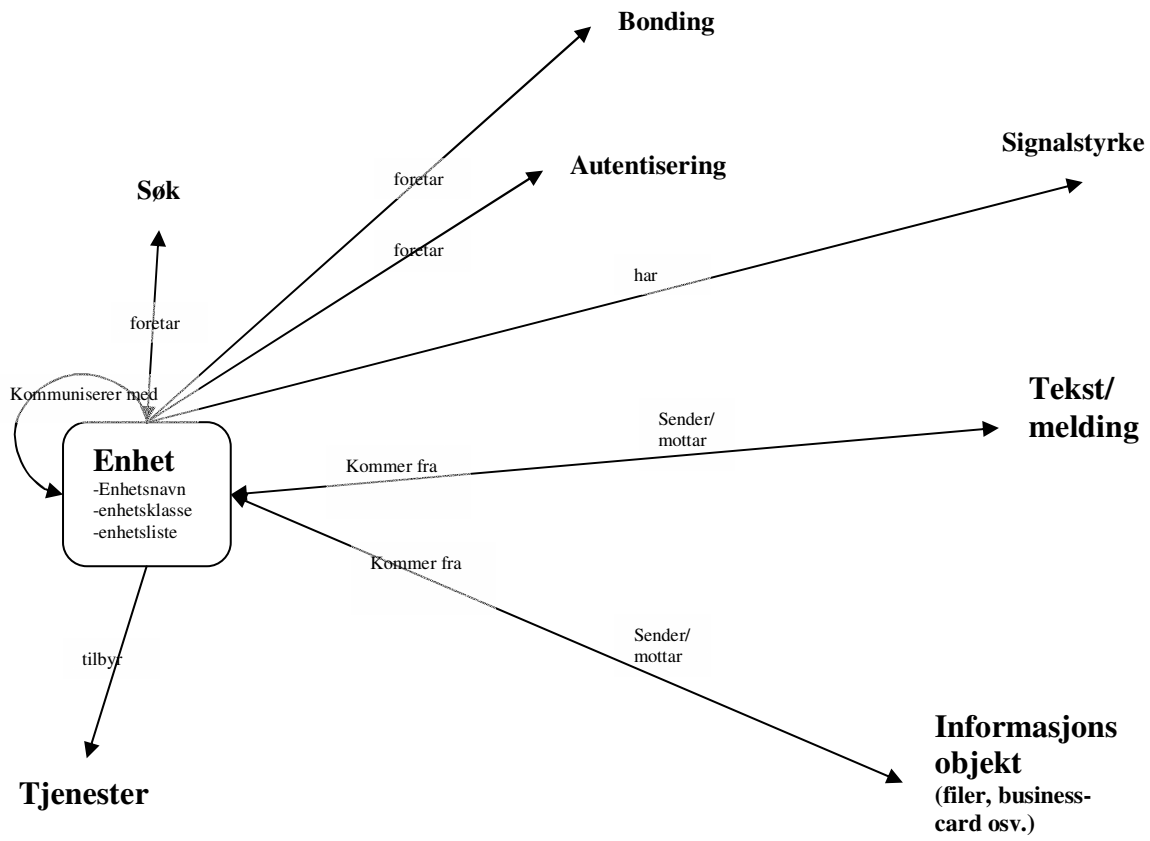
Begrep jeg anser som sentrale i modellen er - som nevnt i kapittel 6.3 - enhetstyper, trådløs kommunikasjon, enhetsnavn, søking etter enheter, enhetslister, rekkevidde, informasjonsobjekter og tjenester.

Forholdet mellom begrepene kan utdypes nærmere.
Før begreper som serieportforbindelse og comport er abstrahert vekk angir den konseptuelle modellen for Bluetoothkommunikasjon blant annet at det først gjøres et søk etter omkringliggende enheter. Det kommuniseres så med de funnede enhetene ved å etablere serieportforbindelse og inngående/utgående com-port åpnes opp. Forholdet mellom de sentrale begrepene i den konseptuelle modellen blir som i figur 6.5 og 6.6.
Figur 6.5 viser den opprinnelige Bluetoothbegrepsmodellen, mens figur 6.6 viser modellen etter at begrep som serieport og Piconet er abstrahert vekk.



Figur 6.5 Opprinnelig begrepsmodell for Bluetooth

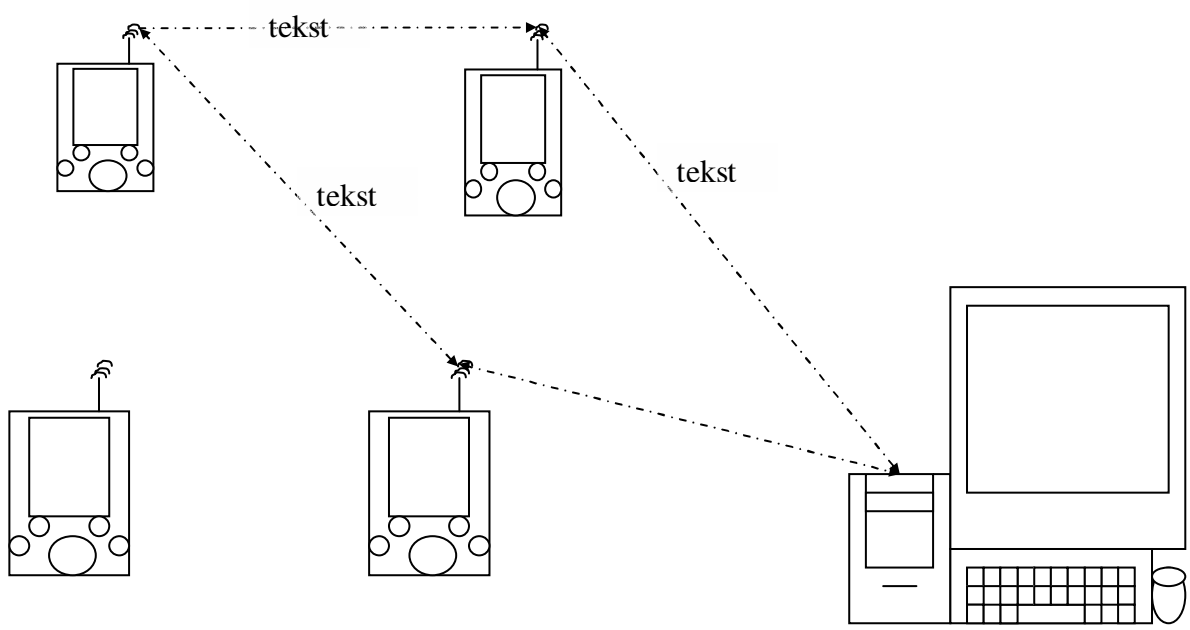
Denne modellen viser begrepene og noen tekniske spesifikasjoner - implementasjonsdetaljer kommer ikke fram her - men man kan se i figur 6.6 at begrepene Serieport og Comport er tatt vekk. Da grensesnittet ble utviklet i C++, ble det benyttet en API som var en overbygning over Bluetooth-stack'en. I implementeringen av funksjonaliteten bak grensesnittet måtte jeg forholde meg til begrep som serieport og comport samt en del kompliserte datastrukturer for å utføre kommunikasjonen. Å utveksle meldinger/tekst mellom to enheter uten noen fastlagt høynivåprotokoll som TCP/IP er ingen enkel sak. Jeg kom til at oppretting, åpning og lukking av serieport og comport til riktig tid, hvor kommuniksjonen ikke innebærer noe fastlagt klient/tjener forhold, ikke er forenlig med hurtig prototyping. Derfor ble behandling av serieport og comport automatisert i metodene SendTekst og LesTekst.



Figur 6.6 Abstrahert (forenklet) begrepsmodell for Bluetooth

Den viktigste metaforen i min presentasjon av Bluetooth kan sies å være selve kortdistanse-kommunikasjonen mellom enhetene. Poenget er å få designeren til å dra nytte av tidligere erfaring med lignende teknologi. De fleste personer har en innarbeidet forestilling om hva man legger i en telefonsamtale og hvordan man går fram for å gjennomføre en slik telefonisk kommunikasjon. En sammenligning kan gi designeren en mental modell også av Bluetooth-kommunikasjon, og den konseptuelle modellen til Bluetoothgrensesnittet kan inneholde en slik metafor for kommunikasjon.

Figur 6.7 viser hvordan jeg ser for meg at designeren oppfatter kommunikasjonen mellom enheter. Designeren forestiller seg at Bluetooth åpner opp for å sende beskjeder fra en enhet til andre enheter, akkurat som at man ved hjelp av telefoner kan kontakte hvem som helst som er tilknyttet telenettet, eller at man kan sende tekstmeldinger (SMS eller MMS) mellom mobiltelefoner via basestasjoner.
Bluetooth gir spontan adhoc forbindelse mellom enheter innen rekkevidde av hverandre, men like fullt vil en som skal sette seg inn i BT-teknologien kunne ha nytte av å sammenligne med teknologier de kjenner fra før. Det benyttes en metaforisk sammenligning med teknologier som allerede er kjent.



Figur 6.7 PDA'er og en PC utveksler tekst /meldinger

Sammenhengen mellom de abstrakte begrepene og den **fysiske manifestasjonen** av dem kan det være vanskelig å si noe presist om, siden grensesnittet består av en usynlig ActiveX komponent. Designer utformer selv det grafiske brukergrensesnittet og den fysiske manifestasjonen blir derfor avhengig av hvilke grep designeren gjør.
Designeren vil imidlertid forventes å utforme et brukergrensesnitt som han synes representerer Bluetooth-teknologien best mulig.
Den konseptuelle modeller legger dermed visse føringer på hva designeren syns er hensiktsmessig å kommunisere i prototypens design og hvilke interaksjonsformer som benyttes.

6.5 Valg av Visual Basic som utviklingsomgivelse

For å legge til rette for prototyping av Bluetooth-teknologi, så jeg etter utviklingsomgivelser som kunne benyttes til hurtig prototyping og som åpnet opp for utvidet funksjonalitet.
Det var meningen at man skulle få en mest mulig realistisk demonstrasjon av teknologien. Designeren skulle kunne teste ut kommunikasjonsteknologien ordentlig, så det var ikke nok å simulere trådløs teknologien ved hjelp av skjermbilder eller lignende. For å kunne legge til rette for realistisk prototyping av teknologien måtte utviklingsomgivelsen gis tilgang til den underliggende funksjonaliteten.
Utviklingsomgivelsen måtte derfor kunne håndtere et eksternt grensesnitt for å kommunisere mot Bluetooth. Et slikt bindeledd kan være en 'Dynamic Link Library' (DLL) eller en prekompilert komponent. Poenget er at bindeleddet må kunne forstå språket til Bluetooth (Bluetooth-stack'en) og formidle dette videre til utviklingsomgivelsen.

Bluetooth er en teknologi for radiokommunikasjon mellom ulike typer data - og kommunikasjonsutstyr (se kapittel 2.3 og 6.2). Det ble lett etter utstyr som støttet Bluetooth-kommunikasjon og valget falt på IPAQ 5450 Personal Digital Assistants (PDA) med innebygd Bluetooth. Ipaqene kjørte på WinCE operativsystemet, så det sto mellom et grensesnitt laget i en C++-omgivelse med en tilhørende WinCE API og en Bluetooth API som basis og et grensesnitt laget i en Java-omgivelse (med Java Virtual Machine) med en Java API og en Bluetooth API for Java som basis. Bluetooth-API'en "BT-access" som vi landet på var skrevet i C++ og vår utviklingsomgivelse måtte ha støtte for WinCE. Jeg valgte derfor C++-omgivelsen Embedded Visual C++ 3.0.

Disse mobile enhetene (PDA) kunne programmeres, på samme måte som en vanlig PC. Applikasjonene/prototypene ble utviklet i en utviklingsomgivelse som var installert på en vanlig PC, men som kompilerte til Win32-plattformen og så overførte dette til de mobile enhetene. Hver gang designeren ville prøve ut endringer han hadde gjort på prototypen, ble prototypen overført til PDA.

Kravet til utviklingsomgivelsen som skulle benyttes var for det første at den støttet den mobile plattformen og for det andre at den ga muligheter til programmering av bluetooth-funksjonalitet. Da jeg vurderte de utviklingsomgivelsene som er nevnt i kapittel 5.6, kom jeg fram til følgende :

- Supercard kunne ikke brukes, da PDA'ene (Ipaq5450) kjørte på et WinCE OS og Supercard kun var tilgjengelig for Macintosh.
- Jbuilder ble også valgt bort, da denne innbar bruk av en Java Virtual Machine plattform, og jeg ikke fant noen brukbar Java-API (Application Programming Interface) mot Bluetooth.
- Director ga ikke støtte for utvikling mot mobile enheter, så jeg måtte velge bort denne.
- Flash støttet den mobile plattform, men det viste seg nær sagt umulig å få tilgang til bluetooth-stack'en.
- Visual Basic sto da igjen. Embedded Visual Basic (som var del av pakken Embedded Visual Studio 3.0 sammen med Embedded Visual C++ 3.0), støttet utvikling mot mobile enheter og ga muligheter til å integrere et bindeledd i utviklingsomgivelsen mot den underliggende bluetooth-funksjonaliteten, i form av en komponent med et enkelt grensesnitt. Utviklingsomgivelsen som jeg brukte til å kommunisere med Bluetooth-API'en og WinCE API'en med(Embedded Visual C++) hadde støtte for OLE Automation og ActiveX slik at jeg kunne lage metoder som var tilgjengelige i Embedded Visual Basic.

Jeg landet således på Embedded Visual Basic (eVB) som utviklingsomgivelse for prototyping av Bluetooth, da eVB ga mulighet til å benytte komponenter med enkle grensesnitt, slik at designere hurtig kunne lage prototyper som demonstrerte Bluetooth-teknologien.

Grensesnitt-teknologien ActiveX ble løsningen for å få Embedded Visual Basic til å kommunisere med Bluetooth-teknologien.

6.6 ActiveX som grensesnitt mot Bluetooth

ActiveX er en teknologi som egner seg til å abstrahere teknologiske begrep og "pakke inn" teknologien i et enkelt og veldefinert grensesnitt. Grensesnittet presenteres på en slik måte at en designer lett kan skjønne hvordan han benytter det. Det som er avgjørende for at det skal la seg gjøre å hurtig prøve ut teknologien, er at abstraksjonsnivået til Bluetooth-grensesnittet er tilpasset designers problemløsning.

- Grensesnittet skal ha en lav terskel for å få Bluetooth til å fungere fysisk i prototypen
- Designer skal ikke bli heftet av uventede hindringer under designet av prototypen.
- Grensesnittet skal være rettledende. Designer må for eksempel kunne forstå ut i fra navnet på en metode hva den kan benyttes til og hvordan.

ActiveX gir en arbeidsdeling mellom ulike applikasjoner. Visual Basic anvender metoder, variabler, hendelser, meldinger osv. utviklet i C++ og eksponert til Visual Basic gjennom ActiveX-grensesnittet. Dette er gunstig fordi C++ egner seg godt til å programmere funksjonalitet og å benytte operativsystemets API (Application Programming Interface) og importerte biblioteksrutiner. Programmereren kan få til nær sagt alt av funksjonalitet med C++ ved å benytte operativsystemets API (Application Programming Interface) og importerte DLL'er og andre bibliotek. C++ med sine avanserte datastrukturer og datatyper har en høy terskel og er tidkrevende å programmere i. C++ tilbyr derfor ikke riktig abstraksjonsnivå for hurtig design av prototyper. ActiveX gjemmer bort problematikk rundt datatyper og strukturer i C++ komponenten, og lar designeren benytte seg av et forenklet grensesnitt til å løse oppgaven i Visual Basic – på et høyere abstraksjonsnivå.

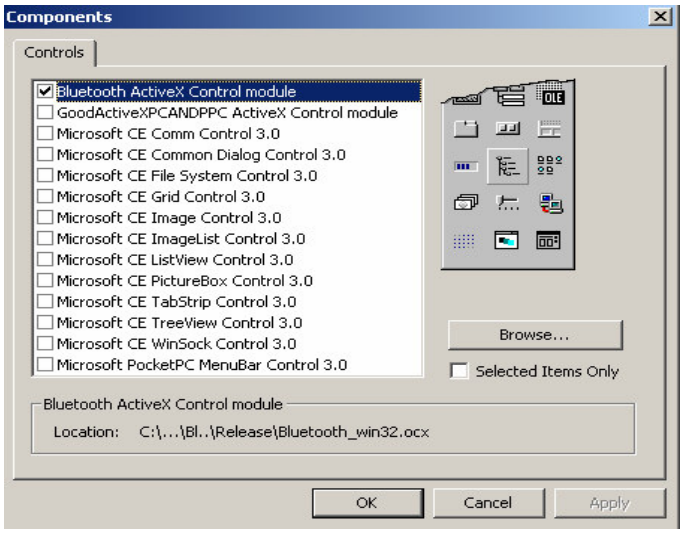
Det er her samarbeidet mellom applikasjonene kommer inn. Det er nemlig fordelaktig at man overlater de programmeringsmessig krevende oppgavene til C++ og heller benytter et språk som Visual Basic til selve problemløsningen. Designeren kan dermed løse prototypings-oppgaven ved hjelp av enkel programmering med kall til komponentens grensesnitt.

6.7 Bruk av komponenten i Embedded Visual Basic(eVB)

ActiveX kontrollen ble laget for å gjøre det mulig å prototype Bluetooth-funksjonalitet på mobile enheter. Prototypen blir programmert og utformet på en stasjonær PC i Embedded Visual Basic. Man bygger opp et skjermbilde i design-editoren og programmerer funksjonaliteten bak skjermbildet ved hjelp av kode-editoren.
Når man vil prøve ut prototypen, overføres den til de aktuelle PDA-enhetene ved hjelp av USB og programvaren ActiveSync. Identiske instanser av programmet/prototypen blir altså kjørt på PDA'ene. Ingen av enhetene får noen bestemt klient eller tjener-rolle, men programmet/prototypen kan lages slik at enhetene f.eks. forespør hverandres tjenester eller sender hverandre meldinger/tekst som den andre svarer på. Hvordan dette gjøres, definerer designer i eVB-programmet.

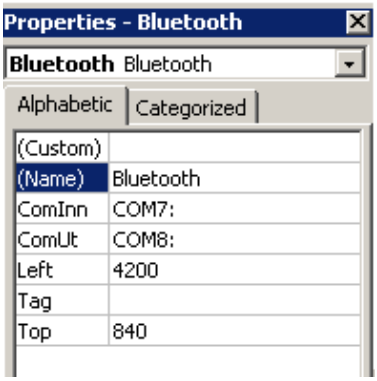
For å kunne benytte komponenten må eVB få oppgitt en refereranse til komponentens plassering, slik at eVB kan tolke nødvendig metainformasjon om komponentens grensesnitt. eVB verifiserer samtidig om komponenten er en "lovlig" ActiveX-kontroll, slik at den kan legges i verktøylinja.

ActiveX komponenten som lastes inn er den som er kompilert for bruk på PC (Win32) plattformen. ActiveX komponenten som benyttes av prototypen i eVB må ligge på PDA'en, og den må registreres i windowsregisteret på både PC'en og PDA'en. Begge versjonene av ActiveX komponenten må ha samme ID, slik at det kan refereres til samme grensesnitt under utvikling som under kjøring av prototypen. Så lenge grensesnittene har samme ID, betyr det ikke noe at det er to forskjellige versjoner av komponenten - en for Win32 og en for WinCE.

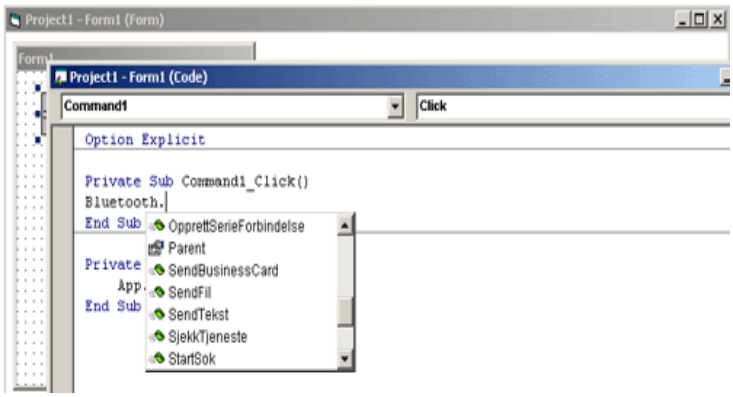


Figur 6.8 Her lastes ActiveX komponenten inn i Embedded Visual Basic

Komponenten kan nå benyttes i kodeeditoren ved å oppgi navnet på komponenten etterfulgt av en metode, attributt eller event. Komponentnavnet og andre attributter kan endres i property-editoren som er vist i figur 6.9a.
Komponentens metoder og attributter kommer opp automatisk når komponentnavnet skrives inn i kode-editoren. Eventer kommer ikke opp automatisk. For at dette skal være mulig må alle metodene/attributtene være med når komponenten kompileres på Win32, slik at eVB kan forbinde disse metodene/attributtene med komponentnavnet.

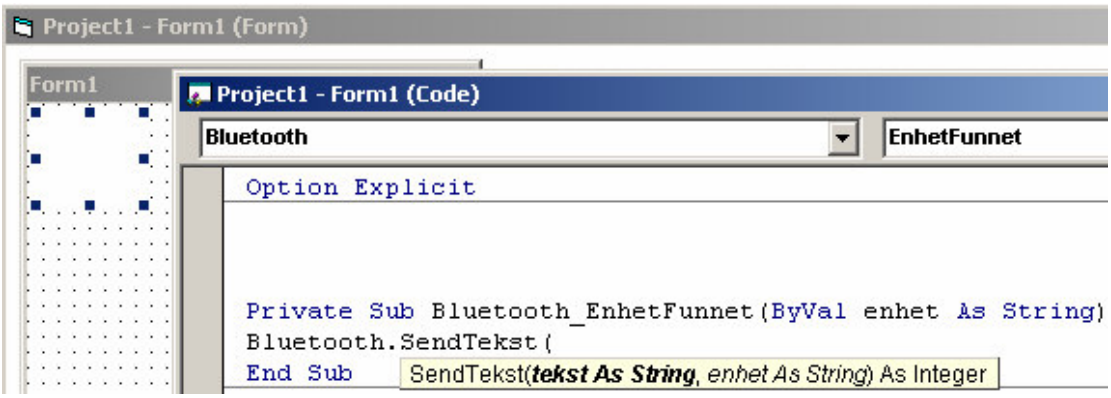


Figur 6.9a
Den redigerbare Property-editoren

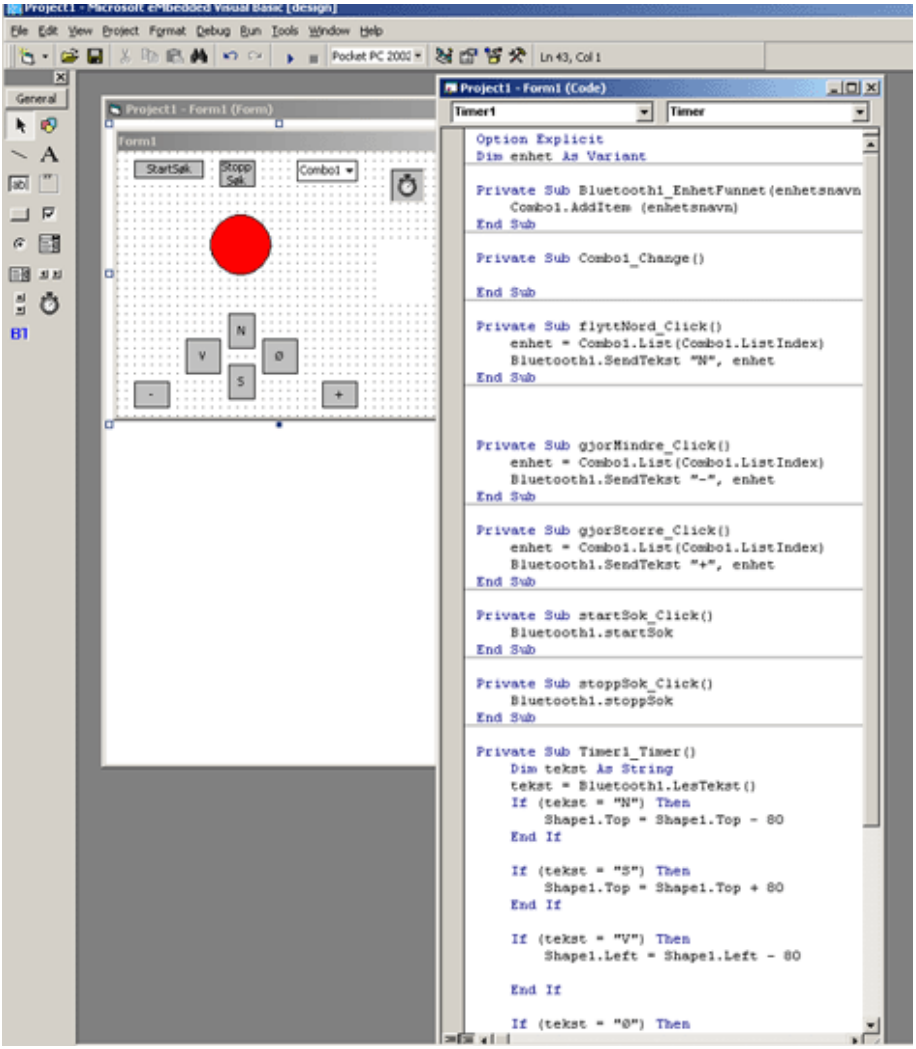


Figur 6.9b
Slik vises komponentens tilgjengelige metoder og attributter i eVB

For å kunne referere til komponenten i programkoden, så må den settes inn i en 'form' slik at den blir en aktiv del av prototypen som skal lages. Komponenten får et instansnavn som designeren kan redigere. Instansnavnet benyttes for å referere til komponenten i programkoden. Komponenten er ikke synlig i grensesnittet når applikasjonen kjøres.



Figur 6.10 Komponentens ligger i formen og metodene/eventene kan ta ulike parametre



Figur 6.11 Prototyping av Bluetooth-funksjonalitet i eVB

Figur 6.11 viser verktøylinja, design-editoren og kode-editoren til et eVB-prosjekt der det er designet en enkel Bluetooth-prototyp. ActiveX komponenten ligger i kontrollpanelet til venstre og dras over i design-editoren (form'en) for å kunne aksesseres i programkoden. Komponenten ligger usynlig i bakgrunnen i design-editoren og blir betraktet som et objekt i programkoden, med tilhørende metoder og attributter. Designer forholder seg bare til grensesnittet, og komponenten blir dermed som en 'blackbox' hvor bare grensesnittet er tilgjengelig.

Metodene og attributtene er synlige i utviklingsomgivelsens grensesnitt når komponenten tas i bruk, mens eventene er usynlige.

Embedded Visual Basics funksjoner angir funksjonaliteten bak de enkelte GUI-objektene i design-editoren, unntatt funksjonen _Timer og _EnhetFunnet, som er egne eventer. De ordinære metodene fyres av når det gjøres handlinger mot GUI-elementene, for eksempel når det trykkes på en knapp. _Timer fyres av etter et visst tidsintervall, mens den egendefinerte eventen _EnhetFunnet fyres av når det forekommer et søkeresultat.

6.8 ActiveX-komponentens interaksjonsmekanismer

ActiveX ble utviklet for at programmerere kunne lage egne applikasjoner som automatiserte vanlige oppgaver (MSDN om automation og ActiveX, 2003). Ved å benytte en ActiveX-komponent med støtte for Automation er det mulig å få tilgang til funksjonalitet til utviklingsomgivelser som Visual Basic og ulike scriptingspråk på en enkel måte. ActiveX er en sammensmelting av COM-teknologien og Automation.

COM-teknologien representerer en standardisert måte for å la applikasjoner eksponere metoder og attributter til hverandre på.

Automation-mekanismen gjør komponenten i stand til å eksponere metoder og attributter til Visual Basic, og gjør komponenten er i stand til å motta hendelser fra en underliggende modul (for eksempel en DLL). Komponenten kan så programmeres til å fyre av meldinger videre til applikasjonen (containeren) som benytter den.

En ActiveX-komponent er lettere å benytte enn en ordinær COM-komponent, da man slipper å kalle operativsystemet direkte, noe som er mer komplisert.

Den som skal benytte komponentens grensesnitt bør ha en forestilling om hvilke mekanismer komponenten har for å utveksle data mellom komponenten og applikasjonen som benytter komponenten. Når man benytter ActiveX-komponenten i Visual Basic til å få tilgang til Bluetooth-funksjonaliteten kaller man i kodeeditoren komponentens eksponerte grensesnitt, som består av bla. metoder, attributter og events.

Designer må forstå hvordan han f.eks. skal programmere en event, da eventmekanismen er nødvendig for å få benytte Bluetooth-radioen. Det kan for eksempel enten være at det blir funnet en enhet i et Bluetooth-søk eller at brukeren gjør en handling direkte i programmet/prototypen. Disse eventene kan behandles i Visual Basic-applikasjonen. I komponenten behandles hendelser fra underliggende lag (API'er og Bluetooth-stack i form av DLL'er) som igjen sendes videre oppover fra ActiveX-komponenten til Embedded Visual

Basic-programmet. Dette gir muligheten til for eksempel å lagre funnede enheter og lese tekst dynamisk når den blir mottatt.
ActiveX-komponentens grensesnitt benytter følgende mekanismer for å samhandle med Visual Basic og andre aktuelle utviklingsomgivelser :

- **Metoder**: Komponenten tilbyr metoder som brukeren av komponenten kan benytte. Metodene kan ha en parameterliste hvor brukeren angir input-verdier for at metoden skal kunne utføre den bestemte funksjonaliteten.
  Hver metode gir fra seg en viss returverdi. For eksempel kan en søkemetode gi fra seg et bestemt resultat eller det kan angis om en metode har lyktes eller ikke.
- **Attributter**: Komponenten tilbyr attributter (variabler) som brukeren kan endre fra Visual Basics grensesnitt eller i kode-editoren.
- **Get/Set-metoder**: I stedet for at et attributts verdier tilordnes eller hentes ut direkte gjøres dette ved hjelp av attributtets tilhørende Get/Set-metoder.
- **Events (hendelser)**: Komponenten tilbyr hendelser som kan behandles av brukeren av utviklingsomgivelsen (designeren). Komponenten programmeres slik at det fyres av bestemte handlinger under visse vilkår. ActiveX-events responderer på interrupts fra underliggende lag. Events er en abstraksjon av interupts(på lavere nivå). Eventene kan også ta parametre.

- **Notifikasjonsmeldinger**: Når verdien til et attributt endres blir dette oppfattet som en hendelse av ActiveX-komponenten og det fyres av en metode. Dette er en kombinasjon av attributt og event.

6.9 Flere konseptuelle modeller definerer komponentens anvendelsesmodell

Det er viktig at designeren forstår Visual Basics konseptuelle modell, som bygger på den visuelle uttrykksmåten til de grafiske grensesnittobjektene.

Visual Basic har både objektorienterte og eventorienterte egenskaper og den konseptuelle modellen tar utgangspunkt i at det opprettes grafiske objekter som knapper, inputboks, osv. i brukergrensesnittet (design-editor). Det tilordnes så funksjonalitet tilknyttet de ulike objektene i designeditoren.
I Visual Basic er events knyttet til hva designeren har definert at skal skje når det gjøres handlinger på grensesnittet eller ved bruk av forhåndsdefinerte eventer. I ActiveX-komponenten skjer det imidlertid events på grunnlag av meldinger fra underliggende hardware/lavnivåhendelser. ActiveX tilbyr altså eksterne eventer som bidrar til å utvide det som er mulig å gjøre i utviklingsomgivelsen.

I tillegg vil jeg som tilrettelegger av Bluetooth-grensensittet at designeren skal ha en forestilling om de Bluetooth-teknologiske begrepene. Den konseptuelle modellen av Bluetooth presenteres ved å forklare designeren viktige Bluetooth-begreper, forklare hvordan de ulike aspektene ved teknologien virker og ved å sammenligne med beslektet kommunikasjonsteknologi. Det kan naturligvis hende at designeren vet en del om Bluetooth allerede, men forestillingen han har fra før vil forhåpentligvis være i samsvar med modellen som presenteres i grensesnittet og tilhørende dokumentasjon.

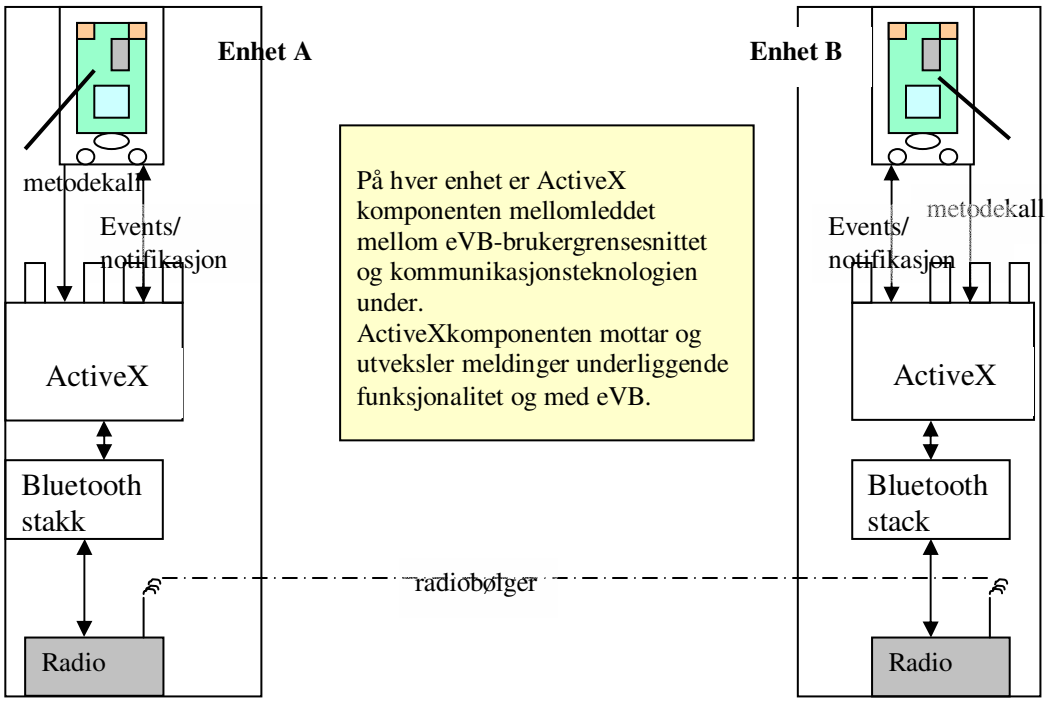
Den konseptuelle modellen til Bluetooth kan sies å være en utvidelse av utviklings-omgivelsens modell, da den representerer den funksjonelle utvidelsen av Visual Basics muligheter. Prototypen lages i eVB og kjøres på PDA'er som tas med fritt rundt i rommet og kommuniserer trådløst. Designeren er oppmerksom på dette når prototypen når den blir laget og må ta stilling til Bluetooth-begrepene når han benytter grensesnittet.

Det kan også knyttes en anvendelsesmodell til ActiveX-grensesnittet i VB. Denne kan sies å være en helhetlig modell som rommer både utviklingsomgivelsen – og teknologiens konseptuelle modell. I tillegg vil måten interaksjonsmekanismene til ActiveX er utnyttet på i komponenten/grensesnittet være en integrert del av anvendelsesmodellen (se kap. 9.2.3).

Med flere konseptuelle modeller som forholder seg til hverandre, er det en utfordring å gi designeren et helhetlig bilde av hvordan han skal prototype Bluetooth i Visual Basic.
Det var viktig å utforme grensensittet slik at det var i samsvar med både Bluetooths - og utviklingsomgivelsens konseptuelle modell. Det måtte kjennes så intuitivt og som mulig å benytte grensensittet til å lage en prototyp. Med dette for øyet vil jeg ta for meg noen sentrale metoder og eventer i grensesnittet.

6.10 Hvordan interaksjonsmekanismene ble utnyttet i grensesnittet

Jeg utviklet en ActiveX-komponent for å gi designer mulighet til å lage interaktive prototyper som demonstrerer bruk av Bluetooth. Tanken var at grensesnittet ville være relativt enkelt å forstå for designeren, slik at det kunne benyttes til hurtig prototyping av teknologien. Designer ville imidlertid likevel måtte skjønne hvordan han skal benytte interaksjons-mekanismene til grensesnittet (metoder, events osv.).



Figur 6.12 ActiveX som mellomledd

Designeren må forstå at når grensesnittets metoder og eventer benyttes, så utveksles det data gjennom Bluetooth-radioen, avhengig av hva som gjøres i enhetens brukergrensesnitt (og på kommuniserende enheter).
Bluetoothradioen både sender og mottar data. Komponenten har visse mekanismer (se kapittel 6.8) som den benytter til å ta i mot og sende meldinger til Bluetooth-radio og Bluetooth-stack og til å samhandle med Visual Basic. Komponenten mottar og behandler handlinger i eVB-prototypens brukergrensesnitt. Ut fra hvilke metoder/eventer som er tilordnet eVB-brukergrensesnittet vil komponenten motta meldinger/data fra underliggende radio/stack eller sende ut data for å kommunisere med andre enheter. I tillegg til handlinger i eget brukergrensesnitt vil naturligvis komponenten motta meldinger/data fra andre enheter som forsøker å kommunisere med enheten.

Grensesnittet ble utformet slik at det skulle være så brukervennlig og enkelt å bruke, samtidig som den viktigste funksjonaliteten var tilgjengelig.

Jeg vil nå se nærmere på en del av metodene og eventene i grensesnittet hvor jeg forklarer hvordan grensesnittet ble designet. Det er verdt å merke seg at grensesnittet jeg presenterer kun er en bestemt løsning på problemet. Det finnes også andre måter å utforme grensesnittet på (se kap.8 om forbedringer av grensesnittet).

Når det gjelder metoden **StartSok** så ble grensesnittet utformet slik at man først utfører et søk etter tilgjengelige enheter i nærheten og deretter kommuniserer med disse enhetene på ulike måter. Da grensesnittet ble utformet mente jeg at dette var i tråd med den konseptuelle modellen for Bluetooth. Enheter blir oppdaget/funnet og så opprettes det en forbindelse med en eller flere enheter.

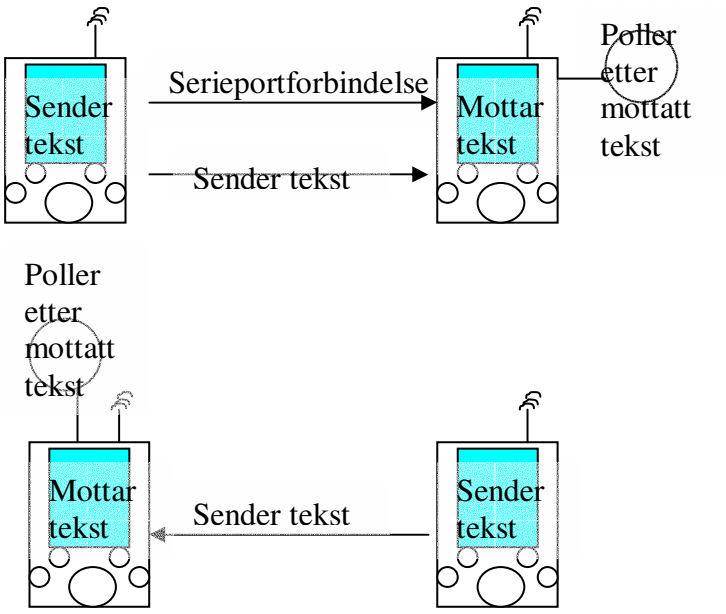
Det kan være flere måter å lagre de enhetene det søkes på. Komponenten kan for eksempel ta vare på enhetene i et array som eksponeres til VB. Jeg valgte å la designeren få lagre enhetene selv i for eksempel en listbox som tas fra verktøylinja.
Jeg valgte å lage en event **_EnhetFunnet** slik at enhetene kan legges inn i en lagringsstruktur etter hvert som de blir funnet. _EnhetFunnet fyres av hver gang en enhet oppdages.
Designeren kan selv definere hva som skal skje når denne eventen inntrer, men hvis målet er å ta vare på de enhetene man vil kommunisere med, er det hensiktsmessig å legge navnet til enhetene inn i en liste, for eksempel en listbox.
Siden det kan være noe uklart for designeren hva som faktisk blir resultatet av søket, er det hensiktsmessig å kunne få opp tilbakemelding i en liste, meldingsboks eller lignende på at enheter er funnet. Dermed kan designer se i prototypen på PDA'en at søkefunksjonaliteten fungerer før han går et skritt videre og benytter andre metoder til å kommunisere.

En del metoder forutsetter at andre metoder eller events allerede er kjørt.
For eksempel har metoden **OppgiSignalstyrke** parameteren enhetsnavn og trenger følgelig denne inputverdien for å kunne utføre metoden. Slik grensesnittet er utformet må enhetsnavnet hentes fra array/lista (for eksempel som element i combobox) og så sendes med som parameter. Metoden returnerer LAV, GOD eller HØY.

SjekkTjeneste har parameterne 'tjeneste' og 'enhetsnavn' og trenger navnet til en gitt tjeneste og enheten som tjenesten evtl. finnes på.
Metoden returnerer 1 eller 0, alt etter om den aktuelle tjenesten finnes på den gitte enheten.

SendFil fungerer på samme måte. Den tar 'enhetsnavn' som input og returnerer 1 eller 0, alt etter om fila ble sendt.

Når det gjelder metodene for utveksling av tekst – SendTekst og LesTekst, så er disse abstraksjoner av begreper som serieportforbindelser og com-porter. For at designeren skal slippe å ta seg av åpning og lukking av com-porter og oppretting og avslutning av serieportforbindelser, er dette skjult i komponenten. Administreringen av serieportforbindelsen og åpning/lukking av com-porter kan nemlig bli komplisert når flere enheter kan ha forbindelser til hverandre.
En Bluetooth-enhet kan for eksempel ha flere inngående serieportforbindelser- men bare en utgående forbindelse om gangen. Hvis man vil sende til en ny enhet må derfor forbindelsen avsluttes med den første enheten.
For at enhetene skal være klare til å motta og lese tekst helt fra programmet startes opp så initialiseres enhetene med en åpnet inngående com-port (com5).
I komponenten er derfor denne logikken skjult og erstattet med SendTekst og LesTekst.



Figur 6.13 Sending og mottakelse av tekst mellom PDA'er

SendTekst tar som parametre teksten som skal sendes og enheten teksten skal sendes til. Returverdien angir om teksten kom fram til mottaker.
SendTekst sjekker om det finnes en serieportforbindelse allerede, avslutter eventuelle andre utgående forbindelser og oppretter en forbindelse til enheten sendt som parameter.
Teksten blir så sendt til enheten via utgående com-port ved hjelp av en funksjon Writefile.

LesTekst benyttes for at teksten skal leses på mottakerenheten.
Denne metoden har ingen parametre men returnerer teksten som mottas.
LesTekst leser av det som kommer inn via den inngående serieporten til enhetens buffer og returnerer den teksten som ligger der.
Slik grensesnittet er utformet må utvikleren benytte Timer-eventen i Visual Basic til å polle etter innholdet i bufferet. Timeren settes for eksempel til å kjøre LesTekst funksjonen hvert

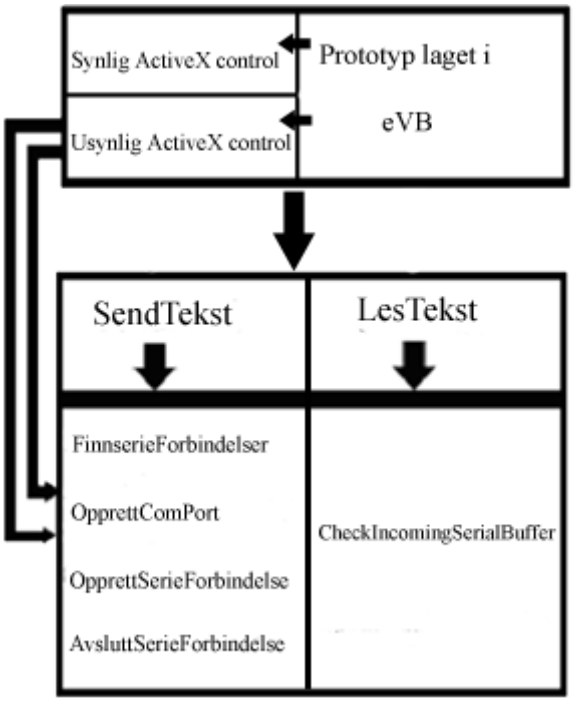
tredje sekund. På denne måten leses teksten dynamisk på PDA'ene med det samme den blir mottatt (avhengig av tidsintervall i Timer), uten at man trenger å gjøre noe aktivt i grensesnittet på PDA'en.

Det var meningen å tilby eventen **_TekstMottatt** som ligger som en egen tråd i bakgrunnen og sjekker på om det kommer tekst inn i inngående serieport. Det lot seg dessverre ikke gjøre å få denne metoden til å virke da komponenten ble utviklet, sannsynligvis pga. problemer med Visual Basic og operativsystemet på PDA'ene. Jeg syns likevel det er verdt å nevne denne hendelsen. Designeren ville sluppet å bruke 'Timer'-objektet/eventen i Visual Basic ved å benytte _TekstMottatt. Timeren stjeler en del ressurser på PDA'en og vi oppdaget at bruk av flere timere samtidig kunne medføre problemer. LesTekst og _Timer i kombinasjon er likevel tilfredsstillende. Det medfører kun
marginalt mer koding (en linje ekstra) og ytelsen er akseptabel så lenge prototypen er relativt liten.

Eventene **_SokStartet** og **_SokStoppet** kan brukes av designer til å programmere hva som skal skje når søket startes eller avsluttes. Det kan for eksempel være at man vil tømme listboxen hver gang et søk startes slik at listboxen bare inneholder enheter som blir funnet i det nye søket.

Jeg forsøkte å være konsekvent i valget av parametre og i navnet på metoder, eventer og parametre. Grensesnittet består hovedsakelig av metoder, hvor en del av dem har parametre. I _SendFil og _SendTekst la jeg som første parameter det som skulle sendes og som andre parameter den enheten det skulle sendes til.
Likedan er første parameter i **SjekkTjeneste** den tjenesten som skal verifiseres, og andre parameter er enheten som det skal sjekkes etter tjenester på.
Metoden **EndreMittNavn(mittenhetsnavn)** er eksempel på at jeg har forsøkt å la navngivningen til metodene, eventene, parametre osv. være så selvforklarende som mulig. Intensjonen var at navnene skulle gi designeren en god indikasjon på hva metodene/eventene gjør og hvilken rolle de spiller i prototypen.
Denne metoden endrer navnet til den enheten man sitter med og ikke noen av enhetene det kommuniseres med, så det var naturlig å bruke 'mitt' i metodenavn og parameternavn. Man oppnår dermed å skille mellom egen enhet og andre enheter slik at man unngår misforståelser.

Grensesnittet ble utformet med tanke på å tilby designeren tilgang til noen av de underliggende metodene som komponenten benytter internt. For eksempel kan metodene som SendTekst benytter lett eksponeres til eVB. Disse metodene er blant andre **OpprettSerieForbindelse**, **OpprettComPort**, **FinnSerieForbindelser** og **AvsluttSerieForbindelser** og **CheckincomingSerialBuffer** og utgjør et metodelag som ligger på et lavere abstraksjonsnivå enn metodene/eventene jeg ellers har omtalt. Disse kan da en viderekommen designer benytte som alternativ til den overbyggende metoden SendTekst. I figur 6.14 ser vi hvordan prototypen som designes i eVB benytter metoder som ActiveX-controlen tilbyr. Enten benyttes SendTekst og LesTekst, eller designer kan alternativt benytte metodene som disse består av.



Figur 6.14 Prototypen benytter metoder på ulike abstraksjonsnivå

Jeg har i utformingen/designet av grensesnittet også åpnet opp for et høyere abstraksjonsnivå enn de metodene jeg har omtalt ovenfor. Brukergrensesnittobjekter og metoder med funksjonalitet kan integereres i synlige ActiveX-komponenter (controls). Dette kan være metoder som benytter en eller flere av metodene til å utføre flere operasjoner på en gang med ett metodekall fra eVB (mer om forbedringer av grensesnittet i kap. 8).

ActiveX-komponenten ble testet for å se hvordan grensesnittet ble benyttet og om komponenten virkelig ble oppfattet som så nyttig og brukervennlig som det var meningen.

7. Test av komponenten

7.1 Problemstilling for testen

ActiveX-komponenten ble laget for å gjøre det lettere å prototype Bluetooth. Jeg har i forrige kapittel sett på hvordan grensesnittet er bygd opp og beskrevet de mest sentrale metodene og eventene.

Ved å teste prototypen har jeg kunnet få verdifull informasjon om hvordan det var å bruke ActiveX-grensesnittet til slik funksjonell prototyping av Bluetooth-teknologien. Etter å ha evaluert resultatet av foregående forsøk har jeg kunnet tilpasse og videreutvikle presentasjonen av grensesnittet noe til neste forsøk for å se om for eksempel mer fullstendig dokumentasjon gjør det lettere å lage prototyper.
I forsøkene ville jeg undersøke om det blir lettere å forholde seg til komponentens grensesnitt hvis den konseptuelle modellen til Visual Basic/ActiveX og Bluetooth ble presentert på en klar og intuitiv måte gjennom dokumentasjon og annen hjelp.
Jeg ville se på om designere klarte å danne seg et mentalt bilde av hvordan Bluetooth-teknologien fungerer og hvordan den kan benyttes, og om dette har betydning for bruken av prototypingen.
Det er helt sikkert mulig å utforme grensesnittet på alternative måter - abstraksjonene jeg har gjort av Bluetoothbegrepene er ikke de eneste riktige, men som det framgår av kapittel 6, har jeg hatt mye arbeid med det konseptuelle designet av grensesnittet før og mens komponentens grensesnitt og funksjonalitet ble implementert. Forsøkene gir en pekepinn på om de abstraksjonene jeg har gjort "gir mening" for designeren og er et nyttig middel for å videreutvikle Bluetooth-grensesnittet. Forsøkene blir således en viktig del av den iterative prosessen med å designe komponentens grensesnitt. Forsøkene gir verdifull evaluering av grensesnittets brukbarhet.
En annen av delproblemstillingene jeg berører i denne oppgaven er nødvendigheten av å utvikle realistiske og funksjonelle prototyper. I en del sammenhenger kan det nemlig være nok å lage look-and-feel og/eller rolleprototyper (se kapittel 1.4 og 4.5). Forsøkene kan gi svar på det hensiktsmessige ved å gi designeren en realistisk forsmak på funksjonaliteten til teknologien.

7.2 Testopplegget

Jeg vil her gi en kort beskrivelse av hver av de fire testene jeg gjennomførte.
Det ble gjennomført fire forsøk. Forsøkspersonene skulle lage funksjonelle prototyper for Bluetooth-kommunikasjon mellom PDA'er. Forsøkene gikk ut på at en forsøksperson satt sammen med forsøksleder og laget prototyper med Bluetooth-funksjonalitet. En fullt funksjonell prototyp skulle designes i løpet av ca en time.

I alle forsøkene assisterte forsøksleder forsøkspersonene hvis han/hun sto fast eller hadde bestemte spørsmål. Siden forsøkspersonene hadde assistanse fra forsøksleder til å lage prototypen kan ikke forsøkene gi noe absolutt svar på grensesnittets kvalitet eller nytte. Forsøkene fungerer først og fremst som et middel for å evaluere grensesnittdesignet og for å gi svar på delproblemstillingene i denne oppgaven.

Forsøkspersonene fikk først en kort presentasjon av Bluetooth, en beskrivelse forsøksopplegget og en kort innføring i Visual Basic. De fikk så vite hvilke oppgaver de skulle løse. I hver av forsøkene ble en prototyp laget i Embedded Visual Basic på en stasjonær PC. Prototypen ble overført og kjørt på de to aktuelle PDA'ene hver gang forsøkspersonen ville sjekke funksjonaliteten, skulle vurdere hvordan designet så ut (look-and-feel) eller se etter feil i koden. Alle forsøkspersonene fikk de samme oppgavene de skulle løse(se kap. 7.3).

Forsøksperson A hadde dokumentasjon og eksempelkode å støtte seg til. Forsøksperson B fikk ingen slike hjelpemidler.
Forsøksperson C fikk eksempelkode og noe mer fullstendig dokumentasjon. Dokumentasjonen ble forsøkt forbedret og gjort mer fullstendig ut fra negative tilbakemeldinger fra Forsøksperson A. I dokumentasjonen fikk hun presentert en konseptuell modell for bruken av Bluetooth-komponenten.
Forsøksperson D fikk de samme hjelpmidler som Forsøksperson C.

Ingen av de tre første forsøkspersonene hadde vært borti språket Visual Basic eller utviklingsomgivelsen Embedded Visual Basic. Forsøksperson D var derimot viderekommen. De tre første forsøkspersonene hadde grunnleggende kjennskap til Bluetooth, mens Forsøksperson D hadde inngående kjennskap.

7.3 Oppgaver

1. Få en enhet til å søke etter andre enheter
2. Stoppe dette søket
3.Legge enhet i combobox/listbox

4 Oppgi signalstyrke på tilknyttede enheter

5. Sende Fil

6. Sende tekst mellom enheter
7. Motta tekst og evenuelt behandle den

Fig 7.1a Et signal sendes fra høyre til venstre PDA med beskjed om å endre farge på sirkelobjekt

Fig 7.1b Venstre PDA mottar signalet (teksten) og objektet endrer farge til rød

7.4 Metode

Forsøkenes metodikk kan beskrives som en type "uformell" deltagende observasjon. Forsøkspersonene fikk tenke høyt (think-aloud) og alt ble tatt opp på video.
Etter hvert forsøk var det et kort intervju hvor forsøkspersonene uttrykte sin mening om forsøket. For mer om metoder som ble brukt i studiet generelt, se kapittel 3. Se også kritikk av metode i kapittel 3.4 og 11.2.

7.5 Utstyr

Det ble benyttet følgende maskinvare for under forsøket :
- 2-3 PDA'er med innebygd Bluetooth
- Cradle og ledning for å overføre data til PDA via USB.
- Videokamera

I tillegg til ActiveX-komponenten ble det benyttet følgende programvare for under forsøket :

- Embedded Visual Basic 3.0
- PocketPC 2002 SDK
- ActiveSync 3.7 for å overføre data fra PC til PDAer

7.6 Forsøk A. Dokumentasjon og eksempelkode som hjelpemiddel

Informatikkstudent på 26 år uten erfaring med Visual Basic, men med god generell programmeringskunnskap og datakunnskap.

Forsøkspersonen fikk tilgang til eksempelkode og dokumentasjon med beskrivelse av komponentens grensesnitt.

7.6.1 Observasjoner

Et nytt Visual Basic prosjekt ble åpnet og forsøkspersonen begynte å designe prototypen.

1. Få en enhet til å søke etter andre enheter
2. Stoppe dette søket
3. Legge enhet i combobox/listbox

-Han var usikker i starten på om ActiveX-kontrollen måtte lagres i design-editor (formen) eller i verktøylinja for å virke.
-Eksempelkode viste seg å være nyttig for å komme i gang. Kom lettere inn i syntaksen og slapp å prøve seg fram gang på gang for å få riktig resultat.

-Han skjønte tankegangen om at ActiveX-kontrollen var et objekt med tilhørende metoder. Dette så jeg av at han i kodeeditoren refererte til navnet han ga på kontrollen når han skulle finne Bluetooth-metoder.

-Han hadde ingen forestilling om hvordan han skulle benytte events. Dette ble ikke presentert til ham før testen og han hadde problemer med å forholde seg til dette begrepet. Trodde bla at combo-box'en var et objekt som skulle programmeres. Combo_change kan brukes i Visual Basic til å manipulere comboboksens egenskaper.
Han dvelte ved dette en stund. Ble gjort oppmerksom på _EnhetFunnet–hendelsen, men trodde hendelsen skulle stå inne i en annen eVB-metode (Combo_Change). Comboboksen skal kun refereres til for legge inn data og hente ut data fra den.

-Forsøkspersonen var ikke komfortabel med å måtte skrive inn hendelsen _EnhetFunnet manuelt i eVBs kodeeditor:

```
Private Sub Bluetooth1_EnhetFunnet(enhet)
   Combo1.AddItem (enhet)
End Sub
```

_EnhetFunnet kommer opp automatisk som default hendelse når man dobbeltklikker på kontrollen i design-editoren, men for andre hendelser må hendelsen skrives inn manuelt.

-Forsøkspersonen misforsto dette og ble forvirret.
Visste ikke hvordan han skulle forholde seg til hendelser i forhold til metoder. Jeg måtte forklare ham at hendelser er en slags metoder som inntrer ved for eksempel trykk på en knapp eller en definert hendelse som at en bluetoothfil er sendt eller at en timer aktiverer noe.
Et objekts metoder og attributter kommer opp grafisk i ei liste (metode/attributtliste) i VB når det refereres til det i kode-editoren mens hendelser som nevnt må skrives inn eksplisitt hvis det ikke er default-hendelsen. Testpersonen prøvde å finne hendelsen i lista men fant den ikke.

-Forsøkspersonen fant ut etter hvert at _EnhetFunnet skulle stå som egen eVB-metode i kodeeditoren. Han hadde fortsatt problemer med å forstå hva events egentlig var for noe og hvordan han benyttet dem.
Han stilte spørsmål som "Hva er eventer i forhold til metoder? " og "Hvor fikk jeg enhet fra?". Han trodde man ikke trengte å lagre enhetsnavnene noe sted, at dette ble gjort av komponenten.

4. Oppgi signalstyrke på tilknyttede enheter

-Forsøkspersonen trodde at man kunne søke etter enheter og få signalstyrke i en operasjon uten å ta i betrakning at man må velge en bestemt enhet. Skjønte hvordan han skulle bruke OppgiSignalstyrke-metoden etter at denne misforståelsen var oppklart.

5. Sende Fil

-Hadde ingen problemer med denne metoden. Brukte kunnskapen han hadde fra forrige oppgave da han anga valgt enhet i combo-box som parameter. Hadde eksempelkode tilgjengelig, noe han valgte å benytte her.

6. Sende tekst mellom enheter
7. Motta tekst og eventuelt behandle den

-Klippet og limte SendFil metoden til Sendtekst og gjorde små endringer og tilpasninger i koden her.

-Glemte at LesTekst metoden (for å motta tekst) returnerte en tekststreng.

7.6.2 Kort intervju etter forsøket

I intervjuet etter forsøket poengterte forsøkspersonen følgende:

Han lurte på om navnet på enheten kunne skrives direkte inn som argument i metodene. Sa at han ikke skjønte hvor den funnede enheten kom fra, men at han nå skjønte at han måtte lagre enhetsnavnet et sted for å kunne kommunisere med de aktuelle enhetene.

-Han trodde at søket etter enheter måtte stoppes med metoden stoppSok før han kunne gjøre andre operasjoner.

Forsøksperson A satte fram følgende forslag til forbedring av søkemetoden, der det ble sendt med en tidsparameter i søket slik at han selv kunne kontrollere i hvor lang tid søket skulle vare.

StartSok(tid)

Ellers så syntes forsøkspersonen at komponenten var forholdsvis enkel å bruke, bortsett fra de nevnte misoppfatningene og problemene.

7.7 Forsøk B. Prototyping uten dokumentasjon og eksempelkode

Informatikkstudent på 29 år, uten erfaring med Visual Basic, men med generelt gode programmerings- og datakunnskaper.

Forsøkspersonen fikk *ikke* tilgang til eksempelkode og dokumentasjon og måtte støtte seg til metode/attributtlista som eVB tilbyr. Forsøksperson B fikk hjelp fra testleder når han sto fast, på samme måte som Forsøksperson A.

7.7.1 Observasjoner

Et nytt Visual Basic prosjekt ble åpnet og forsøkspersonen begynte å designe prototypen.

1. Få en enhet til å søke etter andre enheter
2. Stoppe dette søket
3. Legge enhet i combobox/listbox

-Forsøkspersonen var noe usikker på bruken av metoder, skrev for eksempel 'Blue.Startsok.attributt/metode'. Trodde at metodene var objekter.

-Han benyttet StopSok metoden inne i eventen _EnhetFunnet og trengte hjelp til å oppklare denne misforståelsen.

-Observerte at han ikke hadde klart for seg at han må hente en bestemt enhet fra et eller annet sted (combobox eller listbox i forbindelse med forsøkene).

4. Oppgi signalstyrke på tilknyttede enheter
-Forsøkspersonen trodde først at signalstyrken skulle sendes til den andre enheten.
Forsto hva metoden skulle gjøre, men trodde at parameteren 'enhet' fra Enhet_funnet lå i en global variabel som kunne benyttes i metodene. Han fikk derfor problemer med å angi riktig parameter til metoden.

5. Sende Fil

-Forsto at fil besto av filbane pluss aktuell fil. Hadde lært i oppgave 4 (oppgi signalstyrke) hvordan han sendte med riktig parameter 'enhet'.

6. Sende tekst mellom enheter
7. Motta tekst og eventuelt behandle den

-Forsøkspersonen brukte lang tid på å gå gjennom metodelista for å finne SendTekst. Trengte veiledning på hvilke metoder som skulle brukes.

Da han skulle bruke lese mottatt tekst på en enhet, ville han å benytte bla. metoder som OpprettSerieportforbindelse. Jeg ville ha ham til å bruke LesTekst, en metode som benytter OpprettSerieportforbindelse og hvor serieportbegrepet er abstrahert vekk.

-Han trodde at han måtte opprette forbindelse før han sendte tekst.

7.7.2 Kort intervju etter forsøket

I intervjuet etter forsøket poengterte Forsøksperson B følgende:

Han trodde at søket etter enheter måtte stoppes ved hjelp av StoppSok før han kunne gjøre andre operasjoner. Han trodde videre at startSok returnerte data i form av enheter. Sa at han ikke hadde noen forestilling om at en event _EnhetFunnet skulle benyttes i forbindelse med startSok.

Han sa at han heller ikke helt forsto hva en enhet var for noe.

På spørsmål om han syntes en synlig ActiveX-kontroll ville være å foretrekke framfor et usynlig grensesnitt, sa han at han foretrakk et usynlig grensesnitt slik som nåværende løsning, fordi han ville ha mest mulig kontroll over funksjonaliteten og designet i prototypen.
Han ble spurt om hva han mente om å få tilgang til underliggende metoder. Han mente dette var positivt, siden han liker å se hvordan det han benytter er bygd opp.

7.8 Forsøk C. Konseptuell modell som hjelpemiddel for å forstå ny teknologi

Informatikkstudent, 24 år uten erfaring med Visual Basic, men med grunnleggende programmeringskunnskaper og gode generelle datakunnskaper.

Forsøkspersonen fikk tilgang til eksempelkode og dokumentasjon.
Hun fikk i tillegg presentert en konseptuell modell for hvordan komponenten skulle benyttes, slik at hun for eksempel hadde en forestilling om hva for eksempel søk og events innebar.

Testleder forklarte forsøkspersonen hvilken teknologi hun skulle benytte i forsøket og hvordan denne teknologien virket. Viktige Bluetoothbegrep ble kort beskrevet og topologien ved Bluetooth-kommunikasjon mellom enheter ble utdypet. Hun fikk blant annet vite at en av enhetene starter dialogen og er master, mens andre enheter er slaver. Enhetene fungerer ellers som både klient og tjener – alt etter hvem som sender og mottar data.

7.8.1 Observasjoner

Et nytt Visual Basic prosjekt ble åpnet og forsøkspersonen begynte å designe prototypen.

1. Få en enhet til å søke etter andre enheter
2. Stoppe dette søket
3. Legge enhet i combobox/listbox

-Hun fant ut hvordan hun skulle søke og foreslo å lage en knapp for å søke. Hun spurte hvordan søket fungerte og hva søket innebar.

-Fant eventen _EnhetFunnet etter litt hjelp. Ble forklart eventens mekanisme (eventmodellen). Skjønte da at hun skulle fylle opp en combobox for å lagre funnede enheter.
Hun måtte ha hjelp til å legge inn parameteren 'enhet' i comboboxen fra eventen.

-Prototypen ble testet ut. Søket virket som det skulle og de funnede enhetene ble korrekt lagt i combobox'en.

4. Oppgi signalstyrke på tilknyttede enheter

-Forsøkspersonen foreslo at hun skulle lage en knapp for å løse problemet.

-Hun så ikke helt sammenhengen mellom lagring av enheter vha. en event og en combobox, og det virket ikke som om hun forsto at en enhet valgt herfra skulle brukes som parameter i OppgiSignalstyrke.
Hun hadde noe problem med å hente ut fra combobox, men skjønte prinsippet etter å ha forhørt seg med meg. Måtte ha hjelp til å få signalstyrken ut på skjermen ved hjelp av en Messagebox.

-Prototypen ble testet ut, og signalstyrkemetoden fungerte som den skulle.

5. Sende Fil

-Hun foreslo at hun skulle lage en knapp for å aktivere sending av fil.
Brukte dokumentasjon for å se hvilke parametre som trengtes i SendFil. Så på eksempelkode for legge inn parameteren fil. Visste at parameteren enhet kunne hentes fra combobox.

-Prototypen ble testet ut og filsendingen var vellykket.

6. Sende tekst mellom enheter
7. Motta tekst og eventuelt behandle den

-Så på eksempelkoden. Så at SendTekst har to parametre. Enhet gikk greit. Skjønte at teksten skulle sendes til en annen enhet.
Måtte ha hjelp til å finne en måte å skrive inn tekst som skal sendes til annen enhet. Laget Text-felt og så på metode/attributt-lista i VB at attributtet Text skulle brukes.

-Ble forklart at tekst skal mottas automatisk.
Forsøksleder foreslo at hun skulle bruke en 'timer' fra verktøylinja, som gikk og pollet for eksempel hvert annet sekund på det som kom inn til enhetens buffer. Fortsatt noe uklart hvordan eventer håndteres.

-Prototypen ble testet ut, og tekst ble sendt og mottatt gjentatte ganger.

7.8.2 Kort intervju etter forsøket

I intervjuet etter forsøket poengterte Forsøksperson C følgende:

Hun sa at hun skjønte 'rimelig greit' hva metodene skulle gjøre og hvordan hun benyttet dem. Var imidlertid usikker på eventer, siden de ikke er visuelle og det er vanskelig å forestille seg hvor dataene kommer fra.
Hun sa videre at Visual Basic-syntaksen ga endel vanskeligheter, siden språket var nytt for henne.
Designeditoren ligner mye på andre program hun hadde vært borti, så hvordan hun designet det visuelle brukergrensesnittet syntes hun det enkelt å sette seg inn i.

7.9 Forsøk D. Erfaren systemutvikler benytter komponenten

Stipendiat i informatikk, 29 år. Relativt erfaren Visual Basic-utvikler med meget gode generelle programmerings- og datakunnskaper.

Forsøksperson D fikk i likhet med Forsøksperson A og C benytte eksempelkode og dokumentasjon, og fikk også den konseptuelle anvendelsesmodellen muntlig presentert.

7.9.1 Observasjoner

Et nytt Visual Basic prosjekt ble åpnet og forsøkspersonen begynte å designe prototypen.

1. Få en enhet til å søke etter andre enheter
2. Stoppe dette søket
3. Legge enhet i combobox/listbox

-Laget en søkeknapp med søkefunksjon, men lurte på hva som skjedde teknisk "under panseret" under søkeprosessen. Han lurte på om søket pågikk hele tiden, eller om det stoppet etter en viss tid.
Han skjønte hvordan han benyttet eventer og brukte _EnhetFunnet eventen ved å dobbeltklikke på komponentkontroll i designeditor.

-La inn funnede enheter i en listbox han laget. Forsto at han skulle bruke parameteren 'enhet' inne eventmetoden. Han sa han savnet en event for å angi at søket er stoppet.

-Prototypen ble testet ut. Søket virket som det skulle.

4. Oppgi signalstyrke på tilknyttede enheter

-Han fikk til å vise signalstyrke, men han lurte på hva denne metoden returnerte og om hvorvidt det var en enhets signalstyrke på et gitt tidspunkt som vises. Ville se nærmere på hvordan metoden fungerte.

5. Sende Fil
-Han visste hvor enhet skulle hentes fra (listboxen) og forsto parameteroppsettet med fil og enhet.
-Han var klar over hva komponenten gjorde og hva som skjedde når fila ble sendt over. Han syntes imidlertid det var vanskelig å se om fila var mottatt eller ikke, og til hvilken katalog den var sendt.

6. Sende tekst mellom enheter

-Forsøkspersonen fant en måte å sende tekst til flere enheter i ei liste.

-Han fikk sendt over teksten tilsynelatende uten feil. Laget tekstfelt som teksten ble hentet ut fra.

7. Motta tekst og eventuelt behandle den

-Forsøkspersonen kikket på dokumentasjonen. Han klarte å bruke timer-polling for å motta tekst, men ville helst brukt en egen _tekstmottatt – event
Fikk teksten inn i en variabel og viste den fram på skjermen i et tekstfelt.
Han spurte om hvorfor vi ikke hadde en timer i komponenten så han slapp å bruke denne i eVB. Forsøkspersonen var opptatt av at bruker skal belastes minst mulig.

7.9.2 Kort intervju etter forsøket

I intervjuet etter forsøket poengterte Forsøksperson D følgende:

Han syntes at ActiveX-komponenten var enkel å bruke og hadde en lav terskel, men han foreslo at komponenten holdt styr på de enheter som ble funnet i en datastruktur og så eksponerte denne strukturen til eVB, for eksempel en enkel komma-separert tekststreng. En metode OppgiFunnedeEnheter kan brukes fra eVB til å hente inn søkeresultat. Dette mente forsøkspersonen burde vært som et supplement til nåværende løsning, ikke som erstatning. Viktig at man kan gjøre ting på ulike måter, det gir fleksibilitet.

Han syntes det var til hjelp å få klarhet i sammenhengen mellom de ulike begrep for å kunne se hvordan han skulle benytte komponenten. En del ting var dog fortsatt uklart. I stedet for 'enhet' mente forsøksperson D at man burde kalle attributtet 'enhetsnavn' i stedet, for å gjenspeile at det er snakk om navnet på enheten og ikke objektet i seg selv.

Forsøkspersonen mente at det er viktig å forklare brukere av komponenten (designere) hva metodene er basert på, hvilke mekanismer som skjer, både med hensyn til eventene og til det som metodene gjør og ikke gjør.
Den konseptuelle modellen kan altså for eksempel uttrykkes klarere i dokumentasjon for at brukere ser det store bildet/sammenhengen. Man må tilby å kunne vise litt under panseret i tillegg til en klar konseptuell modell.

Han mente at jeg hadde valgt riktig abstraksjon, og at synlige kontroller kunne tilbys i tillegg, men ikke i stedet for det ikke-visuelle grensesnittet.

7.10 Generelle resultater av forsøkene

Ingen av de tre første forsøkspersonene hadde erfaring med Visual Basic, men de hadde generelle kunnskaper i andre programmeringsspråk og litt kjennskap til Bluetooth-teknologien.
Alle forsøkspersonene fikk litt hjelp med Visual Basic-spesifikke problemstillinger og til å forstå hvordan de benyttet eventer. Ellers besto oppgaveløsningen for en stor del av selvstendig arbeid.
Forsøksperson A fikk tilgang til dokumentasjon og eksempelkode, mens forsøksperson B sto uten andre hjelpemidler enn metode/attributtlista. Dette for å få svar på om dokumentasjon og eksempelkode var til hjelp i oppgaveløsningen.

7.10.1 Å komme i gang

Av hensyn til den praktiske gjennomføringen av forsøkene hjalp forsøksleder til med å laste inn komponenten fra riktig katalog og legge komponenten over i design-editor. Det var begrenset tid til rådighet og de trengte å konsentrere seg om oppgaven og å få teknologien presentert. Bortsett fra denne starthjelpen måtte forsøkspersonene stort sett klare seg selv i oppgaveløsningen.

Når vi tar høyde for at de fikk noe hjelp til å komme i gang å bruke Visual Basic og å hente inn komponenten, kom forsøkspersonene seg overraskende raskt i gang med oppgavene. Terskelen for å komme i gang og bruke komponenten så ut til å oppleves som relativt lav og forsøkspersonene ga uttrykk for at komponenten var enkel å referere til i eVB. Metodene var enkle å finne og metodene hadde så få parametre at dette ikke utgjorde noe vesentlig hinder. Alle så ut til å forstå hvordan de kunne hente inn verdier til parametrene der hvor metodene trengte dette.
Eksempelkoden kan være nyttig for å takle problemer med at Visual Basic var nytt for forsøkspersonen og som et middel for å komme seg i gang. Både Forsøksperson A og C benyttet eksempelkoden til å se hvordan de kunne benytte metodene og sjekke syntaksen hvis de fikk kompileringsfeil.
Men eksempelkoden var også til hjelp for å komme i gang. Eksempelkoden konkretiserte trolig arbeidsoppgaven og gjorde det lettere å sette seg inn i tankegangen bak grensesnittet.

7.10.2 Nytten av en konseptuell modell

Forsøksperson B klarte seg relativt bra til tross for at han ikke hadde tilgang til dokumentasjon som metodebeskrivelser og eksempelkode. Dette kan skyldes at han fikk noe hjelp da han sto fast i Visual Basic-spesifikke spørsmål, men det kan også tyde på at grensesnittet framsto som selvforklarende og enkelt å benytte, selv uten dokumentasjon av grensesnittet. Forsøksperson B fikk imidlertid vesentlig større problemer enn Forsøksperson A (som hadde hjelpemidler) da han skulle løse de mer kompliserte oppgavene med å sende og motta tekst.
Forsøksperson B fikk bruksmodellen for grensesnittet forklart muntlig under selve oppgaveløsingen (som de andre forsøkspersonene), noe som for eksempel kan forklare at han klarte å benytte eventen _EnhetFunnet tilnærmet like bra som Forsøksperson A til å fylle opp lista over oppsøkte enheter.
Forsøksperson A så imidlertid ut til å ha fordel av å kunne se på dokumentasjonen for å få en nøyere beskrivelse av hvilke metoder som var riktig å bruke til å løse oppgaven, hva som var riktige parametre, hvilke returverdier metodene ga fra seg og hvordan metodene/attributtene/eventene forholdt seg til hverandre.

I tillegg ble enkelte punkter i dokumentasjonen revidert til forsøk C og D. Forsøksperson C og D fikk presentert en konseptuell anvendelsesmodell før de skulle ta fatt på å designe prototypen.
Jeg ville se hvorvidt den konseptuelle modellen var til hjelp i oppgaven med prototypen og om det var enklere å benytte komponenten når forsøkspersonene på forhånd hadde en forestilling om komponentens funksjonalitet og mekanismer.
Jeg anså at begrepene og aspektene knyttet til kommunikasjonsteknologien Bluetooth kunne presenteres i en konseptuell modell, men at forsøkspersonene også burde danne seg et bilde av hvordan interaksjonsmekanismene til ActiveX komponenten kunne benyttes i eVB– slik som bruken av events og notifikasjonsmeldinger mellom komponent og eVB.

Det kan stilles spørsmål ved om jeg klarte å formidle Bluetooths konseptuelle modell og komponentens anvendelsesmodell klart og grundig nok før deloppgavene tok til. Forsøksperson C så ut til å bli fortrolig med bruken av events først mot slutten av forsøket. Men det så ut til å være viktig at forsøkspersonene hadde oversikt under de enkelte deloppgavene og at de hadde klart for seg hvordan de kunne benytte komponenten til å programmere Bluetooth-funksjonalitet på PDA'ene. For å få denne oversikten under

prototypingen, så det ut som at det var en fordel å ha fått presentert den helhetlige anvendelsesmodellen på forhånd. Forsøkspersonene A, C og D ga også uttrykk for dette i intervjuene etter forsøkene.

Alle forsøkspersonene fikk en muntlig presentasjon av Bluetooth og Visual Basics konseptuelle modell. De to første forsøkspersonene fikk imidlertid ikke skriftlig presentert den konseptuelle anvendelsesmodellen til komponentgrensesnittet, slik det ble gjort i de to siste forsøkene. Forsøksperson A fikk utlevert dokumentasjon som kun inneholdt oversikt over metodene/eventene, mens forsøksperson B som nevnt ikke fikk bruke noen hjelpemidler. De måtte altså under selve prototypingen helt eller delvis stole på egne mentale forestillinger om Bluetooth og hvordan de anvendte komponenten –fra den muntlige presentasjonen eller fra tidligere erfaring.
Den konseptuelle "sender-mottaker–modellen" for bruk av Bluetooth er for eksempel grunnleggende for å forstå hvordan enhetene kan utveksle tekst. En mobil enhet starter opp en forbindelse med en annen enhet ved å sende noe til den. Enhetene kan så fortsette å sende tekst til hverandre helt til forbindelsen avbrytes. En enhet sender en tekst og en annen enhet mottar teksten. Forsøkspersonene hadde også vanskelig for å forstå hvor enhetene i eventen _EnhetFunnet kom fra. De så ikke ut til å ha klart for seg at enheten kom som en melding/event/interrupt fra den underliggende Bluetooth-radioen. Dette kan tyde på at det er fordelaktig å dokumentere og forklare interaksjonsmekanismene nøye.

7.10.3 Nytten av å tilby informasjon om det som er "under panseret"

Forsøksperson D representerte en brukergruppe som kjente utviklingsomgivelsen og var fortrolig med Bluetooth. Jeg forventet meg at han ville ha mindre problemer med å løse oppgavene enn de andre forsøkspersonene, noe jeg observerte i forsøket. Forsøksperson D trengte ikke hjelp til spesifikke VB-operasjoner, slik som tilordning av variabelverdier, datatyper, lovlige operasjoner mot datatyper, parametersetting og kontrollstrukturer.

Forsøksperson D var mer kritisk enn de andre forsøkspersonene med hensyn til hvordan komponenten var implementert og syntes det var utilfredsstillende hvis han ikke hadde full oversikt over hva for eksempel hva Bluetooth-søket innebar rent teknisk. Dette så ut til å hindre forsøkspersonen noe i forhold til å få gjennomført designet av prototypen i løpet av fastlagt tid (ca. en time). Forsøkspersonen heftet seg ved tekniske problemstillinger og savnet en nøyere beskrivelse av Bluetooth-funksjonaliteten.
Han spurte for eksempel hvor lenge søket varte, om man måtte søke opp funnede enheter på nytt hvis man skulle gjøre kommunisere med en enhet på nytt osv.

Forsøksperson D ønsket altså all den informasjonen om "how-it-works-kunnskap" som han kunne få, i motsetning til de andre forsøkspersonene, som hadde mer enn nok med å løse de de enklere oppgavene. Forsøksperson B var også interessert i "how-it-works", men brukte også mye tid på andre problemer, som for eksempel syntaks i Visual Basic.
Siden så mye ved Bluetooth er abstrahert i komponenten, så man spesielt i Forsøk D at det var viktig å presentere grundig både Bluetooth-funksjonaliteten som skulle prototypes og interaksjonsmekanismene i ActiveX mellom komponent og eVB-applikasjonen.

Ellers greide Forsøksperson D å løse oppgavene uten problemer. Det så derfor ut til at Forsøksperson D i likhet med Forsøksperson C opplevde grensesnittet som relativt klart og

konsistent. Dette ga han også uttrykk for under forsøket og i intervjuet. Skriftlig dokumentasjon av metodene og eventene sammen med en beskrivelse av anvendelsesmodellen bidro sannsynligvis til å gjøre forsøkspersonene sikrere ved løsing av oppgaven. De visste hva de hadde å forholde seg til i grensesnittet.

Forsøksperson D brukte mye tid på å prøve ut ulike skjermbilder ut fra de tekniske aspektene til Bluetooth. Han hadde blant annet ideer om hvordan han skulle utnytte informasjonen om enhetenes tjenester og laget listboxer med forskjellige enheter. Han eksperimenterte med hvordan brukergrensesnittet kunne endre seg etter hvilke data som ble mottatt via Bluetooth-radioen. Forsøksperson D hadde ideer om hvordan han kunne benytte komponenten til å skaffe kontekstinformasjon, noe som tyder på at komponenten imøtekom kravet om at det skulle gi ideer for anvendelse av teknologien (jvf. brukskvalitet 2 i problemstillingen).

7.10.4 Gjenbruk av programkode

Alle forsøkspersonene gjorde gjenbruk av programkode under forsøket. Særlig uthenting av verdier fra combobox/listbox og tilordning av verdier til parametre i metodene ble gjort ved klipping og liming og lignende. Hadde navngivningen og parametersettingen vært mindre konsistent, ville antakelig dette ikke vært så lett å gjennomføre og forsøkspersonene hadde fått en mindre oversiktlig programmeringskode. Måten grensesnittet var utformet på – med hensyn til abstraksjon, navngivning, parametre osv. – så ut til å passe godt til hvordan det er lagt opp til at utviklingsomgivelsen skulle benyttes. Forsøkspersonene syntes det var enkelt å gjøre metodekall i programkoden. De foretrakk å klippe og lime metodekall og andre kodebiter fra foregående oppgaver, da noen av metodene hadde likt parameteroppsett.
I Forsøk B ble for eksempel koden fra oppgave 5 (oppgi signalstyrke) gjenbrukt i oppgave 6 (sende fil).

7.10.5 Forslag til alternative måter å utforme grensesnittet på

Selv om Forsøksperson D brukte noe lenger tid enn fastsatt (en time og førti minutter), ble mye av denne tiden brukt til å bygge opp brukergrensensittet. Han laget en prototyp med et brukergrensesnitt som var mer ”sofistikert” enn det de andre forsøkspersonene greide. Han hadde dessuten ideer om ulike anvendelsesmuligheter for tekst-metodene som han viste i prototypen, noe som kan tyde på at komponenten har et potensial for forbedring.
Både Forsøksperson B og D påpekte at de savnet en lagringsstruktur for Bluetooth enheter i komponenten, men hadde ideer for hvordan denne svakheten kunne utbedres. De mente det ville være hensiktsmessig med flere alternative måter å hente ut enheter på, enten ved at komponenten tok seg av dette eller at man selv sørget for å legge enhetsnavn i listbox eller lignende, slik som i nåværende grensesnitt (se kapittel 8 om forbedringer av grensesnittet).

Del III

Refleksjoner

8. Forslag til forbedringer

Jeg vil i dette kapitlet se nærmere på mulighetene for forbedringer av grensesnittet. Erfaringene fra utviklingen av grensesnittet mot Bluetooth og de problemstillingene jeg sto overfor under denne prosessen har fått meg til å tenke over hvordan grensesnittet kan gjøres enda bedre. Forsøkene ga verdifull informasjon om hvordan designet av grensesnittet kunne tilpasses brukeres behov og hvilke nye funksjonaliteter/muligheter som kunne tilbys i en videreutvikling av Bluetooth-grensesnittet.

8.1 Enhetssøk og løsning for lagring og uthenting av enheter

En problemstilling knyttet til søket var hvor langt tid dette søket skal ta. Et forslag til forbedring kan være at lengden på søkeperioden angis som en valgfri parameter. På denne måten kan designer velge om han vil søke i en gitt tidsperiode eller om søket skal pågå til prototypen avsluttes eller søket stoppes ved hjelp av metoden StoppSok.

Søkeperioden kan ha innvirkning på hvilken tilbakemelding designer skal gi til bruker. Designer bør derfor legge inn en fornuftig søkeperiode. Hva som er en fornuftig søkeperiode kan avhenge av sammenhengen, men det kan være nyttig å gi designer flere alternativer, slik at han har flere valgmulighetene og føler at han har mer kontroll. Samtidig er det en fare for at flere alternative måter å løse oppgavene på kan virke forvirrende for designeren.

Slik grensesnittet er utformet har designeren selv ansvaret for å lagre enheter som blir funnet. Designeren sender filer, tekst/meldinger eller gjør andre operasjoner mot den eller de enhetene han velger i den lista han har laget (for eksempel en listbox).
Det kan imidlertid være hensiktsmessig å tilby designeren flere måter å angi hvilke enheter det skal gjøres operasjoner mot. Flere alternativer for lagring av enheter kan gi større fleksibilitet.
For at designer selv skal slippe å måtte lagre enheter som skal brukes, kan man utforme komponenten slik at alle funnede enheter legges i et array internt i komponenten (utviklers kommentar: dette arrayet vil gjenspeile alle enhetene i DeviceList).
Designer vil da ikke være avhengig av å lagre enhetene i et array, listbox eller lignende i eVB for å kunne gjøre operasjoner mot gitte enheter, og får dermed flere valg. Designer kan for eksempel basere seg på at den aktuelle enheten(e) skrives inn i et tekstfelt i stedet for at man må velge enhet fra en listbox eller lignende.
Alternativt kan enhetsarrayet i komponenten bare lagre resultatet av det siste søket. Dette enhetsarrayet blir mindre persistent, men vil ha en større andel relevante enheter (innen rekkevidde), siden enhetene kan være innen rekkevidde der og da, for så å forsvinne. Enheter som er søkt opp for 10 minutter siden kan ha kommet ut av rekkevidde, mens de som ble funnet i det seneste søket sannsynligvis fortsatt er tilgjengelige.

Figur 8.1 Et enhetsarray i komponenten

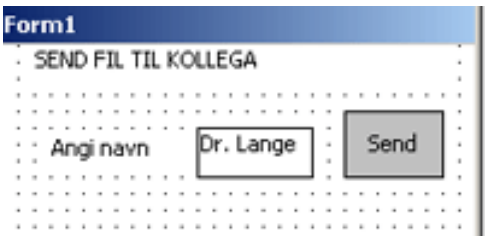
Man kan godt tilby begge typer enhetsarrays, slik at designer kan velge det enhetsarrayet som passer best, men hvis arrayet skal brukes internt i en annen metode i komponenten, må jeg som utvikler av komponenten på forhånd velge om jeg skal benytte et persistent array i implementasjonen. Flere alternative arrays kan imidlertid eksponeres til designer, hvis designer skal gjøre uavhengige operasjoner mot et array (og ikke igjennom en annen metode som abstraherer arrayoppslagene).

8.2 Sending av filer hvor uthenting og søk er abstrahert vekk

Det kan være aktuelt å innlemme metoden StartSok (og evntl. StoppSok) i SendFil. Søkemetodene vil fortsatt kunne brukes uavhengig, men kalles i tillegg i metoden SendFil. På denne måten kunne jeg ha abstraherert vekk problemene knyttet til at man må søke etter enhetene før man kan utføre operasjoner mot dem.

Slik det er nå tar metoden SendFil navnet til enheten som input-parameter og sender den angitte fila til denne enheten. En forbedring av denne metoden kan være at designer ikke trenger å ha søkt opp den enheten det skal sendes ei fil til. Metoden sjekker automatisk enheten opp mot ei liste over funnede enheter i komponenten. Denne lista kan inneholde referanse til alle enheter som er funnet i prototypens levetid (søkehistorikk) – eller alternativt; enheter som er funnet i siste søk.

Komponenten er implementert slik at enheten det opereres mot må ligge i komponentens enhetsliste (DeviceList). Enhetene lagres i denne lista når de blir funnet i et søk.
Hvis enheten som skrives inn i tekstfeltet blir funnet i lista, forsøker metoden SendFil å sende fila til den aktuelle enheten. Hvis den ikke finnes i lista, kan metoden søke etter enheten på nytt.



Figur 8.2 Design av filsendingsmeny med tekstfelt

Hvis det må iverksettes et søk etter den aktuelle enheten, vil designeren (og brukeren) merke en tidsforsinkelse på noen sekunder. Det kan derfor være til god hjelp for designer at ActiveX-komponenten har en måte å angi hvis enheten ikke finnes i enhetsarrayet og må søkes opp. Slik kan designer gi en fornuftig melding til brukeren om at søket kan ta litt tid. Til dette går det an å benytte seg av en notifikasjonsmelding - en av interaksjonsmekanismene til ActiveX.
Om enheten som fila skal sendes til ikke finnes i arrayet, må det foretas et søk, forandres et attributt (variabel) innad i sendemetoden SendFil. Dette invokerer en notikasjon - som for eksempel OnSokestatusChanged. I denne notifikasjons-metoden legges en event som heter _SendFilSok. Hvis SendFil må søke etter den aktuelle enheten, fyres denne eventen av og designer kan dermed legge inn en MsgBox eller lignende når eventen inntrer.

8.3 Sending av tekst

I nåværende design gir ikke SendTekst en melding om forsendelsen virkelig kom fram til mottaker, slik som SendFil gjør. I forbindelse med denne metoden kan designer benytte eventen _SendFile_Complete for å sjekke om mottakerenheten mottok fila og gi fornuftige brukermeldinger på om fila kom fram til mottaker, for eksempel ved hjelp av en messagebox. En slik event ville imidlertid også vært nyttig i SendTekst-metoden.

Det er dessuten mulig å innlemme StartSok i SendTekst, på samme måte som SendFil, slik at den angitte enheten søkes på automatisk hvis den ikke blir funnet i enhetsarrayet.

De andre metodene som benytter parameteret enhetsnavnet kan på tilsvarende vis foreta et enhetssøk hvis enheten ikke allerede befinner seg i enhetslista /arrayet.

8.4 Generisk overføring av data mellom enheter

En naturlig videreutvikling av tekstoverføringsmetodene kan være å tilby generiske metoder for utveksling av binærdata. Disse metodene for sending og mottakelse av data kan danne grunnlaget for høyere nivås regler/protokoller for utveksling av data.
For eksempel kan det sendes med metainformasjon i ”datapakkene” som inneholder informasjon som enhetsnavn, tjenestetyper, dato, sted, eller annen kontekstinformasjon.
Man kan se for seg at overføringsmetodene kan nyttes i en rekke ulike sammenhenger. Det er like mye kreativiteten som teknologien som begrenser mulighetene.

Generiske metoder for overføring av data kan gjerne danne grunnlaget for protokoller for overføring av lyd og bilde, men å lage slike protokoller er komplisert og tidkrevende, og Bluetooth-stackene (på lavere nivå) blir stadig utvidet til å takle multimediaelmenter på en standardisert måte, så det er tvilsomt om det svarer seg å videreutvikle datautvekslingsmetodene til akkurat dette formålet.

8.5 Tjenester

Jeg kunne gjort mer ut av mulighetene Bluetooth i forhold til å behandle tjenester på ulike enheter. Tjenester er nært knyttet til begrepene enhetsklasser og profiler (som er abstrahert vekk i grensesnittet).
Slik det er nå, kan SjekkTjeneste benyttes til å finne ut om en enhet støtter en viss tjeneste, men da må designeren vite tjenesten på forhånd og sende inn den aktuelle tjensesten som parameter. SjekkTjeneste burde følgelig ikke kreve noen inputparametre, men returnere den aktuelle enhetens tjenester.

Man kan se for seg at de forbedrede SjekkTjeneste-metodene kunne kalles av andre metoder som trenger informasjonen om tjenester som enhetene støtter, slik at de kan vite hvilken enhet det er og ut fra dette velge om en bestemt operasjon skal utføres eller ikke.
Det ville for eksempel vært hensiktsmessig å tilby en metode som listet opp alle tjenestene som var tilgjengelige på en bestemt enhet. På denne måten kunne man finne ut om enheten tilhørte en ønsket profil. Man kan da oppnå å få en viss profilfunksjonalitet, selv om profil - og enhetsklassebegrepene ikke er representert i grensesnittet.

8.6 Øvrige muligheter til forbedring

Det er mulig å lage en videreutvikling av ActiveX-komponenten som bruker mindre minne og er mer kompatibel med kjøreegenskapene (run-time) til eVB. Ytelse er viktig fordi komponenten kjøres på PDA'er som har begrensninger, spesielt med hensyn til minne. Prototypen bør kunne kjøre Bluetooth-funksjonaliteten uten forsinkelser og andre kjøreproblemer hvis en designer skal ha full nytte av den.

Da jeg utviklet grensesnittet mot Bluetooth, var tanken å tilby et visuelt grensesnitt. Grensesnittet ble imidlertid utformet som en usynlig ActiveX-kontroll. Designer kan da bestemme selv hvordan brukergrensesnittet skal se ut.
Forsøkene viste at den usynlige ActiveX-kontrollen er så enkel å bruke at det ikke nødvendigvis er noen ulempe at den ikke er representert ved et synlig grensesnitt. I eVB er det lett å få opp et brukergrensesnitt og relativt enkelt å programmere funksjonalitet til de visuelle brukergrensesnittobjektene. Forsøksperson D poengterte i intervjuet etter forsøket at han foretrakk et visuelt grensesnitt som supplement og ikke som en erstatning av det ikke-visuelle grensesnittet (se kap. 7.9.2).
Samtidig kan det være gunstig å gi designeren muligheten til for eksempel å få opp et ferdig og visuelt datautvekslingsobjekt der han slipper å tenke på hvordan legger funksjonalitet til en samling med 'widgets'. Dette kan han spare noe tid på og det synlige grensesnittet kan i det minste fungere som et nyttig eksempel.
Det er altså vektige argumenter både for og imot å tilby ferdigordnede visuelle grensesnitt med integrert design og funksjonalitet.

Ut fra tilbakemelding fra forsøkspersonene mener jeg at jeg kunne ha tilbudt et ferdigordnet synlig grensesnitt som supplement til den eksisterende løsningen. Det ville blant annet være interessant å tilby et synlig grensesnitt for å utveksle tekst/filer, der det både var mulighet for input og output og nødvendige metodekall for å søke og å sende og motta tekst/filer.

Ferdigordnede grensesnitt (synlige kontroller) ville dannet et abstraksjonslag over de metodene/eventene/etc. som det nåværende grensesnittet består av.

Motsatt kan man se for seg at det er mulig å tilby metoder på et lavere abstraksjonsnivå, for eksempel for at designeren selv kan behandle inngående og utgående Comporter og serieport-forbindelser.
En forbedring av Bluetooth-grensesnittet kan følgelig være å dele tydeligere inn i ulike abstraksjonslag, noe som kan øke mulighetene for brukertilpasning, gi større fleksibilitet og få designeren til å føle at han har bedre kontroll. Samtidig er en fare for at dette kan øke kompleksiteten og dermed skape forvirring hos designeren.

Bluetooth-grensesnittet støtter ikke oppretting av Piconet, men det kunne det vært interessant å tilby. Det stiller imidlertid et visst krav til designeren å skulle forholde seg til den kompleksiteten et Piconet medfører. Det stiller høye krav til utvikleren å skulle tilby en slik funksjonalitet, da det ikke er lett å forenkle Piconet og broadcasting mellom opptil 8 enheter slik at det enkelt kan prototypes og designer samtidig beholder gode muligheter til programmering og tilpasning.

Bluetooth er en ung teknologi og det dukker opp nye anvendelsesområder etter hvert som teknologien modnes. Bluetooth-komponenten kan derfor videreutvikles slik at den tilbyr funksjonalitet innen de nye anvendelsesområdene. Dette trenger ikke innebære så mye mer innsats, siden komponenten, rammeverket rundt komponenten og utformingen av de konseptuelle sidene ved komponenten allerede er utarbeidet.

8.7 Anvendelsesmodellen til grensesnittet

Avslutningsvis vil jeg si at en videreutvikling av grensesnittet vil kunne medføre at anvendelsesmodellen blir noe endret.

For eksempel vil flere alternativer for lagring av søk, innbaking av søkefunksjon i andre metoder eller utvikling av et visuelt grensesnitt gjøre at designeren vil måtte forholde seg annerledes til grensesnittet.
Tekstsending vil i en videreutvikling ikke nødvendigvis skje etter at designeren har gjort et søk og valgt enhet fra ei liste. Designeren vil måtte forholde seg til at det er flere alternative måter å kommunisere med andre enheter på. Enheter kan søkes opp og utveksle informasjon med separat, eller alt dette gjøres implisitt i komponentens metoder.
Det er derfor viktig å være oppmerksom på at designendringer i en videreutvikling av komponenten kan medføre at man må tilpasse den konseptuelle anvendelsesmodellen til grensesnittet.

9. Diskusjon

Problemstillingen for denne hovedfagsoppgaven var *hvordan man best kunne legge til rette for prototyping av ny teknologi i en utviklingsomgivelse* (se kap.2 for problemstilling).

For å løse denne problemstillingen har jeg først sett på hvordan man kan abstrahere den nye teknologiens begrep, slik at de framstår som forståelige for designeren som benytter en utviklingsomgivelse.
Jeg har deretter spurt meg hvordan de abstraherte begrepene kan presenteres på en slik måte at designer skjønner sammenhengen og får ideer om hvordan teknologien kan anvendes uten å stå fast i teknologisk kompleksitet.
Et tredje delproblem har å gjøre med selve grunnlaget for å lage et forenklet grensesnitt mot ny teknologien; hvorvidt det er hensiktsmessig å tilby designere muligheten til lage implementasjonsprototyper som demonstrerer teknologiens funksjonelle sider, og ikke kun designe look-and-feel - eller rolleprototyper.

Jeg vil påpeke at løsningene på delproblemene ikke er noen "fasitsvar" på hvordan man kan løse slike problem. Forsøkene gir ikke grunnlag for å dra bestemte slutninger, men må ses på som min betraktning av hvordan Bluetooth-komponenten ble opplevd av forsøkspersonene (se kap. 3.4 og 11.2).

Hovedfagsoppgaven kan gi innspill til hva det er viktig å tenke på når man legger til rette for prototyping av ny teknologi for designere.

9.1 Delproblem 1

"Hvordan kan man abstrahere de viktigste egenskapene ved en ny teknologi slik at man reduserer teknologiens kompleksitet?"

9.1.1 Abstraksjon av teknologi

Å sette seg inn i en kompleks teknologi som man har lite kjennskap til fra før kan være vanskelig. Hvis folk med begrenset kunnskap om teknologien skal kunne lage fullt funksjonelle prototyper der teknologien inngår er det nødvendig å fjerne noe av kompleksiteten så teknologien blir mer tilgjengelig.
Å forenkle noe ved å trekke ut det vesentlige og "gjemme" det uvesentlige kalles for å abstrahere. Ved å abstrahere kan man fjerne kompleksitet. Resultatet blir en abstraksjon som ligger på et høyere nivå enn det som ble abstrahert og som forhåpentligvis er lettere å forstå. Abstraksjon kan brukes for å tilpasse teknologiske begreper til praktisk problemløsning.

For å kunne abstrahere teknologiske begreper må man som regel sette seg grundig inn i teknologien.
I kapittel 6 viste jeg hvordan jeg tok tak i teknologiens kompleksitet og forsøkte å dra ut de vesentlige begrepene. Disse begrepene dannet grunnlaget for en konseptuell modell av teknologien.

9.1.2 Bruk av et grensesnitt for å abstrahere

Løsningen var å tilby abstraksjonen i et grensesnitt som kunne benyttes fra et høynivås utviklingsomgivelse. For å gi høy brukskvalitet måtte det være enkelt å benytte grensesnittet. Et brukskrav var at grensesnittet måtte være velegnet for en designer med begrensede programmeringskunnskaper til å prototype brukergrensesnitt og funksjonalitet. Det var videre et krav at grensesnittet/komponenten måtte kunne brukes til å demonstrere aspekter ved teknologien slik at designeren fikk ideer om hvordan teknologien kunne anvendes.
Da jeg implementerte grensesnittet/komponenten var meningen å redusere den tekniske kompleksiteten ved å presentere en abstraksjon som egnet seg bedre til å løse oppgaven med å lage hurtige prototyper. Kompleksiteten ble gjemt i implementasjonen, mens abstraksjonen ble presentert i grensesnittet.

9.1.3 Valg av abstraksjonsnivå

Det er viktig at det velges et passende abstraksjonsnivå som hverken gjør det vanskelig å benytte grensesnittet eller begrenser designrommet.
Et lavt abstraksjonsnivå kan gjøre det for komplisert for designeren å konsentrere seg om problemløsningen. Man kan risikere at designer henger seg opp i syntaktiske problemer eller generelt har vanskelig for å få finne ut hvordan metodene/eventene i grensesnittet benyttes for å få opp en prototyp som fungerer.
Et høyt abstraksjonsnivå kan også ha ulemper. Ved abstraksjon drar man ut det antatt essensielle ved teknologien. Dermed mister man nødvendigvis de teknologiske detaljene som ikke blir med i abstraksjonen.

Da abstraksjonene av en teknologi kun består av utvalgte trekk ved teknologien, er det gjerne viktig å erstatte de ”utelatte” teknologiske detaljene med en helhetlig forståelse av som gir en god oversikt over teknologiens aspekter. Designeren trenger nok informasjon om teknologien til å forstå hvordan teknologien kan anvendes. For å forstå anvendelsesmulighetene kan det også være nyttig å ha kunnskap om hvordan teknologien fungerer (how-it-works). Designeren trenger imidlertid ikke å forstå ”alt”, men abstraksjonen som er utført vil være med og avgjøre hvilken strukturell modell av teknologien som designeren skal forholde seg til.

I forbindelse med abstraksjonene jeg gjorde i Bluetooth-komponenten, var jeg opptatt av å gi designere et riktig bilde av teknologien. Abstraksjonene utgjorde et begrepsapparat som var utgangspunktet for den konseptuelle modellen av teknologien, noe som fikk direkte innvirkning på hvordan grensesnittet ble utformet.
Bluetooth er en teknologi som gir kommunikasjon mellom flere ulike enheter som tilbyr forskjellige tjenester. Jeg måtte derfor velge ut de aspektene ved teknologien som jeg anså som mest interessante for designere.

9.1.4 Abstraksjon ved hjelp av ActiveX

ActiveX er en komponent-teknologi som egner seg til å abstrahere teknologiske begrep, ved at man kan eksponere funksjonalitet til en høynivås utviklingsomgivelse i et forenklet grensesnitt.
Jeg benyttet en ActiveX-komponent til å utvikle Bluetooth-grensesnittet. For at ActiveX-komponenten skulle tilby designeren å prototype Bluetooth-funksjonaliteten slik at han fikk

demonstrert teknologiens tekniske og konseptuelle aspekter, måtte komponenten ha mekanismer for å behandle Bluetooth-funksjonaliteten.
For eksempel så abstraherer komponenten mottakelse av tekst via radioen og til inngående serieport-buffer. I stedet for å la designer forholde seg til inngående serieportbuffer, tilbys enten en abstraksjon som heter LesTekst eller abstraksjonen _TekstMottatt. Sistnevnte er en event som sjekker om tekst er kommet inn i bufferet og fyres av når tekst blir mottatt (se kap. 6.10).

Mulighetene til ActiveX til å eksponere eventer og metoder som reagerer på signal fra underliggende Bluetooth-funksjonalitet gjør at man kan utvikle kraftfulle abstraksjoner som enkelt kan brukes i den aktuelle utviklingsomgivelsen til å demonstrere teknologien.
I ActiveX-komponenten besto abstraksjonene av metoder som ga designeren gode muligheter til å lage prototyper som øyeblikkelig reagerer på radiosignal fra Bluetooth. Man oppnådde dermed en slags "real-time"-prototyp som ga en realistisk demonstrasjon av teknologien.

For at designeren skulle få nytte av abstraksjonen til å demonstrere teknologien, viste det seg imidlertid at det var viktig at abstraksjonen i ActiveX-grensesnittet samsvarte med nivået som designers problemløsning befant seg på. ActiveX-eventene var abstraksjoner av interrupts på maskinvare-nivå og tilsvarte message-handlers og callback-funksjoner i WinCE-API'en. Eventene var likevel like anvendbare og egnet seg godt til å håndtere meldinger fra underliggende lag.
Eventer og andre interaksjonsmekanismer mellom eVB og komponenten sammen med metodenavn, eventnavn og parameternavn på "godt norsk" så ut til å gjøre grensesnittet såpass brukervennlig at forsøkspersonene i forsøkene hadde relativt lett for å forstå hva som lå i de ulike begrepene i grensesnittet.
Forsøkspersonene hadde imidlertid forskjellige måter å løse oppgavene på, så behovet for mer spesialtilpassede løsninger og alternative måter å utføre oppgaver så ut til å være savnet (se kap. 8 om forbedringer).

Da ActiveX-komponenten ble utviklet var det en utfordring å vite på hvilket nivå jeg skulle legge abstraksjonen. Det var en avveining mellom å gjøre det enkelt og å gi designer handlingsrom, muligheter og innsikt i teknologien.

9.1.5 Mine erfaringer med hva som var viktig for å abstrahere teknologien

Mine erfaringer med abstraksjon av teknologisk kompleksitet i Bluetooth-grensesnittet har gitt meg et inntrykk av hva det er viktig å være klar over når man abstraherer, slik at man imøtekommer kravene om at det skal være enkelt å prototype brukergrensesnitt og funksjonalitet og at designer får ideer om ulike anvendelsesmuligheter.
De viktigste oppdagelsene jeg har gjort i forhold til brukskravene og Delproblem 1 var som følger :

1. Det var viktig å tilpasse abstraksjonsnivået til målgruppens problemløsning.
2. Det var stor forskjell fra person til person hvordan grensesnittet blir brukt, avhengig av faktorer som blant annet kunnskaper, målsetning og brukssituasjon. Noen ønsket brukerkontroll og innsikt i teknologien, mens andre ønsket mer automatikk og skjuling av kompleksiteter og mekanismer i komponenten.
3. Det var nyttig å tilby målgruppen flere alternative måter å utføre oppgaver på.
4. Det var nyttig å tilby flere abstraksjonsnivå i grensesnittet

Forsøkene antydet at disse punktene var viktige for å lage abstraksjoner av teknologien som imøtekom brukskravene til designere.

Forsøkene bekreftet at det var viktig at Bluetooth-grensesnittet var relativt godt tilpasset forsøkspersonenes måte å tenke på ved oppgaveløsingen. I intervjuene etter forsøkene ga alle forsøkspersonene uttrykk for at de syntes at metodene/eventene støttet opp om måten de tenkte på da de løste de forskjellige oppgavene. Metodenavnene antydet for eksempel hvilken funksjonalitet de utførte.

Flere av forsøkspersonene påpekte imidlertid at de ønsket seg alternative løsninger for lagring av enheter og ville ha mer tilpassede søkemuligheter. For eksempel uttrykte forsøksperson A og D ønske om en søkefunksjon der søkets varighet ble angitt som parameter. Dette tolket jeg som at det ville være nyttig med flere alternative måter å utføre oppgavene på.
Særlig Forsøksperson D var opptatt av å vite mer om hvordan teknologien fungerte enn det som ble presentert muntlig og i dokumentasjonen. For å dekke de ulike behovene, foreslår jeg i kapittel 8 (om forbedringer) at grensesnittet kan tilby en klarere inndeling i ulike abstraksjonsnivå enn det nåværende grensesnittet tilbyr, noe som også antydes i retningslinjene i kapittel 10 og i punkt 4 overfor i dette kapitlet.

Abstraksjonene ble gjort ut fra en vurdering av *hvem* som skulle benytte grensesnittet, hvilke *hensikter* jeg trodde de hadde med å benytte komponenten, *kravene* målgruppen ville stille til brukskvalitet og i hvilken *sammenheng* prototypene skulle inngå.

9.2 Delproblem 2

"Hvordan kan man presentere en ny teknologi for designere?"

9.2.1 Teknologiens konseptuelle modell

Når man utvikler et grensesnitt mot ny teknologi er det gjerne om å gjøre å forenkle teknologiens kompleksitet slik at den blir tilgjengelig for designeren.

Abstraksjonene danner utgangspunktet for den konseptuelle modellen til teknologien ved at de definerer det nødvendige begrepsapparatet for å forstå teknologien. Det er viktig at abstraksjonen presenteres på en slik måte at designere kan danne seg en overordnet forestilling om aspektene ved teknologien og hvordan teknologien kan benyttes.

Som det ble poengtert i kapittel 4.4 er hovedutfordringen når man skal lage en konseptuell modell å sørge for en konsistent begrepsdanning – en designmodell, og å legge til rette for at de som skal benytte den konseptuelle modellen danner seg en mental modell som samsvarer med designmodellen. Det er da viktig at begreper og metaforer i teknologi-grensesnittet er tydelige og gir mening, slik at designeren får et inntrykk av grensesnittet som samsvarer med det bildet man ønsker å uttrykke.
I tillegg til begrepsdannelse bør den konseptuelle modellen i grensesnittet gi et presist bilde av hvordan de ulike begrepene og metaforene hører sammen og hvordan de kan uttrykkes i

brukergrensesnittet. Designer bør altså få ideer om hvordan abstraksjonene av de teknologiske begrepene kan anvendes i prototypen både til å lage funksjonalitet og brukergrensesnitt.

Den konseptuelle modellen jeg ville formidle av Bluetooth skulle gi designerne en overordnet forståelse av aspektene ved teknologien. Jeg presenterte abstraksjoner av teknologien i grensesnittet - som for eksempel SendTekst og OppgiMittNavn. Disse abstraksjonene håpet jeg at det gikk greit for brukere av grensesnittet å forstå. Det var om å gjøre å presentere en samling av begreper som gjorde det mest mulig intuitivt og interessant for designere/brukere å lage funksjonelle prototyper.
I tillegg til at navnene til metoder, eventer, parametre osv. i grensesnittet formidlet den konseptuelle modellen og at dokumentasjonen til Bluetooth-komponenten forklarte den overordnede teknologiske modellen, ble teknologiens konseptuelle modell overlevert muntlig til forsøkspersonene under testene av komponenten/grensesnittet (unntatt i forsøk B). Jeg ville se hvilken nytte designere kan ha av å få presentert en overordnet forestilling av Bluetooths aspekter. Det så for eksempel ut til at designere kunne ha nytte av å ha en overordnet forståelse av utveksling av informasjonsobjekter mellom enheter. Det virket nemlig som om forsøkspersonene trengte å forstå den grunnleggende ideen med å sende og motta data mellom to enheter hvor det ikke var noe definert klient-tjener-forhold. Det så ut til å være en fordel å skjønne på forhånd hvilken topologisk/overordnet tankegang som lå bak programmering/ prototyping av tekstutvekslingsmetodene SendTekst og LesTekst i eVB. De måtte få presentert tankegangen, fordi det ellers kunne være forvirrende å programmere enheter som kommuniserte ”rundt i rommet”.

9.2.2 Utviklingsomgivelsens konseptuelle modell

Designer benytter grensesnittet mot teknologien til å lage prototyper i en bestemt utviklingsomgivelse. Denne utviklingsomgivelsen vil høyst sannsynlig ha en nøye utformet konseptuell modell som det er viktig å ta hensyn til. Meningen med den konseptuelle modellen er at den skal lette utviklingsarbeidet. Det er denne konseptuelle modellen designer forholder seg til når han lager skjermbilder og programmerer funksjonalitet.
Utviklingsomgivelser som Flash og Visual Basic har en design-editor som benytter kjente metaforer og begrep som lister, bokser, knapper osv., og tar i bruk prinsippet om direkte manipulasjon av grensesnittobjekter.
Programmeringsspråkene til utviklingsomgivelsene jeg nevnte i kapittel 5.6 kjenntegnes gjerne av at de er objekt – eller hendelsesorienterte, eller i det minste har et relativt høyt abstraksjonsnivå.
Design -og programmeringsmulighetene til utviklingsomgivelser som Flash og Visual Basic gjør at de kan benyttes til prototyping.

For å få tilgang på funksjonalitet i en utviklingsomgivelse som (Embedded) Visual Basic er det lagt opp til å benytte ActiveX-komponenter med veldefinerte grensesnitt.
Når en designer skal benytte en slik komponent er det naturlig for ham å prøve å forestille seg *hvordan* denne komponenten kan brukes. Det er lagt opp til i rammeverket til Visual Basic at komponenter kan hentes inn og benyttes ved å legge dem i design-editoren og referere til dem i programmeringskoden.
Visual Basic gir bestemte muligheter og begrensninger for hvordan designer kan programmere komponentens funksjonalitet, da Visual Basic medfører en viss programmeringssyntaks, gir tilgang til bestemte API’er/ biblioteker og har støtte for visse typer ”eksternt linkede” komponenter som ActiveX.

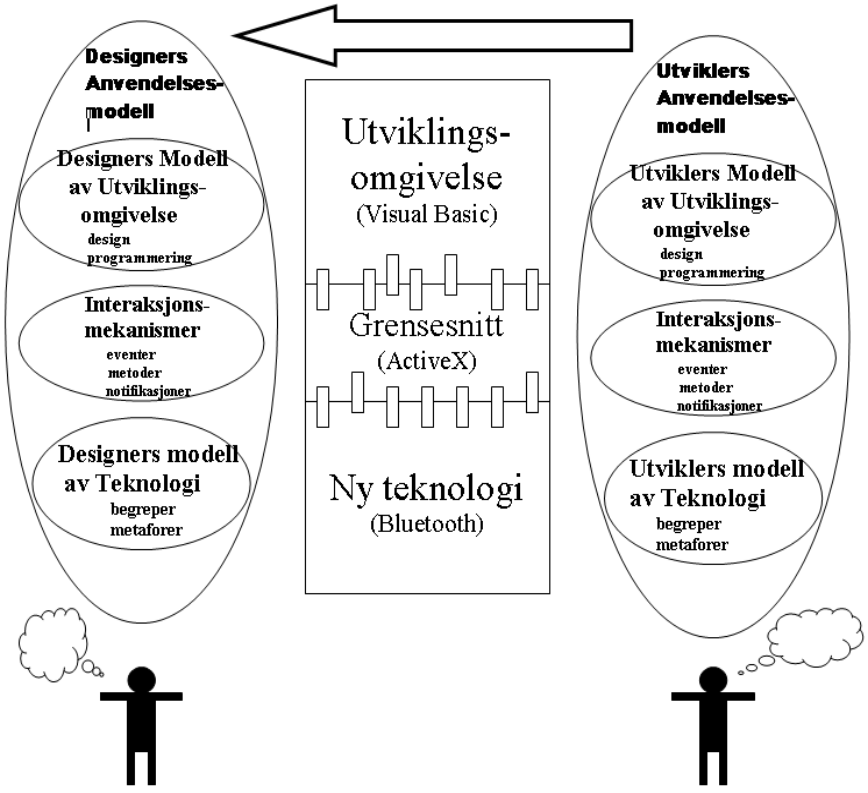
Det er dette som danner utgangspunktet for de mulighetene designer har til å uttrykke seg.

Den konseptuelle modellen til Visual Basic tar utgangspunkt i objekter med bestemte egenskaper og oppførsel. Utviklingsomgivelsen representerer en objekt – og hendelses-orientert tankegang, hvor "programmeringsverdenen" består av en samling objekter som det er knyttet handlinger til.
I denne sammenheng forholder designer/utvikler seg til komponenter på samme måte som til andre objekter.

9.2.3 Teknologiens anvendelsesmodell

Jeg har laget et grensesnitt som gjør Bluetooth tilgjengelig i Embedded Visual Basic (eVB). For at det skulle være enklest mulig for designer å forholde seg til grensesnittet som jeg laget mot Bluetooth, forsøkte jeg å skape en anvendelsesmodell for grensesnittet. Denne modellen måtte ta hensyn til den konseptuelle modellen til teknologien og samtidig til utviklingsomgivelsens konseptuelle modell.
Jeg ville jo formidle det bilde av teknologien som jeg mente det var mest nyttig og interessant for designer å ha, og måtte ta hensyn til at komponenten skulle benyttes i eVB.
Jeg forsøkte derfor å få designer til å forestille seg prototypingen av teknologien som en utvidelse av de mulighetene eVB ga. Komponenten skulle kunne programmeres som om det var et ordinært objekt med egenskaper og oppførsel.

Anvendelsesmodellen til Bluetooth-grensesnittet kan ses på som den helhetlige modellen jeg ønsket at designer skulle ha av å prototype Bluetooth i eVB. Det var anvendelsesmodellen som ga designer illusjonen av å benytte et utvidet "objekt" i eVB.



Figur 9.1 Anvendelsesmodell for teknologi-grensesnittet

Anvendelsesmodellen er imidlertid noe mer enn summen av teknologiens- og utviklingsomgivelsens konseptuelle modeller.
I figur 9.1 er anvendelsesmodellen illustrert. Utvikler (til høyre i figuren) prøver å formidle til designer (til venstre i figuren) en modell av hvordan ActiveX-komponenten kan anvendes. Anvendelsesmodellen er den totale modellen som rommer en modell av både utviklingsomgivelse, teknologi og interaksjonsmekanismer.

Det er lagt opp i eVB til at man kan benytte ActiveX-komponenter i design/utvikling av applikasjoner. Interaksjonsmekanismene som ActiveX tilbyr – slik som eventer og notifikasjonsmeldinger – tilbys også i selve utviklingsomgivelsene, men måten mekanismene benyttes på i ActiveX er særegen. I eVB kan man benytte objekter med eventer, slik som timere, og mange av kontrollene/komponentene som benyttes i eVB er egne ActiveX-komponenter.
Selv om ActiveX som grensesnitt-teknologi ofte benyttes inne i applikasjoner og utviklingsomgivelsen, har ActiveX sin egen "modell".
Når man utvikler et ActiveX-grensesnitt, kan man utnytte mulighetene som for eksempel ligger i notifikasjonsmekanismen til å tilby "real-time" respons på sending og mottak av Bluetooth-radiosignal. Disse mekanismene kan så designere dra nytte av når de anvender ActiveX-grensesnittet.

Gitt at grensesnittet skal benyttes i en bestemt utviklingsomgivelsen og at teknologien har sine fastlagte egenskaper/aspekter, angir ActiveX-grensesnittet en egen anvendelsesmodell som er noe mer enn summen av teknologiens – og utviklingsomgivelsens konseptuelle modell.

Måten man utnytter mulighetene til interaksjonsmekanismene når man utvikler ActiveX-komponenten vil være med å bestemme hvordan designeren skal bruke komponenten.
Ulike designløsninger ved nyttiggjøring av interaksjonsmekanismene, kan gi ulike anvendelsesmodeller for komponentgrensesnittet. Interaksjonsmekanismene til ActiveX har i likhet med eVB en del begrensninger som vil definere hva man faktisk kan tilby designeren av funksjonalitet i komponenten.
Da jeg utviklet Bluetooth-grensesnittet hadde jeg ideer om hvilke aspekter ved teknologien som grensesnittet skulle tilby designeren. Jeg dannet meg en mental modell av teknologien som jeg ville bringe videre til designeren i en konseptuell modell, men jeg var likevel begrenset av de mulighetene som ActiveX åpnet for og av min evne til å nyttiggjøre meg disse mulighetene.

Derfor ble anvendelsesmodellen ikke bare preget av teknologien jeg skulle presentere og av utviklingsomgivelsen eVB, men også av de designvalgene jeg måtte gjøre i forhold til mulighetene som ActiveX-komponentens interaksjonsmekanismer la til rette for.

Det er viktig å få fram at anvendelsesmodellen til teknologi-grensesnittet som jeg presenterte også ble gjort på grunnlag av hvem som skulle bruke grensesnittet og hvilke kvalitetskrav jeg mente at målgruppen ville ha til bruken av grensesnittet.

9.2.4 Mine erfaringer med hvordan teknologien kunne presenteres

På grunnlag av mine erfaringer med å utvikle Bluetooth-grensesnittet kan jeg komme med følgende punkter som er sentrale for å presentere et teknologisk grensesnitt på en måte som tilfredstiller kravene til brukskvalitet :

1. Teknologiens konseptuelle modell modell tok utgangspunkt i de abstraksjonene av teknologien som jeg gjorde. For at målgruppen (designere) skulle danne seg en forestilling om teknologien som var i samsvar med den konseptuelle modellen av teknologien som jeg ønsket å formidle, opplevde jeg at det var effektivt å presentere modellen på flere måter, slik at jeg var sikker på at jeg fikk formidlet den konseptuelle modellen tydelig nok til målgruppen.
2. Anvendelsesmodellen til grensesnittet måtte ta hensyn til både teknologiens - og utviklingsomgivelsens konseptuelle modell.
   Ved bruk av Bluetooth-grensesnittet ble Visual Basics konseptuelle modell utvidet med utvalgte Bluetooth-abstraksjoner og begrep, og med interaksjonsmekansmer i grensesnittet mellom teknologien og utviklingsomgivelsen.
   Dermed ble anvendelsesmodellen den helhetlige modellen for å løse oppgaven med å prototype Bluetooth-teknologien i Visual Basic.

3. Det var fordeler og ulemper med at grensesnittet jeg tilbød var ikke-visuelt.
   Fordelen med et ikke-visuelt grensesnitt var etter min mening at funksjonaliteten ble adskilt fra brukergrensesnittet, slik at designer hadde mer kontroll med hvordan prototypen ble seende ut, og at det ble større fleksibilitet, ved at brukeren hadde kunne sette sammen funksjonaliteten ved å referere direkte til grensesnittets metoder og eventer.
   Ulempen var at det sannsynligvis var en høyere terskel for å komme i gang å bruke grensesnittet enn med et visuelt grensesnitt.

Mine erfaringer tilsa altså at det var viktig å formidle teknologiens konseptuelle modellen på flere måter. Både gjennom dokumentasjon, som en muntlig presentasjon av Bluetooth og i selve grensesnittet (som navn på metoder, eventer, parametre osv. som gjenspeilte tankegangen i modellen).
I dokumentasjonen forsøkte jeg også å formidle hvordan interaksjonsmekanismene i grensesnittet kunne benyttes. Dette uttrykte jeg også muntlig til personene i testen.
I forsøkene testet jeg også om det var mulig å kompensere for ulempene ved et ikke-visuelt grensesnitt ved å tilby forsøkspersonene eksempelkode. Forsøkene antydet at eksempelkode var et nyttig hjelpemiddel i starten for Forsøksperson A og C for å komme i gang med prototypen, for å gjøre oppgaven mer konkret slik at den forsøkspersonene forsto mappingen mellom metodene/eventene og problemsløsningen, og for å skjønne tankegangen bak Bluetooth-grensesnittet.
Det er et åpent spørsmål om et visuelt grensesnitt eller høyere abstraksjoner i grensesnittet i virkeligheten ville medføre de fordeler eller ulemper som nevnt i punkt 3 overfor.
For eksempel ville det vært interessant å se om en automatisk håndtering av søkfunksjoner innad i andre metoder ville vært hensiktsmessig for designere (se kap. 8 om forbedring).
Figuren under viser

9.3 Delproblem 3

"Hvilken nytte gir det for designere å kunne prototype teknologiens funksjonelle egenskaper?"

9.3.1 Forutsetninger for at grensesnittet skal egne seg til prototyping

Bluetooth-caset har vist at det kan være nyttig å få demonstrert den type ny teknologi som Bluetooth i en funksjonell implementasjonsprototyp. På bakgrunn av tilbakemeldinger i forsøkene så virket det som om ActiveX-grensesnittet var enkelt å sette seg inn i og ga mulighet til å designe prototyper med en del interessant funksjonalitet. Abstraksjonen av Bluetooth-begrep og presentasjonen av anvendelsesmodellen (inkludert Bluetooth sin konseptuelle modell) ble gjort med den hensikt å imøtekomme de kravene til brukskvalitet for designere som jeg identifiserte i problemstillingen (se kap.2).

For å tilfredsstille kravet om at grensesnittet skulle gi designere rom for ideer og kreativitet, måtte grensesnittet gjøre designer i stand til å demonstrere de muligheter som teknologien ga. Dette betinget at abstraksjonene av teknologien (begrepene) var på et nivå som støttet designers oppgaveløsning og at abstraksjonene var presentert i en innpakning som ga mening for designer.
I kap 9.2 og 9.3 konkluderte jeg med at abstraksjoner og presentasjon av de konseptuelle modellene i Bluetooth-grensesnittet stort sett ga mening, men at det var rom for forbedringer i forhold til at designer blant annet kunne tilbys flere alternative måter å løse oppgavene på og metoder/eventer på enda høyere abstraksjonsnivå (se kap. 8 om forbedring av løsninger).

Det er imidlertid ingen selvfølge at funksjonelle prototyper er løsningen på ethvert problem. Det kan være ulike designspørsmål som det kan være interessant å få svar på og ikke alle spørsmål trenger å innebære en demonstrasjon av den implementerte funksjonaliteten.

9.3.2 Behovet for funksjonelle prototyper

Kunne ikke teknologien imidlertid vært prototypet ved hjelp av look-and-feel- eller rolle-prototyper? Disse prototypingsteknikkene er jo vanligvis både billigere, er enklere å gjennomføre og tar kortere tid.
For å svare på dette spørmålet må man se på hva som var formålet med prototypingen.
Hvis designeren kun var interessert i å få demonstrert hvordan brukergrensesnittet ble seende ut, ville en look-and-feel-prototyp vært bedre, og likeledes ville en rolleprototyp vært en billig og enkel løsning for å skissere hvilken betydning/rolle de ulike aspektene ved teknologien har for brukere (se kapittel 4 for mer om Houde og Hills presentasjon av ulike prototypingsteknikker).
Hvis formålet med prototypingen er å få en realistisk demonstrasjon av mulighetene og begrensningene til en ny teknologi som man ellers har begrenset kjennskap til, kan det være hensiktsmessig å vurdere en implementasjonsprototyp. Likeledes hvis man er ut etter å utforske samspillet mellom teknologien og brukergrensesnittet. Det kan være at man tidlig i et forskning - og utviklingsprosjekt vil se på hvordan man for eksempel kan lage brukergrensesnitt for applikasjoner som tar i bruk en ny kommunikasjonsteknologi som WLAN. Det kan da være vanskelig å få demonstrert interaksjonen mellom den nye

teknologien og brukergrensesnittet uten at man faktisk ser hvordan teknologien fungerer og hvordan brukergrensesnittet passet til teknologien.

Teknologier som Bluetooth og WLAN har en stor grad av interaktivitet og mangler innarbeidede høynivå-protokoller slik som TCP/IP og andre veletablerte kommunikasjonsstandarder og protokoller.

Det ligger også i sakens natur at er man først ute etter å prototype en teknologi som Bluetooth eller WLAN, er man gjerne ute etter å få demonstrert spesifikke egenskaper eller aspekter ved teknologien som bør prøves ut i praksis. Radioen må "stilles inn" på ordentlig. Det er vanskelig å simulere teknologien, fordi det er den spontane kontakten (eller eventuelt mangelen på kontakt) mellom enhetene som skal utforskes.

9.3.3 Når er funksjonelle prototyper nyttige?

Ut fra erfaringer jeg har gjort meg under utvikling av grensesnittet, tilbakemelding fra testene og teoretiske betrakninger rundt hensikten med prototyping, vil jeg i dette kapitlet si litt om når designeren generelt kan ha nytte av å prototype den reelle funksjonaliteten tidlig i designfasen. De viktigste oppdagelsene jeg har gjort i forhold til brukskravene (se kap. 2) og Delproblem 3 kan oppsummeres som :

Nytten av å benytte funksjonelle prototyper i forhold til kun å benytte look-feel- eller rolle-prototyper kan avhenge av *formålet* og sammenhengen det prototypes i :

- Hvis designeren ikke primært vil undersøke hvordan teknologien fungerer, men primært vil teste ut brukergrensesnittet og de konseptuelle aspektene ved teknologien, kan han trolig klare seg med å designe look-and-feel - eller papir/rolle- prototyper i startfasen av et systemutviklingsprosess. Slike prototyper kan både være billigere og være enklere og mindre tidkrevende å prototype, og funksjonaliteten blir da først introdusert senere i systemutviklingsprosessen.

- Hvis det er forholdet mellom teknologien og brukergrensesnittet som er interessant eller formålet er å få demonstrert muligheter og begrensninger ved teknologien, kan en funksjonell prototyp gi designeren nyttig informasjon som han ellers ikke ville fått med kun prototyping av skjermbilder eller papirsketcher (look-and-feel eller papir).

- Teknologier med høy grad av interaktivitet og nært forhold mellom maskinvare og funksjonalitet, kan det være vanskelig å illustrere eller simulere funksjonaliteten til uten å prøve ut den reelle funksjonaliteten på den aktuelle maskinvaren.

Avslutningsvis vil jeg si at designere nok er som folk flest ; de kan ha lettere for å forholde seg til det konkrete enn det vage og usikre.
En prototyp som demonstrerer alt fra skjermbilde-sekvenser til reell radiokommunikasjon kan både gi innspill til ulike aspekter ved Bluetooth og vise sammenhengen mellom Bluetooth-teknologien, brukergrensesnittet og betydningen teknologien har for vanlige brukere.

Det er imidlertid viktig at teknologi-grensesnittet gjør designeren i stand til å få demonstrert teknologien på en realistisk måte. Hvis grensesnittet samtidig har en lav terskel og samtidig

gir handlingsrom og uttrykkskraft er det lettere for designeren å få handlingsrom og utfolde sin kreativitet.
Hvis ikke grensesnittet oppfyller kravene til brukskvalitet, kan designeren like gjerne klare seg med look-and-feel- eller papir/rolle- prototyper. De er uansett ofte enklere og mindre ressurskrevende sammenlignet med implementasjonsprototyper, slik at designere kanskje benytter look-and-feel- eller papir/rolle- prototyper i første omgang, og eventuelt supplerer disse med implementasjonsprototyper.
Hvilke prototypingsteknikker som skal benyttes blir i alle tilfeller en vurderingssak. Hvilke aspekter ved designet eller teknologien designerne har behov for å evaluere kan også raskt forandre seg underveis i prosessen.

Implementasjonsprototyper som demonstrerer funksjonaliteten til en ny teknologi er altså hensiktsmessig hvis de gir designer nytte som han ellers ikke kan få kun ved bruk av andre prototypingsteknikker.

10. Retningslinjer for tilrettelegging av ny teknologi

Ut fra erfaringene jeg har hatt med utvikling og testing av et grensesnitt for ny teknologi har jeg kommet fram til noen punkter som det kan være viktig å være klar over for en utvikler som skal utvikle et lignende type grensesnitt.
Dette er ikke noe fasitsvar på hva man skal gjøre for å utvikle et hensiktsmessig grensesnitt, men retningslinjer som det kan være nyttig å være klar over slik at det kan bli lettere å oppnå de målsetninger man har satt seg som tilrettelegger av ny teknologi i en utviklingsomgivelse.
Da jeg utviklet grensesnittet mot Bluetooth hadde jeg som målsetning å imøtekomme de kvalitetskravene jeg antok at designere hadde. Grensesnittet skulle gjøre det enkelt for designer å prototype brukergrensesnitt og funksjonalitet, og skulle gi designeren rom for kreativitet og ideer for anvendelse.
Retningslinjene er utarbeidet for å gi en pekepinn på hva man kan gjøre når man legger til rette for prototyping av ny teknologi for lettere å imøtekomme krav til brukskvalitet (se kap. 4.3.2 om usability og kap. 2 om problemstilling).

Kort oppsummert gir punkt 1-5 noen råd for hvordan man kan velge utviklingsomgivelse og rammeverk og betoner viktigheten av å sette seg inn i teknologien i en tidlig fase.
Punkt 6-11 tar for seg hvordan man kan gå fram for å abstrahere teknologien og presentere den i grensesnittet i en forenklet utgave.
Punkt 12 berører behovet for å lage en funksjonell implementasjonsprototyp.

10.1 Valg av utviklingsomgivelse og teknologisk rammeverk

Punkt 1-5 antyder at det er naturlig å først sette seg godt inn i teknologien som skal abstraheres og finne en egnet utviklingsomgivelse hvor grensesnittet kan benyttes.
Samtidig bør man tenke ut hvilke teknologier/verktøy/utviklingsomgivelser man vil bruke for å lage grensesnittet, noe som avgjør hvilket type grensesnitt man kan lage.

1. *Man bør begynne med å sette seg inn i teknologien så tidlig som mulig* slik at man har klart for seg hvordan man skal gå fram for å presentere teknologien og hvilke teknologiske muligheter og begrensninger som foreligger. Man bør utrede hvilken maskinvare man trenger. Hvis man velger den mobile plattformen er det spesielle utfordringer knyttet til mangel på standardisering, uklar topologi (klient-tjener, ulike interagerende enheter osv.) og maskinvareproblemer.
   Det er dessuten begrenset hva som er gjort av forskning og utvikling på verktøy for prototyping av ny teknologi, så man har ikke så mye å støtte seg på.

2. *Finn fram til en utviklingsomgivelse som er velegnet for prototyping og gir tilgang til teknologien på en enklest mulig måte.* En løsning kan være å lage grensesnitt i en utviklingsomgivelse som kan benyttes i en annen omgivelse. Utviklingsomgivelsen bør legge til rette for hurtig prototyping av applikasjoner. Det være enkelt å designe brukergrensesnittet og programmere funksjonalitet og det må være mulig å utvide funksjonaliteten med eksterne komponenter hvis ikke funksjonaliteten inngår i selve utviklingsomgivelsen.

3. *Grensesnittet mot teknologien kan ikke utformes før man har fått kontakt med den underliggende funksjonaliteten.* Man må kontinuerlig teste ut endringer i grensesnitt og implementasjon ved å bygge opp og kjøre en prototyp, slik at man får bekreftet at grensesnittet "virker". Det kan imidlertid være lurt å først lage et tomt "hello world"-grensesnitt for i det hele tatt å få grensesnittteknologien til å virke, før man kan utføre virkelige operasjoner mot den underliggende teknologien/ funksjonaliteten. På denne måten ser man hvor feilen ligger i tilfelle man ikke klarer å få grensensittet til å kjøre ordentlig.
   Det kan være vanskelig å finne feil hvis flere applikasjoner/ utviklingsomgivelser samhandler, da det er flere ulike feilkilder og man er avhengig av at den underliggende teknologien og de applikasjonene over får kontakt med hverandre og interagerer på riktig måte.

4. *Man bør ha i tankene de mulige interaksjonsmekanismene for kontakt mellom utviklingsomgivelse og den underliggende funksjonalitet* når man velger teknologisk rammeverk. Man er gjerne prisgitt de interaksjonsmekansismer det er tilgang til i rammeverket når man har bestemt seg for maskinvare, operativsystemplattform og utviklingsomgivelse for utvikling av verktøyet.

5. *Arbeidet med å implementere grensesnittet henger nøye sammen med det konseptuelle designet av grensesnittet.* Verken implementasjonen eller designet av grensesnittet kan gjøres adskilt, da dette er en iterativ og interaktiv prosess med mye prøving og feiling. Hvis flere personer er involvert i utvikling av et grensesnitt mot ny teknologi er tett samarbeid i teamet derfor nødvendig. Noen kan ha ansvar for implementasjon og andre for design, men alle i utviklingsteamet samarbeider om utformingen av grensesnittet. Det konseptuelle designet utformes gjerne ved hjelp av for eksempel papirsketcher eller idedugnad, og tilpasses så selve implementasjonen av grensesnittet.
   Man kan erfare at det konseptuelle designet av grensesnittet ikke lar seg implementere og må tilpasses de tekniske muligheter som finnes. Det er også mulig at den tekniske implementasjonen ikke er forenlig med et brukervennlig grensesnitt og må endres.

10.2 Abstraksjon av teknologiens kompleksitet

Punkt 6-8 foreslår hvordan man kan gå fram for å abstrahere de teknologiske begrepene på en måte som gjør grensesnittet brukervennlig og fleksibelt. Formålet med funksjonelle prototyper er å få demonstrert mulighetene og begrensningene som ligger i ulike egenskaper ved teknologien. Grensesnittet bør derfor gi designeren nok uttrykkskraft samtidig med at det er enkelt å benytte. Punktene sier også hvordan man skal utnytte de interaksjonsmekanismene man har til rådighet i grensesnittet (som er fastlagt i rammeverket) til det beste for designeren.

6. *Finn et abstraksjonsnivå som er tilpasset designeren og abstraher det man tror designeren har interesse og nytte av.* Abstraher vekk det som designeren ikke behøver å vite noe om.
   La abstraksjonen av de teknologiske begrepene danne grunnlaget for den konseptuelle modellen til teknologien.
   For lavt abstraksjonsnivå gjør det vanskelig for designer å konsentere seg om oppgaveløsningen. For høyt abstraksjonsnivå kan redusere fleksibiliteten og valgmulighetene og dermed begrense hva designeren kan utrette med teknologien.

Man må regne med å miste en del informasjon om teknologien når man abstraherer. Vær derfor ekstra nøye med å dokumentere teknologigrensesnittet. Vis gjerne litt "under panseret"og røp noen detaljer rundt hvordan teknologien fungerer, så lenge dette ikke skaper forvirring.

7. *Definer hvor mye designeren skal gjøre selv og hvor mye grensesnittet skal gjøre for ham. Tilby gjerne flere abstraksjonsnivå og alternative måter å benytte grensesnittet på.* Dette kan øke fleksibiliteten og gi rom for individuell tilpasning. Grensesnittet kan være visuelt eller ikke-visuelt. Det er både fordeler og ulemper med visuelle grensesnitt. Det kan gjøre det lettere å komme i gang og benytte grensesnittet, men samtidig kan designerens muligheter bli begrenset av hva det visuelle grensesnittet legger opp til og fleksibiliteten blir mindre.

8. *Lag et helhetlig og lettforståelig grensesnitt som fungerer som et mellomledd mellom utviklingsomgivelsen og den nye teknologien*. Vær nøye med å ha klar og konsistent syntaks og navngivning i grensesnittet. Uansett hvilke interaksjonsmekanismer man kan utnytte til å la designeren få tilgang til teknologien, bør man bestreve seg etter å velge den mest brukervennlige løsninger. Dette er bedre enn å tilby en teknisk avansert løsning, med flere muligheter, men som de færreste klarer å benytte.

10.3 Presentasjon i en konseptuell anvendelsesmodell

Punkt 9-11 antyder nytten av å utforme en helhetlig anvendelsesmodell for grensesnittet som bygger på en konseptuell modell for teknologien og utviklingsomgivelsens konseptuelle modell. Det er gunstig å tenke ut teknologiens konseptuelle modell så tidlig som mulig - på grunnlag av de kunnskaper om teknologien man tilegner seg - og anvendelses-modellen for grensesnittet.

9. *Utform teknologiens konseptuelle modell* ved å abstrahere de viktigste egenskapene ved teknologien og presentere teknologien i en forenklet "innpakning". Designeren bør kunne danne seg en forestilling som er tilnærmet lik det bildet man ønsker å gi av teknologien. La designer dra nytte av tidligere erfaring med lignende teknologi. Grensesnittet bør kunne brukes både for folk med og uten erfaring med den spesifikke teknologien. Teknologiens konseptuelle modell bør derfor være sammenlignbar i forhold til lignende teknologier.
10. *Ta hensyn til utviklingsomgivelsens konseptuelle modell*. Utviklingsomgivelsen som brukes til å lage prototyper har gjerne en nøye utformet konseptuell modell som designeren er vant til og som han forholder seg greit til.
    Grensesnittet bør harmonere med utviklingsomgivelsens konseptuelle modell slik at det føles for designeren som om han benytter et objekt eller en egenskap i utviklings-omgivelsen.
    Helst bør grensesnittet kunne brukes på samme måte som andre tilgjengelige objekter, og for å benytte grensesnittets metoder så skal ikke designer være nødt til å utføre kompliserende tilleggsoperasjoner. Dette fjerner bare fokus fra designoppgaven.

11. *Utform en klar og konsistent anvendelsesmodell for grensesnittet som reflekterer og harmonerer med både utviklingsomgivelsens - og teknologiens konseptuelle modell.*
Anvendelsesmodellen kan sees på som den helhetlig modellen som designer forholder seg til når han benytter grensesnittet i utviklingsomgivelsen til å prototype ny teknologi. Utviklingsomgivelsens konseptuelle modell blir dermed utvidet med den konseptuelle modellen til teknologien til en helhetlig anvendelsesmodell. Det er viktig at denne utvidelsen føles så naturlig som mulig for designeren.

10.4 Behovet for en implementasjonsprototyp

12. *Grensesnittet er et verktøy for å lage funksjonelle prototyper og må kunne benyttes til å lage realistiske prototyper som demonstrerer teknologien så godt som mulig.*
Få formidlet til designer hvordan teknologien fungerer og hvilken betydning ulike aspekter ved teknologien har for anvendelsen av den. Hvis man ikke får demonstrert de vesentlige trekkene realistisk kan designeren like gjerne lage look-and-feel- eller papir/rolle- prototyper.

11. Konklusjon

11.1 Oppsummering og svar på delproblem

I hovedfagsoppgaven har jeg diskutert hvordan man på best kan legge til rette for prototyping av ny teknologi.
Bakgrunnen var at nye teknologier ofte har vært forbeholdt noen få eksperter og generelt lite tilgjengelig for vanlige folk Jeg ville se på hvordan man kunne gjøre nye teknologier tilgjengelige slik at også ikke-eksperter med kun grunnleggende programmerings - og data-kunnskaper kunne prototype aspekter ved teknologien.

Ut fra denne problemstillingen ble det utviklet en ActiveX-komponent som fungerte som et forenklet grensesnitt mot den trådløse radioteknologien Bluetooth. For å oppnå dette var det nødvendig å fjerne noe av kompleksiteten og presentere teknologien i en forenklet innpakning i grensesnittet.
Utviklingen av ActiveX-komponenten blir i denne oppgaven framstilt som et case som illustrerer hvordan man generelt kan legge til rette for prototyping av ny teknologi i en utviklingsomgivelse.

I forbindelse med problemstillingen presenterte jeg noen kvalitetkrav som jeg mente det var naturlig å stille til et teknologi-grensesnitt for at prototyping av teknologien skulle gi mest mulig nytte for designere.
Gjennom kvalitetskravene ville jeg identifisere hvilke hovedkriterier som lå til grunn for at teknologien ble tilrettelagt på best mulig måte. Kvalitetskravene var at grensesnittet måtte være velegnet til å prototype brukergrensesnitt og funksjonalitet og gi mulighet til å demonstrere sentrale og interessante aspekter ved teknologien slik at designeren fikk ideer om hvordan teknologien kunne anvendes (se kap. 2 om problemstilling).

Jeg har i denne oppgaven forsøkt å svare på hvordan de enkelte delproblemene kunne løses, ut fra teoretisk bakgrunn fra litteratur og erfaringer jeg hadde med å utvikle og teste ActiveX-komponenten. Problemstillingen ble delt inn i tre delproblemer, og løsningene på del-problemene har blitt vurdert ut fra hvordan kvalitetskravene til designere har blitt ivaretatt (se problemstilling i kap 2).

Abstraksjon

Første delproblem dreide seg om hvordan teknologi kunne abstraheres.

Jeg kom fram til at ut fra mine erfaringer med å abstraherte teknologi kompleksitet, var det viktig å velge ut interessante og sentrale begrep som ga mening for designeren.
Når man abstraherer må man velge hvilke begreper som skal tas med i abstraksjonen og hvilke som skal skjules.
Jeg har vist til hvordan jeg gjemte teknisk kompleksitet i Bluetooth-komponenten og eksponerte den forenklede abstraksjon i grensesnittet. Jeg ville med dette illustrere at hvilke begreper som tas med i abstraksjonen får innvirkning på hvordan designer skal forholde seg til teknologien. Designer forholder seg til det som er eksponert i grensesnittet.
Det var også viktig å velge et abstraksjonsnivå som gjorde det lettest for designer å løse sine oppgaver.

Presentasjon av teknologien

Andre delproblem dreide seg i sin tur om hvordan de teknologiske abstraksjonene av teknologien kunne presenteres for designere.

Det kom så fram hvordan jeg gikk fram for å bygge opp en konseptuell modell av teknologien ved å ta utgangspunkt i abstraksjonene. Jeg har påpekt utfordringene med å få formidlet en konsistent og klargjørende konseptuell modell til designer.

Jeg har videre framhevet hvor viktig det var å ta hensyn til den eksisterende konseptuelle modellen til utviklingsomgivelsen benyttet av designer, og framstilte anvendelsesmodellen til ActiveX-komponenten som en helhetlig modell for bruken av komponenten. Denne besto av både teknologiens –og utviklingsomgivelsens konseptuelle modell.
I tillegg bestemte måten interaksjons-mekanismene i ActiveX-komponenten var blitt utnyttet på hvordan anvendelsesmodellen ble. Utnyttelsen av disse mekanismene utgjorde mange av de designvalgene jeg måtte gjøre ved utvikling av grensesnittet mot Bluetooth.

Nytten av å demonstrere teknologiens egenskaper

I tredje delproblem spurte jeg meg hvilken reell nytte designere har av kunne prototype teknologiens funksjonelle egenskaper.

Jeg diskuterte under hvilke omstendigheter det ville være mest nærliggende for designeren å lage realistiske/ funksjonelle implementasjonsprototyper og når han like godt kunne klare seg med prototyper som bare illustrerte brukergrensesnittet eller konseptuelle aspekter. Jeg kom fram til at nytten i å lage implementasjonsprototyper framfor å klare seg med look-and-feel – eller papir/rolle- prototyper kom an på hva designerens formål med prototypen var og hva som skulle testes.

Ut fra erfaringene jeg har gjort meg i dette studiet ble det i kapittel 10 presentert noen retningslinjer for hvordan man generelt kan gå fram for å legge til rette for prototyping av ny teknologi i en utviklingsomgivelse. Disse retningslinjene kan ses på som den lærdommen jeg har å gi til de som senere skal legge til rette for prototyping av ny teknologi.

Overraskelser

Det jeg synes var mest overraskende var hvor viktig det var å tilpasse abstraksjonene til designernes problemløsning og at det å tilby ulike abstraksjonsnivå og alternative måter å utføre oppgavene trolig kan gjøre teknologigrensesnittet mer fleksibelt. Selv om forsøkene i dette hovedfagsstudiet hadde sine mangler og ikke kan nyttes til å dra endelige konklusjoner, erfarte jeg at de fire forsøkspersonene hadde ganske ulike ønsker med hensyn til hvor mye av teknologiens funksjonalitet som skulle være tilgjengelig i grensesnittet. Det ble ytret ønske både om å få vite hva som var "under panseret" og om abstraksjoner over det presenterte grensesnittet.
Jeg trodde i utgangspunktet at det ville være mest brukervennlig med et visuelt grensesnitt. Utviklingen og testingen av komponenten tilsa imidlertid at det var argumenter både for og

imot en slik løsning. Det var altså ikke selvsagt at Bluetooth-grensesnittet ble presentert med et ferdigordnet brukergrensesnitt.
I utviklingsomgivelser som Visual Basic er det enkelt å designe brukergrensesnitt, og inntrykket mitt fra dette hovedfagsstudiet er at designere helst vil ha mest mulig kontroll over utformingen av designet.

11.2 Metodekritikk

Den praktiske delen av studiet besto av utvikling av et Bluetooth-grensesnitt, og testing av grensesnittet.
Som jeg påpekte i kapittel 3.4 var testmaterialet for begrenset til at jeg kunne trekke noen endelige konklusjoner ut fra testene. Det ble gjennomført fire tester med i alt fire testpersoner. Disse fikk noe hjelp underveis i oppgavene med å benytte Bluetooth-grensesnittet til prototyping. Dette var først og fremst hjelp til Visual Basic-spesifikke problemer.
Da jeg som testleder både observerte og deltok i prototypingen må det tas med i betrakningen at jeg kan ha hatt innvirkning på resultatet.
Dessuten var testpersonene trolig for homogent sammensatt til at de var representativ for en uensartet gruppe av designere. Dette var altså ingen fullstendig usability-test, men heller en heuristisk gjennomgang av grensesnittet.

Testene har vært nyttige for å vurdere Bluetooth-grensesnittets styrker og svakheter, og har dannet grunnlag for forbedring og videreutvikling.
I testene har jeg prøvd å få noen svar på hvor velegnet abstraksjonene av teknologien og presentasjonen av anvendelsesmodellen har vært for å gjøre det enkelt for designere å prototype og gi rom for ideer og kreativitet.
Testene gir også retningslinjer for hva det er viktig å være klar over når man lager forenklede grensesnitt mot ny teknologi. På denne måten kan testene si noe om hvordan man best tilrettelegger for prototyping generelt.
Dessuten ga selve utviklingen av grensesnittløsningen nyttig kunnskap om prosessen med tilrettelegging av ny teknologi.

Til tross for at testene ikke gir grunnlag for å dra sikre konklusjoner, synes jeg likevel at denne oppgaven gir et verdifullt innspill til hva man bør ta hensyn til når man forenkler teknologisk kompleksitet, og presenterer likevel denne forenklingen (abstraksjonen) til designere.

Resultatene av dette studiet kan ha overføringsverdi til kommende problemstillinger hvis de blir oppfattet som en pekepinn på hvordan utfordringer med tilrettelegging av teknologi-grensesnitt kan løses, og ut fra en erkjennelse av at testene ikke gir definitive svar på problemstillingen.

11.3 Mine tanker rundt hovedfagsstudiet

Arbeidet med hovedfagsoppgaven har bestått av en teoretisk del og en praktisk del.

Den teoretiske delen besto av litteraturstudium og skriving av hovedfagsoppgaven. Gjennom teorien har jeg jobbet selvstendig med modeller, teori og problemstillinger innen fagfeltet Menneske-maskin-interaksjon.

Den praktiske delen har hovedsakelig dreid seg om utvikling av et forenklet grensesnitt mot Bluetooth og om testing av denne løsningen.

Jeg har i samarbeid med Frank Werner Johansen gjort implementasjon og design av en Bluetooth-komponent. Selv om jeg hadde hovedansvaret for designet av Bluetooth-komponenten, har jeg møtt både tekniske og konseptuelle problemstillinger i forbindelse med utviklingen av komponenten.
Det var mye prøving og feiling for å få den tekniske løsningen til å virke og det var krav til at komponenten skulle være enkel å benytte.

Gjennom utviklingen og testingen av Bluetooth-komponenten har jeg kunnet knytte teori innen fagfeltet Menneske-maskin-interaksjon opp mot hvordan hvordan teorien kan anvendes til å lage brukervennlige løsninger.

En ny utfordring kan være å se på hvordan Bluetooth-komponenten kan videreutvikles, slik at den kan benyttes til å demonstrere kontekstsensitivitet i en omgivelse der ulike typer enheter samhandler.

Jeg synes denne hovedfagsoppgaven har illustrert hva det er viktig å tenke på når man tilrettelegger for prototyping av ny teknologi. Jeg har vist at måten abstraksjonen av teknologien blir utført på har stor betydning og at det er viktig med en tydelig og konsistent konseptuell modell.

Kilder :

Brookshear, J.G. (2003): Computer Science, an Overview – Edition 7. Addison-Wesley

Carr M. og J. Verner (1995): Prototyping and Software Development Approaches.
Webkilde: http://www.is.cityu.edu.hk/Research/WorkingPapers/paper/9704.pdf

Carroll, J.M. (2002): Introduction; Human-Computer Interaction, the Past and the Present in Human Computer Interaction in the New Millenium. Addison Wesley

Durham, J. (2001): History-making components. IBM developerWorks.
Webkilde: http://www-106.ibm.com/developerworks/library/co-tmline/

Habermann, A.N. (1986): Technological advances in software engineering. ACM

Hartvigsen, G. (1998): Forskerhåndboken. Høyskoleforlaget

Houde, S. og Hill C. (1997): What do Prototypes Prototype?. Apple Computer Inc.

ISO 13407 (1999)

ISO 9241.11 (1998)

Johnson og Henderson (2002): Conceptual Models: beginning by designing what to design. ACM Interactions

Khella, A. (2002): Knowledge and Mental Models in HCI
Webkilde: http://www.cs.umd.edu/class/fall2002/cmsc838s/tichi/knowledge.html

Myers, B (1994): Challenges of HCI Design and Implementation. ACM Interactions

Myers, B. (1995): User Interface Software Tools. ACM

Myers, B. (1998): A brief history of HCI technology. ACM Interactions

Myers, B m.fl. (2000): Past, Present and Future of User Interface Software Tools. ACM

Maus, A. (2002): Objektorientert programmering og systemutvikling - en kortfattet innføring. Institutt for informatikk. Universitet i Oslo

Naumann J. og Jenkins M. (1982): Prototyping: The New Paradigm for System Development. MIS Quarterly

Nielsen, J. (1986): Usability Engineering. Academic Press Inc.

Norman, D. (1986): Cognitive Engineering. In User Centered System Design. Lawrence Erlbaum Associates

Norman, D. (1988): The Psychology of Everyday Things. Basic Books

Preece, J. m.fl. (1994): Human Computer Interaction. Addison-Wesley

Rowe (1980): Data Abstraction from a Programming Language Viewpoint. Computer Science Division-EECS University of California

Tognazzini B. (1992): TOG on Interface. Addison-Wesley

Wang, J.A. (2000): Towards Component-based software engineering. The Consortium for Computing in Small Colleges

Weiss S. (2002): Handheld Usability. John Wiley& Sons Ltd.

Winograd,T. (1995): From Programming Environments to Environments for Designing. Communications of the ACM

Websider:

Carnegie Mellon University : On Overview(2004)
Webkilde: http://www.cmu.edu/home/about/about.html

Free On-Line Dictionary of Computing (2003)
Webkilde: http://foldoc.doc.ic.ac.uk/foldoc/

MSDN: On Automation and ActiveX (2003)
Webkilde: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnaxctrl/html/msdn_events.asp

The Xerox Star: A Retrospective (2004)
Webkilde: http://www.digibarn.com/friends/curbow/star/retrospect/

Vedlegg 1 : Dokumentasjon av ActiveX-Komponentens grensesnitt slik det ble presentert i Forsøk C og D

Bluetooth-komponenten Bluedoo

Oversikt over eventer og metoder

Denne ActiveX komponenten er beregnet for kommunikasjon mellom mobile enheter. Den gjør det mulig å prototype Bluetooth-funksjonalitet i Embedded Visual Basic ved hjelp av meget enkel programmering.

Du bygger opp et skjermbilde i grensesnittbyggeren i eVB og programmerer funksjonaliteten bak skjermbildet ved hjelp av kode-editoren i eVB.
Komponenten legges på verktøylinja i eVB ved å gå inn på Project->Component på kontroll-menyen. Dra da bare komponenten 'BT' over fra verktøylinja til grensesnittbyggeren og vipps så kan den benyttes til å gi prototypen kontakt med Bluetooth.
Prototypen overføres så til de enhetene som skal kommunisere når du vil prøve ut det du har laget.

Det samme programmet kjøres altså på flere enheter. Merk deg at ingen av enhetene er tjener eller klient i ordets rette forstand. En enhet starter kommunikasjonen, men så vil enhetene bytte på å sende og motta data.

EVENTER

En Event er en slags metode som fyres når noe skjer. En event er et nyttig hjelpemiddel for å programmere interaktive applikasjoner.

Noen eventer fyres som følge av trykk på knapper eller andre elementer i brukergrensesnittet, og behandles automatisk av Windowsmanager.
Andre eventer fyres som følge av andre hendelser som ikke befinner seg i det synlige grensesnittet. Akkurat som at man angir i en 'Sub-metode' funksjonaliteten til en knapp, lager man en tilsvarende Sub-metode for funksjonaliteten som en event medfører.
F.eks kan en funksjon slå seg på når en tidsmåler(klokke) viser en viss verdi eller en termostat reagerer på temperaturøkning med å slå av strømmen.

Du benytter altså en event i programmeringskoden nesten på samme måte som en metode.
I metoden kan det defineres hva som skal skje når eventen fyres.
Det er imidlertid din oppgave å lagre enhetsnavnene. Hvordan det skjer spiller ingen rolle.

Framgangsmåte for å sende tekst eller filer er for eksempel:

1.Gjør et søk etter andre enheter vha metoden StartSok eller StartStoppSok
2.Legg enheter i ei liste etter hvert som de blir funnet.
Benytt derfor eventen EnhetFunnet(enhet) og legg enhetene, representert ved enhetsnavn, i ei liste. eks. Bluetooth_EnhetFunnet(enhet) (erstatt "." med "_")
3.Hent den enheten du vil sende til ut av lista, og bruk denne enheten som parameter i metoden SendFil/SendTekst for å indikere hvilken enhet du vil sende til

EnhetFunnet(enhetsnavn)
Beskrivelse: Fyres når en annen enhet oppdages.
Forutsetter: Det pågår søk etter andre enheter (vha. 'StartSok' eller 'StartStoppSok')
Parametre: enhet – navnet på bluetooth-enheten som er funnet.

TekstMottatt()
Beskrivelse: Fyres når det kommer ny tekst inn til enhetens buffer
Forutsetter: Tekst er sendt til enheten fra en annen enhet
Parametre: ingen

SokStartet()
Beskrivelse: Fyres når et enhetssøk startes.
Forutsetter: Det pågår søk etter andre enheter (vha. 'StartSok' eller 'StartStoppSok')
Parametre: ingen

SokStoppet()
Beskrivelse: Fyres når et enhetssøk stoppes.
Forutsetter: Det pågår søk etter andre enheter (vha. 'StartSok' eller 'StartStoppSok')
Parametre: ingen

METODER

StartSok()
Beskrivelse: Starter et søk etter bluetooth-enheter. Søket varer i en viss periode inntil det går i time-out.
Forutsetter: Ingenting
Parametre: Ingen
Returverdi: Ingen

StoppSok()
Beskrivelse: Stopper et søk etter bluetooth-enheter.
Forutsetter: Ingenting
Parametre: Ingen
Returverdi:Ingen

StartStoppSok(tid)
Beskrivelse: Starter eller stopper et intervallsøk. Når intervallsøket starter så søkes det etter bluetooth-enheter i et gitt tidsintervall inntil søket stoppes og venter til søket startes igjen.

Forutsetter: Ingenting
Parametre: tid - Angir i hvor mange millisekunder tidsintervallet er.
Returverdi: Returner 0 for å vise at intervallsøket er stoppet
Returnerer 1 for å vise at intervallsøket er startet.

SendFil(fil, enhetsnavn)
Beskrivelse: Sender en fil til en angitt enhet.
Forutsetter: Det er gjort søk etter andre enheter vha. 'StartSok' eller 'StartStoppSok'
Eventen EnhetFunnet er benyttet for å lagre informasjon om enheter man finner
Parametre: fil - angir plasseringen og navnet til fila og
enhet - angir navnet på bluetooth-enheten man vil sende fila til.
Returverdi:Ingen

SendTekst(tekst, enhetsnavn)
Beskrivelse: Sender tekst til en fjern bluetooth-enhet.
Forutsetter: Det er gjort søk etter andre enheter vha. 'StartSok' eller 'StartStoppSok'
Eventen 'EnhetFunnet' er benyttet for å lagre informasjon om enheter man finner
Parametre: tekst - den teksten som skal sendes og enhet – den enheten teksten skal sendes til
Returverdi: Ingen

LesTekst()
Beskrivelse: Leser teksten som har blitt sendt fra en annen bluetooth-enhet.
Forutsetter: Det er opprettet en forbindelse mellom to enheter
Parametre: Ingen.
Returverdi: Teksten som har blitt mottatt fra en bluetooth-enhet.

OppgiSignalstyrke(enhetsnavn)
Beskrivelse: Oppgir signalstyrken som den lokale Bluetooth-enheten har mot en annen Bluetooth-enhet.
Forutsetter: Det er gjort søk etter andre enheter vha. 'StartSok' eller 'StartStoppSok'
Parametre: enhet - angir navnet på bluetooth-enheten man vil finne signalstyrken til.
Returverdi: Returner signalstyrkene LAV, GOD og HØY.

OppgiMittNavn()
Beskrivelse: Oppgir navnet på den lokale bluetooth-enheten.
Forutsetter: Ingenting
Parametre: Ingen
Returverdi: Returnerer navnet på den lokale bluetooth-enheten.

EndreMittNavn(mittenhetsnavn)
Beskrivelse: Endrer navnet på den lokale bluetooth-enheten.
Forutsetter: Ingenting
Parametre: mittenhetsnavn - angir det nye navnet på den lokale bluetooth-enheten.

Returverdi: Ingen

OppgiNavnTilknyttetEnhet()
Beskrivelse: Oppgir navnet på en bluetooth-enhet man har serieforbindelse med.
Forutsetter: Det er opprettet en serieforbindelse vha OpprettSerieForbindelse eller det er sendt en tekst vha 'SendTekst'
Parametre: Ingen.
Returverdi: Navnet på en annen bluetooth-enhet.

Sjekktjeneste(tjeneste, enhetsnavn)
Beskrivelse: Sjekker om en angitt tjeneste er tilgjengelig på en enhet
Forutsetter: Det er opprettet en forbindelse mellom to enheter
Parametre: tjeneste: den typen tjeneste man skal benytte, f.eks 'sendfil', og navnet på enhet
Returverdi: 1 hvis tjeneste finnes, ellers 0

OpprettSerieForbindelse(enhetsnavn)
Beskrivelse: Oppretter en serieport forbindelse mot en annen bluetooth-enhet.
Forutsetter: det er gjort søk etter andre enheter vha 'startSok' eller 'startStoppSok'
Parametre: enhet - navnet på bluetooth-enheten man vil ha forbindelse med.
Returverdi: Returner 1 hvis serieforbindelse ble opprettet.
Returner 0 hvis serieforbindelse ikke ble opprettet.

AvsluttSerieForbindelse(enhetsnavn)
Beskrivelse: Avslutter en serieport forbindelse.
Forutsetter: det er gjort søk etter enheter (startSok eller startStoppSok)
Parametre: enhet - navnet på bluetooth-enheten man vil avslutte forbindelsen med.
Returverdi: Ingen

SendBusinessCard(enhetsnavn)
Beskrivelse: Sender business-card til en annen enhet
Forutsetter: Det er gjort søk etter andre enheter vha. 'StartSok' eller 'StartStoppSok'
Parametre: enhet - navnet på bluetooth-enheten man vil avslutte forbindelsen med.
Returverdi: Returner 1 hvis business/card ble sendt
Returner 0 hvis business/card ikke ble sendt

MottaBusinessCard(enhetsnavn)
Beskrivelse: Tar imot business-card som en annen enhet har sendt
Forutsetter: Et business-card er sendt
Parametre: enhet - navnet på bluetooth-enheten man vil avslutte forbindelsen med.
Returverdi: Returner 1 hvis business/card ble mottatt
Returner 0 hvis business/card ikke ble mottatt

UtveksleBusinessCard(enhetsnavn)
Beskrivelse: Sender business-card til en annen enhet og mottar samtidig fra denne

Forutsetter: Det er gjort søk etter andre enheter vha. 'StartSok' eller 'StartStoppSok'
Parametre: enhet - navnet på bluetooth-enheten man vil avslutte forbindelsen med.
Returverdi: Returner 1 hvis business/card ble utvekslet
Returner 0 hvis business/card ikke ble utvekslet

OpprettBond(pinkode, enhetsnavn)
Beskrivelse: Bonder med en annen enhet. Brukes for eksempel til mobilhandsfree
Forutsetter: Ingenting
Parametre: enhet - navnet på bluetooth-enheten man vil avslutte forbindelsen med og pinkoden
Returverdi: 1 hvis bonding lyktes 0 hvis det mislyktes

AvsluttBond(enhetsnavn)
Beskrivelse: Avslutter en bonding og gjør enheten discoverable mot andre enheter igjen
Forutsetter:Det er gjort en bonding som nå skal avsluttes
Parametre: enhet - navnet på bluetooth-enheten man vil avslutte forbindelsen med.
Returverdi:Ingen

Vedlegg 2 : Programmeringskoden til prototyp laget i eVB i Forsøk D

```
Option Explicit

Private Sub Bluetooth1_EnhetFunnet(ByVal enhet As String)

   List1.AddItem (enhet)

End Sub

Private Sub Command1_Click()

   Bluetooth1.StartSok

End Sub

Private Sub Command2_Click()

   Bluetooth1.StoppSok

End Sub

Private Sub Command3_Click()

   Dim i As Integer
   Dim Signal As String
   List2.Clear
   i = 0

   For i = 0 To (List1.ListCount - 1)
      Signal = Bluetooth1.OppgiSignalstyrke(List1.List(i))
      List2.AddItem (Signal)
   Next
End Sub

Private Sub Command4_Click()
   Bluetooth1.SendFil "\Temp\PIMAgent.txt", List1.List(List1.ListIndex)
End Sub

Private Sub Command5_Click()

   Bluetooth1.SendTekst Text1.Text, List1.List(List1.ListIndex)

End Sub

Private Sub Form_OKClick()
   App.End
End Sub

Private Sub List1_DblClick()
   Command4_Click
End Sub
```

```
Private Sub List2_Click()

End Sub

Private Sub Timer1_Timer()
   Command3_Click
End Sub


Private Sub Timer2_Timer()

   Text2.Text = Bluetooth1.LesTekst

End Sub
```

Vedlegg 3 : Brukergrensesnittet til prototyp laget i eVB i Forsøk D