FFIE/730/134

Godkjent
Kjeller 15 september 2001

Vidar S Andersen

Vidar S Andersen
Forskningssjef

ARKITEKTURER FOR KOMMANDO OG KONTROLL INFORMASJONSSYSTEMER

Arkitekturer, komponentbasert systemutvikling og teknologier

HAFNOR Hilde, ELVESÆTER Brian, VEUM Kurt A,
VIKEN Kjell

FFI/RAPPORT–2000/04582

FORSVARETS FORSKNINGSINSTITUTT
Norwegian Defence Research Establishment
Postboks 25, 2027 Kjeller, Norge

FORSVARETS FORSKNINGSINSTITUTT (FFI)
Norwegian Defence Research Establishment
P O BOX 25
NO–2027 KJELLER, NORWAY

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE
(when data entered)

REPORT DOCUMENTATION PAGE

1) PUBL/REPORT NUMBER	2) SECURITY CLASSIFICATION	3) NUMBER OF PAGES
FFI/RAPPORT–2000/04582	UNCLASSIFIED	96
1a) PROJECT REFERENCE	2a) DECLASSIFICATION/DOWNGRADING SCHEDULE	
FFIE/730/134		

4) TITLE

ARKITEKTURER FOR KOMMANDO OG KONTROLL INFORMASJONSSYSTEMER
Arkitekturer, komponentbasert systemutvikling og teknologier
(ARCHITECTURES FOR COMMAND AND CONTROL INFORMATION SYSTEMS)

5) NAMES OF AUTHOR(S) IN FULL (surname first)

HAFNOR Hilde, ELVESÆTER Brian, VEUM Kurt A, VIKEN Kjell

6) DISTRIBUTION STATEMENT

Approved for public release. Distribution unlimited
(Offentlig tilgjengelig)

7) INDEXING TERMS

IN ENGLISH:	IN NORWEGIAN:
a) Information system	a) Informasjonssystem
b) Architecture	b) Arkitektur
c) C3 Interoperability	c) Operativ interoperabilitet
d) Component based developement	d) Komponentbasert systemutvikling
e) Distributed processing	e) Distribuert prosessering

THESAURUS REFERENCE: Inspec

8) ABSTRACT

This document contains a discussion addressing the use of a reference architecture as a foundation for a C2IS strategy. The motivation for this discussion is based on the need for flexible command and control information systems that are open for changes as the requirements change, and the need for interoperability between existing and new systems both within the Norwegian Navy and across services. In this context modern system developement orientations and methodologies are described together with technologies such as object– /component technology and middleware.

9) DATE	AUTHORIZED BY This page only	POSITION
15 september 2001	Vidar S. Andersen	Director of Research

ISBN 82-464-0544-6

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE
(when data entered)

INNHOLD

		Side
1	INNLEDNING	9
1.1	Ytre rammefaktorer	9
1.2	Fokusområde og avgrensninger	10
1.3	Oppbygging av rapporten	11
2	SENTRALE BEGREPER OG DEFINISJONER	12
2.1	System	12
2.2	Maritim ledelse	12
2.3	Kommando og kontroll informasjonssystem	12
2.4	Distribuert system	13
3	ARKITEKTURER OG NIVÅER AV ARKITEKTURER	13
3.1	Nivåer av arkitekturer	14
3.1.1	Arkitekturrammeverk	15
3.1.2	Referansearkitektur	17
3.1.3	Systemarkitekturer	17
3.2	Referansearkitektur støtter virksomheten	19
3.3	Systemarkitektur støtter systemutviklingsprosessene	19
3.4	Arkitekturer – den "mest bestandige" delen av et system	20
3.5	Utfordringer	20
4	MOTIVASJON FOR ARKITEKTURER I FORSVARET	20
4.1	Begrensede økonomiske ressurser	22
4.2	Strukturelle endringer	23
4.3	Teknologisk utvikling	24
4.3.1	Teknologiuavhengighet	24
4.3.2	"Legacy systems"	24
4.3.3	"Nye muligheter skaper nye behov"	24
4.4	Operativ interoperabilitet	25
4.4.1	Arkitektur– og interoperabilitetsarbeid i NATO	26
4.4.1.1	NATO Policy for C3 Interoperability	27
4.4.1.2	NIF	27
4.4.1.3	NIE	27
4.4.1.4	"NC3S Architecture Framework"	28
4.5	Mulige strategialternativer ved bruk av arkitekturer	29
4.5.1	Kun NOTS–systemer	30
4.5.2	Nasjonal referansearkitektur	31
5	ARKITEKTURRAMMEVERK	32
5.1	C4ISR–AF	33

5.1.1	Vurdering	34
5.2	RM–ODP	34
5.2.1	Interoperabilitetsnivåer	36
5.2.2	Vurdering	37
5.3	MACCIS	38
5.3.1	Vurdering	39
5.4	Konklusjon arkitekturrammeverk	39
6	REFERANSEARKITEKTURER	40
6.1	OMA	41
6.1.1	C4I DSIG	43
6.2	Konklusjon referansearkitekturer	44
7	PROGRAMVAREARKITEKTURER	45
7.1	Klient–tjener–arkitekturer	45
7.1.1	2–lags arkitektur	46
7.1.2	3–lags arkitektur	46
7.1.3	N–lags arkitektur	47
7.1.4	Sammenlikning	47
7.2	Agentarkitekturer	48
8	ARKITEKTURER PÅVIRKER SYSTEMKVALITETER	50
8.1	Funksjonelle og ikke–funksjonelle krav	50
8.2	Kvaliteter ved et fremtidig K2IS	51
8.2.1	Utviklings– og forvaltningsegenskaper	51
8.2.2	Kjøretidsegenskaper	52
9	KOMPONENTBASERT SYSTEMUTVIKLING	54
9.1	Objektorientert systemutviklingsperspektiv	55
9.2	Komponentbasert systemutviklingsperspektiv	56
9.2.1	"Nivåer" av komponenter	57
9.2.2	Moderne komponentbaserte distribuerte systemer	57
9.2.2.1	Distribuerte objekter	59
9.3	Organisasjonperspektiv i systemutvikling	60
9.4	Systemutviklingsmetodikk	61
9.4.1	Inkrementer og iterasjoner	63
9.4.2	Arkitekturfokus i komponentbasert systemutvikling	63
9.5	"Rational Unified Process"	64
9.6	"Catalysis"	65
10	OBJEKT– OG KOMPONENTTEKNOLOGI	67
10.1	Objektorienterte databaser	67
10.1.1	ODMG 3.0	67
10.1.2	Fordeler ved objektorienterte databaser	68
10.2	Applikasjonstjenere og komponenter	68

10.2.1	J2EE	69
10.2.1.1	Teknologi– og markedsvurdering	70
10.2.2	Windows Distributed interNet Application	70
10.2.2.1	Teknologi– og markedsvurdering	71
10.2.3	CORBA Component Model	72
10.2.3.1	Teknologi– og markedsvurdering	72
11	MELLOMVARETEKNOLOGI	72
11.1	Oversikt over grunnleggende mellomvareteknologier	73
11.1.1	Fjernprosedyrekall	73
11.1.2	Fjerndataaksess	74
11.1.3	Meldingsorientert mellomvare	75
11.1.4	Andre grunnleggende tjenester	76
11.2	Distribuerte transaksjonsmonitorer	77
11.3	ORB mellomvare	77
11.3.1	CORBA	79
11.3.2	DCOM	79
11.3.3	Java RMI	79
11.4	"Message broker"	80
11.5	"Legacy" tilgang/integrasjon mellomvare	80
11.5.1	Skjermskraping	81
11.5.2	Databasebroer	81
11.5.3	Applikasjonsbroer	81
11.5.4	Objektinnpakkere	82
11.5.5	Integrasjonsbroer	83
11.6	Strategi for bruk av mellomvareteknologi	83
12	KONKLUSJON	83
	Litteratur	86

APPENDIKS

A	DEFINISJONER	89
A.1	Norske definisjoner	89
A.2	Engelske definisjoner	90
B	FORKORTELSER OG AKRONYMER	91
	Fordelingsliste	95

ARKITEKTURER FOR KOMMANDO OG KONTROLL INFORMASJONSSYSTEMER
Arkitekturer, komponentbasert systemutvikling og teknologier

1 INNLEDNING

I prosjekt 730 KKI–Sjø har det vært gjennomført innledende analyser av maritime operative informasjonssystemer. Disse analysene har vist at det til nå har vært innført og utviklet operative informasjonssystemer i Forsvaret som ikke er tilstrekkelig interoperable og som ikke har en tilstrekkelig avstemt funksjonalitet. Analysene viser at det er et klart behov for en overordnet strategi i den videre utvikling av disse. Kartleggingen gjort i (21) konkluderer også med at interoperabilitet på høyt nivå mellom informasjonssystemer krever mer helhetlige strategier. En stor utfordring fremover vil være å sikre at eksisterende og planlagte systemer vil ha tilstrekkelig interoperabilitet og er i stand til å samvirke på det nivå som kreves for å kunne utøve effektiv kommando og kontroll.

Målsettingen med dette arbeidet har derfor vært å:

- formidle den sentrale rollen som bruk av arkitekturer har som et viktig fundament i en *overordnet strategi* for kommando og kontroll informasjonssystemer (K2IS).
- vurdere og skissere valg av arkitekturer, metodikker og teknologier for utvikling og vedlikehold av et fremtidig K2IS basert på en visjon om at dagens "system av systemer" over tid går over til ett fremtidig distribuert, komponentbasert og tjenesteintegrert informasjonssystem – d v s et sømløst og integrert K2IS.
- etablere et beslutningsgrunnlag for de valg prosjektet gjør angående informasjonssystemer i anbefalt system (40), og valg av metoder og teknologi i f m utvikling av bildeoppbyggingsdemonstratoren i prosjektet.

Resultatet fra dette arbeidet dokumenteres i denne rapporten.

1.1 Ytre rammefaktorer

Et sentralt trekk i den generelle teknologiske utviklingen er fremveksten av distribuerte systemer. Særlig innen forskning og utvikling av komplekse informasjonssystemer settes nå fokus på distribuerte løsninger som skal støtte grupper av mennesker og tilhørende virksomhetsprosesser. Innenfor organisasjon og ledelse introduserer denne utviklingen et viktig skifte i perspektiv på utvikling og bruk av teknologi i organisasjoner. Organisasjonsutvikling og systemutvikling må ses i sammenheng, og bør betraktes som to sider av samme sak.

Et meget sentralt trekk ved denne teknologiske utviklingen er betydningen av å kunne håndtere *heterogenitet*. Dette gjelder både på virksomhetsnivå (mennesker og arbeidsprosesser) og på system– og teknologinivå. Det går bl a på heterogenitet i representasjon av

informasjonsinnhold, i realisering, teknisk infrastruktur (f eks forskjellige kommunikasjonsnett med ulik kapasitet) og i ulike behov for brukertjenester. En stor utfordring fremover vil være å håndtere integrasjon av nye produkter med eksisterende systemer, og at generasjoner av teknologier må kunne spille sammen. Innenfor moderne systemutvikling vil derfor programvareplattformer, d v s operativsystem, mellomvare og grunnleggende generiske tjenester (f eks mot databaser), spille en sentral rolle i utviklingen av fremtidige distribuerte informasjonssystemer. Denne utviklingen krever mer overordnete og helhetlige strategier i forhold til systemutviklingen av informasjonssystemer i fremtiden.

Disse trekkene i den teknologiske utviklingen vil berøre Forsvaret i sterk grad og vil bidra til en økning i kompleksiteten i de operative informasjonssystemene fremover. Graden av kompleksitet i systemene vil ytterligere økes ved stadig økende krav til funksjonalitet tilpasset brukernes behov i tillegg til at brukerne vil ha løsninger stadig raskere og med høy kvalitet. Dette vil kreve økt helhetstenkning rundt utvikling og vedlikehold av K2ISer.

I fremtiden vil en i Forsvaret, med stor sannsynlighet, ha flere parallelle utviklings– og vedlikeholdsprosjekter med varierende grad av overlappende målsettinger og behov for informasjonsutveksling. Disse prosjektene vil Sjøforsvaret ha varierende grad av innflytelse på, avhengig om det er nasjonale prosjekt eller for eksempel NATO–prosjekt. Systemene som utvikles vil ha krav til kontinuerlige modifikasjoner i sin levetid. Dette er en problemstilling som Forsvaret må ha en overordnet strategi for, fordi den økte kompleksiteten vil være vanskelig å håndtere hvis man fortsetter dagens måte å utvikle systemer på.

Fremtidens K2IS vil være et geografisk distribuert system. Kommunikasjonsbåndbredden i systemet vil være særdeles variabel og vil være avhengig av egenskaper ved geografien, hvilke sambandsressurser som er tilgjengelig og hvilke krav det stilles i f m å unngå eksponering. Videre vil deler av systemet kunne settes ut av funksjon for eksempel som en konsekvens av fiendtlige handlinger. I slike sammenhenger vil det være et krav at ytelsen til systemet degraderes gradvis og at systemet ikke kollapser. K2ISer vil også ha spesielle karakteristikker i forhold til sivile systemer, som f eks krav til sikkerhetsarkitektur. Sentrale egenskaper for K2ISer er identifisert i prosjektets konseptarbeider og oppsummert i kapittel 8.2.

I NATO er det sterk fokus på arkitekturer og operativ interoperabilitet. Det arbeides mye med standardisering på arkitekturer, bl a med tanke på å oppnå interoperabilitet mellom systemer. En kort summarisk beskrivelse av dette arbeidet er gitt i kapittel 4.4. For at allierte lands nasjonale systemer skal kunne benyttes i fremtidige NATO–operasjoner, vil disse etterhvert måtte følge arkitekturstandarder. Visjonen for NATOs "Information Management Strategy" er blant annet at kommunikasjonsbærerene som benyttes av et hvilket som helst nettverk eller bruker vil være transparent for brukeren. Videre er visjonen at grensene mellom dagens systemer vil forsvinne og målet om et fullt integrert system eller system av systemer vil bli realisert.

1.2 Fokusområde og avgrensninger

I tidligere kommando, kontroll og informasjon (KKI) –relaterte prosjekter har fokuset på K2ISer hovedsakelig vært avgrenset til problemstillinger tilknyttet krav til informasjons-

innhold og funksjonalitet. Dette er viktige aspekter og adresseres i f m bildeoppbyggingsaktiviteten i prosjektet. I denne rapporten har vi imidlertid valgt å gå utover diskusjon på systemnivå og heller fokusert på helheten. I et helhetlig perspektiv er aspekter som nyutvikling, videreutvikling, vedlikehold, innkjøp og drift (forvaltning) sentrale. Teknologi, organisasjon og systemutviklingsprosesser er også sentrale aspekter. Ved å se på helheten ser vi på *strategier* og *teknologier* for fremtidig systemutvikling og forvaltning av K2ISer.

Ved å innta et helhetlig perspektiv kan oppgaven fort bli stor og omfattende. Tiden til rådighet for denne aktiviteten har derfor bidratt til at vi har prioritert helhet i forhold til det å gå i dybden på enkelttemaer.

Et annet problem vi har støtt på under arbeidet er at utvikling og bruk av overordnete arkitekturer som strategi i forhold til informasjonssystemer (IS), fortsatt er et relativt umodent fagområde innenfor system– og organisasjonsutvikling. Gode eksempler på overordnete arkitekturer og anvendelser og bruk av disse har tildels vært vanskelig å finne bl a fordi fagområdet ennå ikke er så godt dokumentert. Imidlertid er interessen for dette fagområdet meget stor. Spesielt innenfor IS–forskning, men også innenfor system– og organisasjonsutviklingsmiljøer generelt. Dette gjelder både innenfor det sivile markedet og i militær sammenheng.

1.3 Oppbygging av rapporten

Dokumentet er delt inn i to deler. Den første delen argumenterer for bruk av arkitekturer som en strategi for utvikling og vedlikehold av K2ISer og kan leses av alle. Den andre delen retter seg inn på lesere som har kunnskaper og interesse for konkrete teknologier og arkitekturer, og som er fortrolig med en mer teknologisk fagterminologi.

1. I kapittel 2 gis først noen definisjoner av sentrale begreper slik de blir brukt i denne rapporten. I kapittel 3 introduseres begrepet arkitektur og arkitekturnivåer. Dette er et sentralt kapittel for å forstå hva prosjektet legger i begrepet arkitektur. Kapittelet anbefales å leses av alle. Kapitlene 4, 5 og 6 beskriver og argumenterer for bruk av arkitekturer som en strategi for utvikling og vedlikehold av K2ISer. Kapittel 4 utdyper noen motivasjonsfaktorer for bruk av arkitekturer i Forsvaret. Kapittel 5 og 6 gir eksempler på eksisterende arkitekturrammeverk og referansearkitekturer.

2. I kapitlene 7, 8, 9, 10 og 11 beskrives programvarearkitekturer, krav til systemkvaliteter, metodikker og teknologier i moderne systemutvikling. Moderne systemutvikling fokuserer på *bygging* av åpne distribuerte systemer. Objekt–, agent– og komponentorienterte teknikker og metoder står sentralt i denne sammenheng. I kapittel 7 beskrives noen generelle programvarearkitekturer. Kapittel 8 beskriver hvordan krav til systemkvaliteter påvirker valg av arkitekturer. Aspekter ved systemutvikling for distribuerte systemer beskrives i kapittel 9. Kapitlene 10 og 11 beskriver objektorienterte og komponentbaserte teknologier, samt teknologiarkitekturer og standarder.

Rapporten avsluttes med en konklusjon i kapittel 12.

2 SENTRALE BEGREPER OG DEFINISJONER

I dette kapitelet introduserer vi noen sentrale begreper og definisjoner slik de blir brukt i denne rapporten. Begrepet arkitektur utdypes nærmere i kapittel 3.

2.1 System

Fra et systemperspektiv er en definisjon av et system, slik det er bruk i denne rapporten, gitt som (12):

Et system er en del av verden som en person (eller gruppe) – i en viss periode og av en eller annen årsak – velger å betrakte som en helhet bestående av visse komponenter, som hver er karakterisert ved visse egenskaper som er utvalgt som relevante og visse handlinger relatert til disse egenskaper og til andre komponenters egenskaper.

I NATO er et system definert på følgende måte (2)(3):

Samling av doktriner, metoder, personell, prosedyrer, utstyr og fasiliteter for å utføre en bestemt funksjon.

2.2 Maritim ledelse

I konteksten kommando og kontroll brukes begrepet system for å beskrive ressursene som er nødvendige for å støtte ledelsesfunksjoner. Med *kommando og kontroll* (K2) generelt forstås den myndighet, det ansvar og de aktiviteter som utøves av militære sjefer ved ledelse og koordinering av militære styrker, og ved implementering relatert til iverksettelse av operasjoner (2).

Et ledelsessystem, slik det er definert i dette prosjektet, er et system som utfører ledelsesfunksjoner. Et maritimt kommando og kontroll system er et ledelsessystem for maritime operasjoner (6). I denne rapporten brukes begrepet ledelsessystem synonymt med maritim kommando og kontrollsystem.

2.3 Kommando og kontroll informasjonssystem

I NATO defineres et informasjonssystem som (3):

Samling av utstyr, metoder og prosedyrer, og hvis nødvendig personell, organisert for å utføre informasjonsbehandlingsfunksjoner.

I et informasjonssystem kan det finnes ett eller flere programvaresystemer som brukes for automatisert prosessering av innsamlet data/informasjon og for å generere (syntetisere) ny informasjon:

Et programvaresystem er et system som mottar data, og som bearbeider data til andre data (informasjon), distribuerer og lagrer data/informasjon, og støtter opp under tolkning, beslutningstaking og planlegging.

Informasjon er kunnskap om objekter f eks fakta, prosesser eller ideer, inklusive begreper som i en viss sammenheng har en spesiell mening (1).

K2IS betraktes som en del av et ledelsessystem. Med et K2IS menes et databasert informasjonssystem som understøtter militære sjefer i utøvelsen av kommando og kontroll. I denne rapporten brukes begrepet *informasjonssystem* synonymt med K2IS definert ved (2):

Et informasjonssystem som gir militære myndigheter støtte i forbindelse med kommando og kontroll.

Informasjonssystemene vil være meget sentrale i ledelsessystemet fordi de skal understøtte militær ledelse. Et eksempel på et funksjonsområde i informasjonssystemet er bildeoppbygging. Dette funksjonsområdet kan realiseres som en distribuert bildeoppbyggingsapplikasjon. En *distribuert applikasjon* er en applikasjon hvor deler av applikasjonen befinner seg på forskjellige maskiner i et nettverk. Informasjonssystemet bruker sensor– og sambandssystemet for å samle inn og distribuere data og informasjon som distribuerte applikasjoner kan utnytte.

2.4 Distribuert system

En definisjon av et distribuert system er gitt i (10):

"A system that runs on a collection of machines that do not have shared memory, yet looks to its users like a single computer."

En bruker av et distribuert system kan være en applikasjon, en person eller gruppe av personer.

3 ARKITEKTURER OG NIVÅER AV ARKITEKTURER

Arkitekturbegrepet kan ha forskjellige betydninger avhengig av sammenhengen det brukes i, noe som ofte fører til misforståelser. I forhold til informasjonssystemer er utvikling og bruk av overordnede arkitekturer fortsatt et relativt umodent fagområde, med en umoden terminologi. Det kan derfor være vanskelig å gi en presis definisjon av begrepet. Formulert litt løselig kan en si at en arkitektur beskriver de mest "bestandige" og sentrale aspektene ved et system. Sagt på en annen måte: en arkitektur sier noe om hvordan et system er satt sammen (struktur), eller angi rammer for hvordan det *kan* settes sammen.

Den internasjonale standarden IEEE STD 610.12 (15) definerer en arkitektur som:

"Architecture has various meanings, depending upon its contextual usage: (a) The structure of components, their interrelationship, and the principles and guidelines governing their design and evolution over time. (b) Organizational structure of a system or component."

For et distribuert informasjonssystem kan man, med utgangspunkt i definisjonen ovenfor, generelt si at en arkitektur beskriver *overordnete prinsipper* og *retningslinjer* for hvordan

komponenter i systemet skal designes og utvikles over tid. Det gjøres ofte igjennom et sett av regler og standarder for hvordan man skal bygge opp programvaresystemer. En arkitektur i denne sammenheng fokuserer spesielt på infrastruktur og grensesnitt mellom de ulike komponentene i systemet. Denne måten å definere arkitekturer på er imidlertid noe snever, og prosjektet ønsker å fremheve i det følgende at arkitekturer kan være av en mer overordnet type. En arkitektur kan også inkludere virksomheten.

Frem til nå har hovedfokus vært på programvarearkitekturer – d v s implementasjons–arkitekturer som delsystemer kan bygges over. F eks er Maritime Command and Control Information System (MCCIS) et system bygget over en klient–tjener arkitektur. Det som det har vært mindre fokus på, er:

h*vordan potensialet i anvendelsen av arkitekturer kan utnyttes på et overordnet nivå.*

I denne sammenheng vil arkitekturen fungere som en mer teknologiuavhengig "referanseramme" for alle systemer eller delsystemer som skal utvikles innenfor et bestemt virksomhetsområde, og som et "verktøy" eller en "strategi" for å samordne og integrere teknologi, arbeidsoppgaver og arbeidsprosesser tettere sammen. På dette nivået er ikke systemdetaljer interessante.

Dette vil si at en arkitektur også kan brukes til å definere de overordnete retningslinjene for hvilke operative behov som skal dekkes, hvordan systemer skal bygges og passes inn i en bestemt omgivelse. En arkitektur fungerer da som en overordnet felles *referanseramme* for *alle* systemer eller delsystemer som skal anskaffes eller skal designes og utvikles innenfor et spesifikt virksomhetsområde. Et virksomhetsområde kan *velges* å være en hel organisasjons virksomhetsområde, eller bare deler av en organisasjons virksomhetsområde (f eks kommando og kontroll virksomheten). Eller til og med bare være et bestemt teknologiområde av interesse (f eks Reference Model for Open Distributed Processing (RM–ODP) innenfor distribuert prosessering[1]).

Bruk av arkitekturer på "flere nivåer" kan foreslås som en strategi som støtter et helhetlig perspektiv i forholdet teknologi, arbeidsoppgaver og organisasjon.

I delkapittel 3.1 utdypes arkitekturer på flere nivåer litt nærmere.

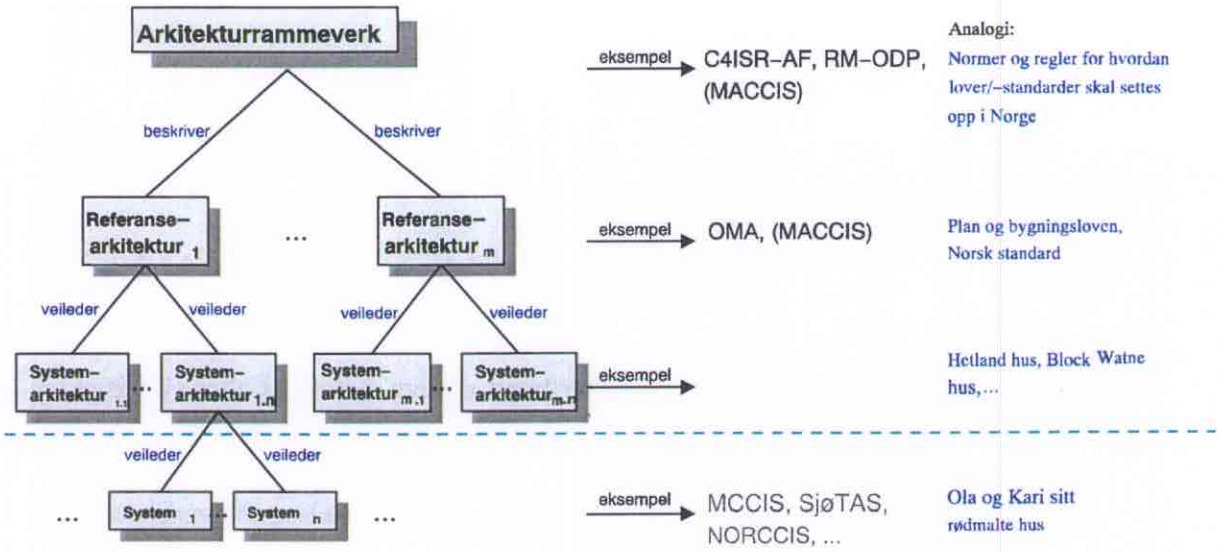
3.1 Nivåer av arkitekturer

I prosjektet har vi valgt å klassifisere arkitekturbegrepet etter hvilket abstraksjonsnivå de omhandler. Dette må ikke forstås som en hierarkisk inndeling, men mer som en illustrasjon av relasjoner mellom arkitekturer og systemer. Følgende klassifisering gjøres:

1. *Arkitekturrammeverk*
2. *Referansearkitekturer*

1. RM–ODP er et arkitekturrammeverk og beskrives nærmere i kapittel 5

Figur 3.1 gir en prinsipiell fremstilling av denne klassifiseringen. En analogi til husbygging er brukt som en hjelp i beskrivelsen av hvilken abstraksjongrad som omfattes av de ulike nivåene.

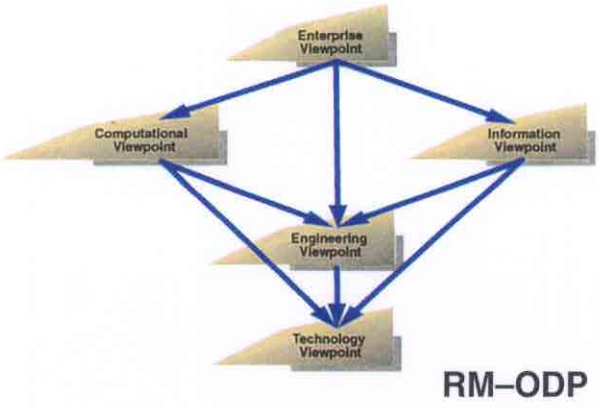


Figur 3.1 Nivåer av arkitekturer

3.1.1 Arkitekturrammeverk

Et arkitekturrammeverk (refereres også ofte til som spesifikasjonsmodeller) gir oss standardiserte måter å beskrive arkitekturer på. Et arkitekturrammeverk angir retningslinjer for *hvordan beskrive* spesifikke arkitekturer, men gir ingen retningslinjer for hvordan designe eller implementere konkrete arkitekturer. Et arkitekturrammeverk dekker *ett bestemt område* av interesse og brukes til å veilede utarbeidelse av f eks referansearkitekturer eller systemarkitekturer. Mer konkret så vil dette si at et arkitekturrammeverk definerer *hvordan* man skal gå frem for å beskrive en spesifikk arkitektur, samt *hva* som skal beskrives. Når modellene foreskrevet i arkitekturrammeverket er fylt med innhold, er *resultatet* en referansearkitektur eller en systemarkitektur (avhengig av detaljeringsgrad). Disse vil igjen vil være basis for utviklingen av konkrete systemer innenfor det bestemte området.

Modellene angitt i arkitekturrammeverket brukes for å spesifisere forskjellige aspekter ved et system. Rammeverket angir *hva* som inngår i disse modellene og *hvordan* disse modellene skal beskrives. Modellbeskrivelsene er lagt inn under såkalte "viewpoints" eller det vi i denne rapporten har valgt å kalle for *arkitekturperspektiver.* Rammeverket definerer *hvilke* arkitekturperspektiver som skal beskrives, *hva* som inngår i disse arkitekturperspektivene og *hvordan* disse skal beskrives. Hvert arkitekturperspektiv representerer en abstraksjon av hele systemet og fokuserer på et spesifikt aspekt, f eks informasjonsaspekt ("Information viewpoint"), teknologiaspekt ("Technology viewpoint"), organisasjonsaspekt ("Enterprise viewpoint"), o s v (se figur 3.2). Arkitekturperspektivene vil til sammen omfatte hele systemet. Hensikten med arkitekturperspektiver er å "zoome" inn på de problemene som studeres og ignorere irrelevante ting.



Figur 3.2 Et arkitekturrammeverk beskriver de modeller som skal brukes for å spesifisere forskjellige aspekter ved et system.

For å kunne håndtere kompleksiteten av et distribuert system, betraktes altså systemet fra ulike perspektiver. Da kan hver "bruker" eller "viewer" bare konsentrere seg om sin del av verden, og dette brukersynet beskrives separat. F eks vil en typisk "bruker" eller "viewer" til organisasjonsperspektivet ("enterprise") være organisasjonsledelsen. Hovedhensikten med dette perspektivet er å redegjøre for og begrunne rollen til et informasjonssystem som brukes av en eller flere organisasjoner. En "enterprise"–modell sier noe om hvordan man skal beskrive f eks:

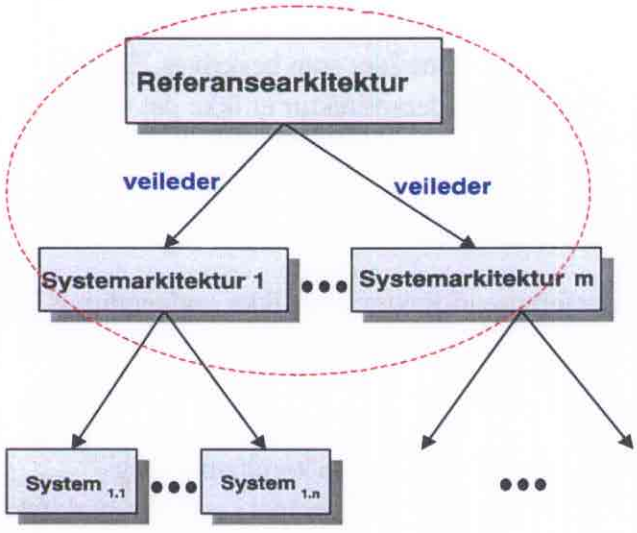
- roller og aktiviteter innen organisasjonen(e) som benytter systemet
- organisasjonsstrukturen til den eller de organisasjoner som benytter systemet
- forskjellige interaksjoner mellom systemet og dets omgivelser
- hvilke typer organisasjonsinformasjon som er tilgjengelig for hver av brukerrollene
- hvor i organisasjonen ulike typer prosseseringer kan finne sted, samt sikkerhet og "management" gjeldende organisasjonen, o s v.

En systemanalytiker vil være en typisk "bruker" eller "viewer" til informasjons–perspektivet. Hensikten med informasjonsperspektivet er informasjonsmodellering, hvor målet er å etablere et konsistent og felles syn på ulike informasjonskilder og informasjonsflyt.

Et arkitekturrammeverk kan altså i stort sies å være en slags "metodikk" med et begrepsapparat for hvordan man kan beskrive andre og mer spesifikke arkitekturer innenfor et avgrenset område. Hvordan dette skal gjøres varierer med hensikten med systemet og hvilket behov det skal dekke. Retningslinjer for å opprettholde konsistens i arkitekturen, d v s relaterbarhet mellom arkitekturperspektivene, skal også angis i arkitekturrammeverket. Arkitekturer som er beskrevet ut fra det samme rammeverk vil være sammenlignbare. Eksempler på eksisterende arkitekturrammeverk gis i kapittel 5.

3.1.2 Referansearkitektur

En referansearkitektur fungerer som en "referansemodell" for et bestemt område og utvikles som regel gjennom veiledning fra et arkitekturrammeverk. En referansearkitektur blir brukt som basis for utvikling av systemarkitekturer, som igjen vil være basis for utviklingen av konkrete systemer (figur 3.3).



Figur 3.3 *En systemarkitektur gjenbruker elementene fra referansearkitekturen og inneholder i tillegg det som er spesifikt for det bestemte systemområdet som skal beskrives.*

I en referansearkitektur nedfelles overordnete strategier og generelle krav, samt aktører og roller som gjelder i det gitte virksomhetsområdet. Virksomhetsområdets sentrale funk–sjonsområder som skal dekkes, beskrives også. Utover dette beskrives detaljerte spesifikasjoner for generell funksjonalitet (for gjenbruk), sentrale data– og utvekslingsmodeller og arbeidsmetodikk for hvordan man lager systemarkitekturer. Arbeidsmetodikken for å lage en systemarkitektur beskriver hvilke aspekter ved systemene som skal spesifiseres, hvordan spesifikasjonene skal realiseres, hvilken notasjon som skal brukes, o s v.

En referansearkitektur bør ikke være for detaljert i sine beskrivelser. En referansearkitektur skal gi organisasjonen fleksibilitet – ikke redusere den. En referansearkitektur skal være "bindeleddet" mellom systemene innenfor et gitt virksomhetsområde. Et sentralt mål er at en godt utformet referansearkitektur skal danne fundamentet for en fleksibilitet i og rundt utvikling og vedlikehold av komplekse distribuerte informasjonssystemer.

Eksempler på eksisterende referansearkitekturer gis i kapittel 6.

3.1.3 Systemarkitekturer

Systemarkitekturer er en spesifikk og mer konkret arkitektur for design og implementasjon av systemer eller delsystemer innenfor et bestemt systemområde. I denne sammenheng er

det viktig å fremheve at en systemarkitektur, slik den defineres i dette dokumentet, ikke er det samme som en programvarearkitektur. En programvarearkitektur er en mer konkret implementasjonsarkitektur som delsystemer eller applikasjoner kan bygges over. Eksempler på programvarearkitekturer er typisk klient–tjener arkitekturer og agent–arkitekturer[2]. En systemarkitektur, derimot, inneholder ofte beskrivelsen av *ett eller flere* programvaresystemer med sine egne spesifikke programvarearkitekturer.

En systemarkitektur gjenbruker elementene fra referansearkitekturen og inneholder i tillegg krav til det som er spesifikt for det bestemt systemområdet som beskrives. En systemarkitektur består også ofte av flere delarkitekturer. En delarkitektur er ikke det samme som programvarearkitektur. Et eksempel på en delarkitektur kan være en sikkerhetsarkitektur som alle programvaresystemene i informasjonssystemet må følge. Man kan også snakke om spesifikke delarkitekturer innenfor et bestemt programvaresystem.

En systemarkitektur kan beskrives ut fra en referansearkitektur, men ikke nødvendigvis. En kan også beskrive systemarkitekturer direkte ut fra et arkitekturrammeverk. System–området i en systemarkitektur trenger nødvendigvis ikke reflektere hele virksomhets–området i referansearkitekturen (eller arkitekturrammeverket). Eksempelvis dersom virksomhetsområdet i en referansearkitektur er valgt til å omfatte hele kommando– og kontrollorganisasjonen i det norske forsvar, så kan f eks systemområdet i en systemarkitektur være avgrenset til kun å omfatte den maritime delen.

En systemarkitektur *stiller krav* til systemet mhp hva systemet skal gjøre (funksjonalitet), hvordan systemet skal forholde seg til omgivelsene og hvordan de ulike aspektene ved systemet skal realiseres, d v s hvordan man kan realisere de ulike *perspektivene* som til sammen spesifiserer hele systemet (figur 3.4).

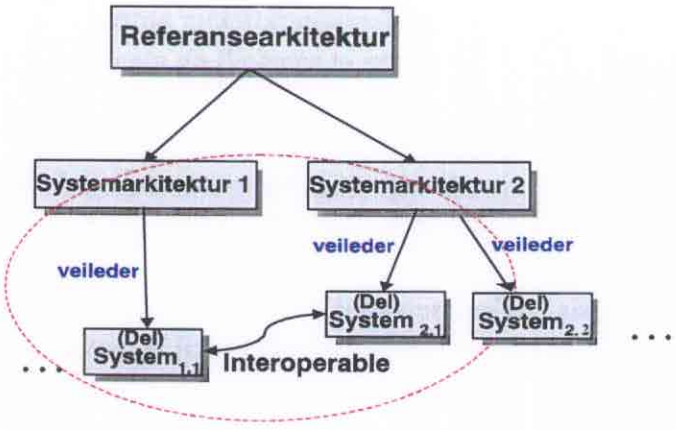
Figur 3.4 Et informasjonssystem kan sees ifra flere perspektiver som til sammen spesifiserer hele systemet.

I tillegg spesifiseres hvilke ferdige komponenter som skal benyttes og hva som skal nyutvikles. Systemarkitekturer fundert på samme referansearkitektur gir systemer som:

- er i h t overordnede strategier og krav
- fyller definerte funksjoner i en større sammenheng
- gjenbruker ferdige spesifikasjoner og komponenter

2. Programvarearkitekturer blir nærmere beskrevet i kapittel 7.

- baserer seg på de samme informasjonsstrukturer
- er interoperable
- er lettere å administrere og vedlikeholde.



Figur 3.5 Ved å referere til samme sett av abstraksjoner kan systemspesifikasjoner lettere kobles og utviklere kan lettere forstå spesifikasjoner av andre systemer.

3.2 Referansearkitektur støtter virksomheten

En referansearkitektur ivaretar og støtter virksomhetens behov. Det gjør den ved at utvikling av komplekse informasjonssystemer kan beskrives og forankres på en overordnet måte som gir føringer for mer enn *ett* system eller applikasjon. Det muliggjør en skrittvis utvikling mot et sett av informasjonssystemer eller applikasjoner, som spiller sammen og dekker virksomhetens totale behov. På denne måten vil en referansearkitektur kunne være et *fundament i en overordnet og helhetlig K2IS–strategi.*

Når vi videre i rapporten snakker om bruk av arkitekturer mener vi utvikling og bruk av en *overordnet referansearkitektur* som vil utgjøre *fundamentet* i en fremtidig K2IS–strategi. En eller flere systemarkitekturer er da fundert på den samme referansearkitektur. På denne måten snakker vi om *arkitekturer på flere nivåer.* Virksomhetsområdet av interesse i denne sammenheng er avgrenset til kommando og kontroll. Fra et teknologiperspektiv er hovedmålet systeminteroperabilitet. Fra et operativt virksomhetsperspektiv er målet å oppnå et koordinert samspill mellom teknologiutvikling, operativ virksomhetsutvikling og systemutviklingsprosessene.

3.3 Systemarkitektur støtter systemutviklingsprosessene

Systemarkitekturen støtter systemutviklingsprosessene ved at uviklingen av distribuerte informasjonssystemer fordrer en systemarkitektur som forankrer systemløsninger på et overordnet nivå i organisasjonen. Dette kan realiseres gjennom at systemarkitekturen funderes på en felles overordnet referansearkitektur.

3.4 Arkitekturer – den "mest bestandige" delen av et system

En referansearkitektur vil representere den mest "bestandige" delen av et system fordi en referansearkitektur skiller ut de mer "bestandige" delene (f eks "Enterprise", "Information" og "Computational") fra de mer teknologiavhengige delene ("Engineering" og "Technology"). En vil da lettere kunne gjennomføre skifte i valg av underliggende teknologier. Imidlertid er det viktig å fremheve at selv når en referansearkitektur anses som ferdig utviklet, er den ikke fastlagt for alltid. Det vil alltid være et samspill og gjensidig påvirkning mellom teknologiutvikling og operativ virksomhet som jevnlig medfører endringer i arkitekturen. Endringsprosesser i den operative virksomheten vil f eks kunne resultere i nye eller endrede krav til den underliggende teknologien, og endringer i den underliggende teknologien vil kunne gi nye muligheter for endringer i den operative virksomheten. Likevel, en referansearkitektur vil endres saktere enn selve teknologien og de aktuelle systemene som er utviklet med basis i referansearkitekturen. En referansearkitektur vil da fremstå som den mest teknologiuavhengig beskrivelsen, og en vil lettere kunne gjennomføre et skifte i underliggende teknologier. Arkitekturer vil på den måten representere den "mest bestandige" delen av et system.

3.5 Utfordringer

Det som imidlertid er et problem, er at det er utviklet få arkitekturrammeverk, referansearkitekturer og systemarkitekturer. Det betyr at vi ennå ikke har så godt erfaringsmateriale. Få systemer er utviklet med basis i f eks en referansearkitektur. Potensialet anses imidlertid som betydelig.

4 MOTIVASJON FOR ARKITEKTURER I FORSVARET

Effektiv utnyttelse av informasjonsteknologi er et viktig område for Forsvaret fremover (19). Den teknologiske utviklingen åpner for nye muligheter for raskere informasjonsflyt og beslutningstaking, men gir også store nye utfordringer. F eks vil utviklingen mot åpne distribuerte systemer, sammen med økt krav til funksjonalitet gi en betydelig økning i kompleksiteten i de operative informasjonssystemene i fremtiden. Denne økte kompleksiteten vil være vanskelig å håndtere hvis man fortsetter dagens måte å utvikle systemer på. Uten en mer samordnet og helhetlig strategi i forhold til systemutviklingsprosessene, ser man for seg et lite egnet K2IS til å understøtte en fremtidig kommando og kontroll organisasjon.

Som nevnt innledningsvis har informasjonssystemer for kommando og kontroll til nå ofte vært utviklet isolert, med lukkede tekniske og proprietære løsninger uten særlig tanke på å skulle kunne samvirke med andre systemer. Resultatet har blitt et vidt spekter av forskjellige "enkeltstående" systemer som ikke utveksler informasjon med andre systemer. Systemene har til dels ofte overlappende funksjonalitet og informasjonsinnhold, og er vanskeli-

ge å vedlikeholde[3]. Dette blir svært kostbart og etterhvert ganske uhåndterbart. Behovet for et mer helhetlig perspektiv i synet på hvordan man anskaffer og utvikler operative informasjonssystemer fremover, er derfor blitt svært stort. For å imøtekomme denne problematikken, fokuseres det på arkitekturer som en strategi for bedre å samordne og styre utviklingen av distribuerte informasjonssystemer opp mot operative krav.

Analyser i (21) viser at Forsvaret mangler en overordnet arkitektur som omfatter en modell av den operative virksomheten. (21) konkluderer med at en slik modell bør ligge til grunn for all systemutvikling i Forsvaret. I KKI–Sjø–prosjektet ser vi på hvilke muligheter bruk av en overordnet referansearkitektur og systemarkitekturer fundert på denne (som strategi), kan gi for å støtte et mer helhetlig perspektiv i forholdet teknologi, systemutvikling og organisasjon[4]. Gjennom en slik strategi vil utviklingen av informasjonssystemer struktureres inn i en sammenheng med felles virksomhetsmodeller, retningslinjer, krav og standarder som *forankrer systemutviklingsprosessene tettere opp til virksomhetssprosessene* i ledelsessystemet. Det blir da virksomhetsmodellene, gjeldende krav og vedtatte standarder som setter føringene for systemene og ikke selve teknologien.

Arkitekturer brukt på denne måten mener vi kan danne fundamentet i en overordnet og helhetlig K2IS–strategi. Målsettingen med dette vil for Forsvaret være å:

- samordne utviklingen av (del)systemer mot operative krav
- samordne (harmonisere) nasjonale systemer mot NATO–systemer
- gjøre det enklere å oppnå interoperabilitet mellom systemer, samt støtte organisasjons-interoperabilitet – det som ofte refereres til som operativ interoperabilitet (beskrives nærmere i kapittel 4.4)
- håndtere økt kompleksitet relatert til introduksjon av distribusjonsteknologi, hurtigere endringer i krav, nye krav, krav til kortere utviklingstider og krav til løsninger med høy kvalitet
- ha bedre oversikt, styring og kontroll
- oppnå kosteffektiv utvikling og forvaltning (muliggjør økt gjenbruk av delsystemer og arkitekturen)
- oppnå en bedre utforming av informasjonssystemer for å støtte ønskede egenskaper ved ledelsessystemet, f eks robusthet, fleksibilitet, hurtighet, o s v
- få en felles mal for teknologistandarder og spesifikasjoner. Et stort problem er at K2ISer i Forsvaret i dag ikke utvikles på grunnlag av felles utvekslingmodeller. Kartleggingen gjort i (21) viser at det er stort samsvar mellom alle forsvarsgrenene i de krav som spesifiserer informasjonsinnholdet i de forskjellige systemene. Vi mener at bruk av en overordnet referansearkitektur vil kunne sikre at alle systemer utvikles basert på de samme modellene

3. I denne sammenheng inkluderer vedlikehold også videreutvikling av funksjonalitet.
4. Her i betydningen av kommando og kontroll organisasjonen.

- lette integrasjon av Commercial Off The Shelf (COTS), Nato Off The Shelf (NOTS), Military Off The Shelf (MOTS), Government Off The Shelf (GOTS) og egenutvikling, samtidig som man ivaretar "systemarven"
- støtte evolusjon i systemer og virksomhet
- ha muligheten for at våre nasjonale K2IS–arkitekturer kan inngå som en del av en felles NATO–arkitektur (hvis nødvendig)
- være bidragsyter og påvirke arkitekturarbeidet som foregår i NATO.

Andre forhold som taler for bruk av arkitekturer er:

- å sikre at prioriterte egenskaper (kvaliteter) ved systemet blir håndtert og ivaretatt
- å få en bevisstgjøring om de valg som tas og konsekvensene av disse
- å forbedre kommunikasjonen mellom bruker og ekspert
- å tilrettelegge for validering, verifisering, simuleringer og testing av systemer før de utvikles
- at arkitekturer er egnet som en basis for kontrakter og avtaler (som et tillegg til kravspesifikasjoner).

Konkrete årsaksforhold som motiverer for bruk av arkitekturer i Forsvaret er:

- begrensede økonomiske ressurser,
- strukturelle endringer (nasjonalt og i NATO),
- teknologisk utvikling, og
- økt fokus på operativ interoperabilitet (nasjonalt og i NATO) (4).

Disse årsaksforholdene er diskutert nedenfor.

4.1 Begrensede økonomiske ressurser

Det er et krav at fremtidens K2IS må kunne utvikles og forvaltes på en mer kostnadseffektiv og forutsigbar måte enn i dag.

Utvikling av distribuerte applikasjoner og tjenester er komplekst samtidig som krav til endringer i funksjonalitet og teknologi øker raskt. Raskere endringer i krav gir økende press mot kortere utviklingstider og lavere kostnader, spesielt i forhold til vedlikehold, samtidig som brukerkravene skjerpes mht funksjonalitet og kvalitet. Erfaringsmessig vil 2/3 av kostnadene til et informasjons–/programvaresystem gå med til vedlikehold, d v s utgifter etter første leveranse. En stor prosentandel av disse vedlikeholdskostnadene er i dag ofte knyttet til integrasjonsarbeid av eksisterende systemer.

Kravet om lavere vedlikeholdskostnader, kortere utviklingstider og økt kvalitet kan bl a oppfylles gjennom økt gjenbruk i form av mer standardiserte og dels distribuerte komponenter. Bruk av arkitekturer vil gi en felles norm for å beskrive systemer slik at de er relaterbare og samtidig lette identifisering av komponenter som kan gjenbrukes. Man vil f eks lettere kunne relatere systemer til hverandre i en anbudssammenheng dersom de er beskrevet på en standardisert måte, noe som vil kunne bidra til økt leverandøruavhengighet. Kostnader forbundet med overlapp av funksjonalitet og informasjonsinnhold vil også kunne reduseres gjennom en samordning av systemutviklingsprosessene.

4.2 Strukturelle endringer

I NATO er fokus på bruk av arkitekturer etterhvert blitt nærmest et krav for å imøtekomme fremtidige utfordringer, hvor man må kunne håndtere økt usikkerhet i forhold til strukturelle endringer ("Combined", "Joint and Multinational Military Operations", "Service", o s v). Økt fleksibilitet er et gjennomgående nøkkelord, og kan i denne sammenheng bety å unngå at informasjonssystemet blir en flaskehals for endringer i organisasjonen. Dette forutsetter at det legges til grunn arkitekturer som går utover det teknologiske perspektivet. Med det menes at en overordnet arkitektur bør inneholde sentrale virksomhetsbeskrivelser som vil være styrende for utviklingen av systemer eller delsystemer i et større perspektiv.

For å få etablert en slik arkitektur i Forsvaret, ligger mange av de nye utfordringene på de operative myndigheter. Ny informasjonsteknologi er mer dynamisk og Forsvaret bør innrette seg slik at man faktisk blir i stand til å utnytte de muligheter distribusjonsteknologien gir til å støtte militære operasjoner. Det betyr at de operative myndigheter har et ansvar for at den riktige operative kompetansen tilgjengeliggjøres. Det må altså settes korrekte og relevante krav til utforming av arkitekturene i henhold til operative behov. Dette forutsetter engasjement og forankring på høyt operativt myndighetsnivå, og er helt avgjørende dersom man på sikt ønsker å forme og utvikle et distribuert og komponentbasert K2IS, som skal evne å underbygge og støtte opp under strukturelle aspekter som f eks krav til fleksibilitet og hurtighet i et ledelsessystem. I (20) vektlegges både en menneskelig og en strukturmessig fleksibilitet. Den menneskelige fleksibiliteten relaterer seg til mental fleksibilitet og utypes ikke videre i denne rapporten. En *struktur og organisasjonsmessig fleksibilitet* tilsier en kommando og kontrollstruktur som gir størst mulig handlefrihet til å komponere ulike taskorganiserte styrker, som kan settes sammen avhengig av situasjon (d v s de er behovsdrevet). Dette tilsier evne både til å inngå i allierte formasjoner og integrering på tvers av forsvarsgrenene. Slike strukturelle aspekter må et fremtidig K2IS kunne evne å understøtte. Eksempler på hva vi mener med dette er forsøkt beskrevet nedenfor.

For Sjøforsvaret vil f eks overgangen til en taskorientert organisasjon med oppdragsbasert ledelse medføre større krav til fleksibilitet i h t oppgaver og måter å organisere seg på, og at kommando og kontroll kan utøves flere steder. I dette ligger det et økt krav om fleksibilitet i det K2IS man har til rådighet. *Tilpasningsdyktighet* og *modifiserbarhet* vil i denne sammenheng være kritiske egenskaper med tanke på brukbarhet og levedyktighet for et fremtidig K2IS. Dette kan illustreres på følgende måte: Fleksibilitet kan bety systemers evne til å *kommunisere* med hverandre på flere nivåer og på tvers av teknologiske grenser

og plattformer. I såkalte komponentbaserte informasjonsssystemer vil fleksibilitet også bety evnen å til kunne *sette sammen* komponenter på tvers av teknologiske grenser og plattformer, der brukerbehovet til enhver tid bestemmer sammensetningen. Analogt vil fleksibilitet i en sjømillitær organisatorisk sammenheng bety evnen til å *"sette sammen"* styrkekomponenter som overflatefartøy, undervannsbåter og mobile kystartilleriavdelinger til en større styrke – en "pakke" – der oppdraget bestemmer sammensetningen av styrken (f eks Norwegian Task Group). Dette illustrerer en type fleksibilitet i ledelsessystemet som er analog med den fleksibiliteten man ønsker seg i et K2IS. D v s at det må være samsvar og sammenheng mellom infrastruktur i de underliggende teknologiske og funksjonelle systemene opp mot den operative struktur disse systemene er ment å understøtte.

Et annet eksempel på strukturell fleksibilitet er spennet fra direkteledelse til oppdragsbasert ledelse, og ledelsessystemets evne til hurtig å veksle mellom disse etter behov.

Alt dette er med på å sette nye operative og organisasjonsmessige krav til informasjonssystemene og den teknologiske infrastrukturen. En godt utformet arkitektur, som går ut over det teknologiske perspektivet, vil kunne støtte disse nevnte strukturelle aspektene ved et K2IS.

4.3 Teknologisk utvikling

Med dagens hurtige utvikling er teknologi nærmest blitt en "ferskvare" som på enkelte områder har kort levetid.

4.3.1 Teknologiuavhengighet

Ved å bruke arkitekturer til å isolere beskrivelsen av de mest "bestandige" delene av et system innenfor et område, kan håndtering av endringer til disse systemene forenkles. Ved å ha en mer teknologiuavhengig beskrivelse av virksomheten, f eks en ledelsesmodell med definerte behov og krav, vil en lettere kunne gjennomføre et skifte i valg av teknologi. Dette kan bidra til økt teknologiuavhengighet.

4.3.2 "Legacy systems"

I fremtiden vil man fremdeles ha behov for å integrere og anvende generasjoner av informasjons– og programvaresystemer sammen med andre systemer. Mange av de eksisterende systemene vil også i fremtiden være kritiske for Sjøforsvarets virksomhet slik at de må påberegnes å være gjenstand for videreutvikling i flere år fremover. Disse systemene omtales ofte som "legacy"–systemer og kan ivaretas gjennom bruk av arkitekturer. Et fremtidig distribuert K2IS vil alltid inneholde "legacy"–systemer. En teknikk for å håndtere dette er gitt i kapittel 10.5.4.

4.3.3 "Nye muligheter skaper nye behov"

Et annet viktig aspekt er at økt tilgjengelighet av ny teknologi viser vei for endringer i arbeidsprosesser og nye måter å *ta i bruk* teknologi på. F eks dersom krav til tettere sam-

handling, koordinering og informasjonsdeling legges til grunn som prinsipper for organisering av arbeid, vil det over tid være med på å skape behov for å ta i bruk ny teknologistøtte i arbeidet – som f eks teknologier innenfor "Computer Supported Cooperative Work" (CSCW) og annen informasjons– og kommunikasjonsteknologi (IKT). Ny teknologistøtte kan igjen føre til fremvekst av nye og kanskje mer effektive arbeids– og samarbeidsformer. Bruk av arkitekturer kan bidra til lettere å inkludere ulike former for nyere teknologier innen f eks gruppevare, agentsystemer, brukergrensesnitt, kunnskapssystemer og multimedia. På den måten kan arkitekturer bidra til å støtte opp under samspillet mellom teknologi og mennesker i arbeid. Dette er et viktig aspekt som representerer et stort potensiale for bedre og mer effektiv utnyttelse av det som til enhver tid er av tilgjengelig teknologi på det kommersielle markedet.

Behov som skal dekkes vil noen ganger kreve hurtig gjennomføring, mens andre vil måtte integreres i mer langsiktige utviklingsprosjekter. F eks bør mindre systemutviklings–prosjekter eller innføringsprosjekter kunne foregå på lokalplan i organisasjonen uten å måtte være sentralstyrt. En referansearkitektur skal gi organisasjonen fleksibilitet – ikke redusere den. Bruk av systemarkitekturer fundert på den samme referansarkitektur vil i en slik sammenheng bidra til fleksibilitet i forholdet lokale utviklingsbehov kontra sentrale og mer langsiktige utviklingsbehov. Det gjør den ved at man på lokalplan ikke trenger å komme i konflikt med sentrale utviklingsprosjekter nettopp fordi man forholder seg og utvikler over "samme lest" (f eks man forholder seg til de samme modellene, grensesnittene og teknologistandardene). På denne måten unngår man suboptimalisering samtidig som man får økt "oversikt, styring og kontroll" over systemutviklingsprosessene gjennom referansearkitekturen.

4.4 Operativ interoperabilitet

Et av hovedformålene ved å bruke arkitekturer er å oppnå interoperabilitet.

Selv om interoperabilitet mellom systemer alltid har vært viktig for Sjøforsvaret og Forsvaret generelt, har interoperabilitet i den senere tid fått et mye større fokus enn tidligere. Strukturelle endringer og økt deltagelse i internasjonale operasjoner har aktualisert denne problematikken i stor grad, noe som medfører strengere krav til systemintegrasjon og interoperabilitet mellom våre nasjonale systemer og mot internasjonale samarbeidspartnere.

I NATO defineres interoperabilitet på følgende måte (1):

"The ability of systems, units or forces to provide services to and accept services from other systems, units or forces and to use the services so exchanged to enable them to operate effectively together"

For Sjøforsvaret vil interoperabilitet på alle nivåer være et krav. I dette ligger det at sammen med krav til interoperabilitet på systemnivå vektlegges det også en organisasjonsmessig interoperabilitet. Man må materielt, prosedyremessig og doktrinemessig være interoperable. Dette vil i stor grad omfatte det som man refererer til som *operativ interopeabilitet*

(se diskusjon kapittel 4.4.1). For å få økt nasjonal og internasjonal operativ interoperabilitet er det selvfølgelig en forutsetning at alle de underliggende systemer er interoperable. Fra et arkitekturperspektiv vil dette bety at det ligger et stort ansvar på de operative myndigheter for å gjøre tilgjengelig den operative kompetansen som er nødvendig for at det settes riktige og relevante krav til utforming av arkitekturer i henhold til operative behov.

I delkapittelet 4.4.1 gis kun en kort summarisk oversikt over sentrale arkitektur– og interoperabilitetsarbeider i NATO som vil være med på å sette føringer for et eventuelt arkitekturarbeid i Forsvaret.

4.4.1 Arkitektur– og interoperabilitetsarbeid i NATO

I NATO har det blitt et betydelig strengere krav til det som nå omtales som C3–interoperabilititet. C3 står for *Consultation, Command and Control*, og i NATO defineres C3–interoperabilitet på følgende måte (4):

"The degree to which the consultation, command and control systems of different forces, including forces from different nations, within NATO can work together in the planning and execution of combined and/or joint operations across the spectrum of conflict. Effective C3 interoperability requires common, or commonly understood, C2 doctrine and procedures and the timely exchange of all relevant information to ensure unity of effort, within the philosophy of decentralised command, and full integration and co–ordination of NATO forces in support of COMAJF's missions"

Krav om operativ interoperabilitet og C3–systeminteroperabilitet ("C3–system interoperability") har fått høy prioritet i NATO.

C3–systems (C3S) defineres som følger (33):

"Collective term for communication and information systems, sensor systems and facilities which enable authorities and their staff to carry out consultation, command and control activities"

NATOs prioriteringer på dette området legger sterke føringer på alle alliertes nasjonale systemer. I denne sammenheng opererer man med en interoperabilitetsreferanse som følger:

- **Vertikal interoperabilitet** er definert som evnen til å dele informasjon gjennom hele kommandokjeden (fra politisk–strategisk nivå, militær strategisk nivå, operativt nivå og ned til taktisk nivå).
- **Horisontal interoperabilitet** er definert som evnen til å dele informasjon på tvers av forsvarsgrener (land, sjø, luft), på tvers av funksjoner (f eks etterretning og logistikk) og mellom nasjonale styrkeelementer.
- **Intern interoperabilitet** er definert som NATOs og de innbyrdes alliertes nasjonale systemers evne til å dele informasjon horisontalt og vertikalt.

- **Ekstern interoperabilitet** er definert som NATOs og de innbyrdes alliertes nasjonale systemers evne til å støtte den konsultative prosessen (Consultation) med partnere og andre sammarbeidende nasjoner som deltar i NATO–operasjoner.

Vertikal og horisontal interoperabilitet refererer til organisasjon mens intern og ekstern interoperabilitet referer seg til C3–systemers interoperabilitet, d v s C3–systemers evne til å støtte vertikal og horisontal interoperabilitet. Ekstern interoperabilitet refererer seg til de konsultative prosessene.

4.4.1.1 NATO Policy for C3 Interoperability

I arbeidet med å oppnå økt C3–integrasjon er det sterkt fokus på bruk av arkitekturer, og det er igangsatt en rekke tiltak.

North Atlantic Council (NAC) opprettet i juli 1996 NATO Consultation, Command and Control Organisation (NC3O) hvor NATO C3 Board (NC3B) er det øverste styrende organ. NCB3 har godkjent *NATO Policy for C3 Interoperability* som beskriver NATOs overordnete programplan for C3–interoperabilitet. Denne programplanen inngår som et element i *NATO Interoperability Framework* (NIF) hvor *NATO C3 Interoperability Environment* (NIE) står svært sentralt. I NIE fokusereres det spesielt på utvikling og bruk av felles arkitekturer og standarder. I tillegg omfatter NIE et testbed (NIETI) for interoperabilitetstesting og demonstrasjoner av interoperabilitetsprodukter.

4.4.1.2 NIF

NIF styres og forvaltes av NC3B og er det overordnete rammeverk for å forvalte, samordne og koordinere alt arbeid innenfor sentrale interoperabilitets– og standardiseringsområder i NATO. NIFs overordnete målsetting er å øke de alliertes C3–effektivitet. NIF består av fire deler (eller lag) (4), se tabell 4.1.

4.4.1.3 NIE

NIE inngår som ett av fire sentrale elementer i NIF. NIEs rolle er å identifisere produkter for design og implementasjon av interoperable C3–systemer. NIE fokuserer spesielt på utvikling av felles arkitekturer, standarder og profiler vedrørende *interoperabilitetsprodukter.* NIE refereres til som "produktlaget" i NIF ("The NIF C3 Products Layer").

NC3 Technical Architecture (NC3TA) er et eksempel på et interoperabilitetsprodukt fra NIE. I 1997 fikk Open Systems Working Group (NOSWG) i oppdrag fra Information System Sub–Committee (ISSC) å utarbeide NC3 Common Operating Environment (NC3COE) og NC3 Common Standards Profile (NCSP). NC3TA ble utviklet med planlagt første versjon ferdigstilt juni 1999. NC3TA er en sentral del av NIE hvor NC3COE og NCSP utgjør deler av denne beskrivelsen. NC3TA er ikke en implementasjonsarkitektur, men definerer et minimum sett av regler, tjenester, grensesnitt og retningslinjer som kreves for å implementere interoperable systemer. Hensikten med NC3TA er å gjenspeile alle relevante NATO og nasjonale innsatser gjort i forbindelse med "Communication and Infor-

mation Systems" (CIS)–standariseringen, og vil anvendes på alle NATO CIS for å sikre både effektiv implementasjon og interoperabilitet innenfor NATO og innenfor nasjonale systemer. NC3TA er et resultat av standardiseringsarbeidet i NIE, og er nå godkjent av NC3B. NC3TA inngår som et element i NATO C3 Systems (NC3S) Architecture Framework.

C3 Elements of NATO Interoperability Framework				
	Role	**Objective**	**Scope**	
NATO Policy for C3 Interoperability	Provide Overarching C3 Interop Policy, Strategy and Framework	Achieve Interoperability of NATO and National C3 Systems	Address essential C3 Interop Issues and Define NIF	**POLICY**
NIMP: NATO Interoperability Management Plan	Manage the execution of the C3 Interoperability Policy and Strategy	Management Plan	Processes, Procedures and Relationships	**EXECUTION**
RIP: Rolling Interoperability Programme	Manage and coordinate NATO Programmes and coordinate with National Interoperabiliy Programmes	Achieve Interoperability/ Standardization Objectives	5–year Interoperability Programme	
NIE: Nato C3 Interoperability Environment	Spesifies Architectures, NCIS, Profiles and NC3COE Subset required for Interoperability	Common Architecture, Standards, Profiles	Interoperability Products	**PRODUCTS**

Tabell 4.1 Lagene i NIF–rammeverket (4)

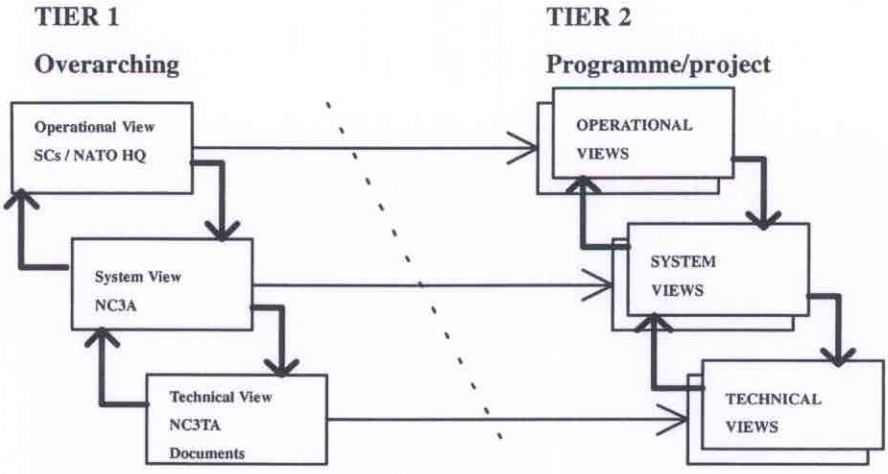
4.4.1.4 "NC3S Architecture Framework"

I NATO er man inspirert av DoDs "Command, Control, Communications, Computers, Intelligence, Surveillance and Reconnaissance Architectural Framework" (C4ISR–AF), og arkitekturarbeider innenfor NATO er hovedsaklig fundert på dette rammeverket. Dette gjelder også arbeidene innenfor Arcitecture Framework for NC3S (33). Hensikten med NC3S er å definere regler for hvordan man skal beskrive arkitekturer innenfor C3–domenet. Disse reglene er basert på "Operational View", "System View" og "Technical View". NC3 System Architecture er en fellesbenevnelse for et sett av arkitekturbeskrivelser for fremtidige NATO C3–systemer. Dette inkluderer følgende fem NC3S–arkitekturbeskrivelser: Base–line ("as–is"), Target ("transitional"), Reference ("to–be"), Functional og Overarching. Nærmer beskrivelse av disse er gitt i (33).

I NC3S er det lagt opp til en to–lags ("two–tier") arkitekturtilnærming til C3–systemer (figur 4.1).

Lag 1 skal reflektere NC3S overordnete planer. De operative elementer hentes fra Bi–Strategic Commands (Bi–SC) C2 Plan (36) og Political Consultation and NATO Civil Emergency Planning (PCNCEP) CIS Concept, samt operative konsepter og doktrine fra NATO Political and Military Authorities. Systemelementene vil i hovedsak reflektere NATO C3

policy aspekter, og tekniske elementene hentes fra NC3TA. Lag 2 (høyre side i figuren) refererer til system–/subsystem– og prosjektnivå, og skal støtte design og implementasjon av C3–systemer i henhold til NIE.

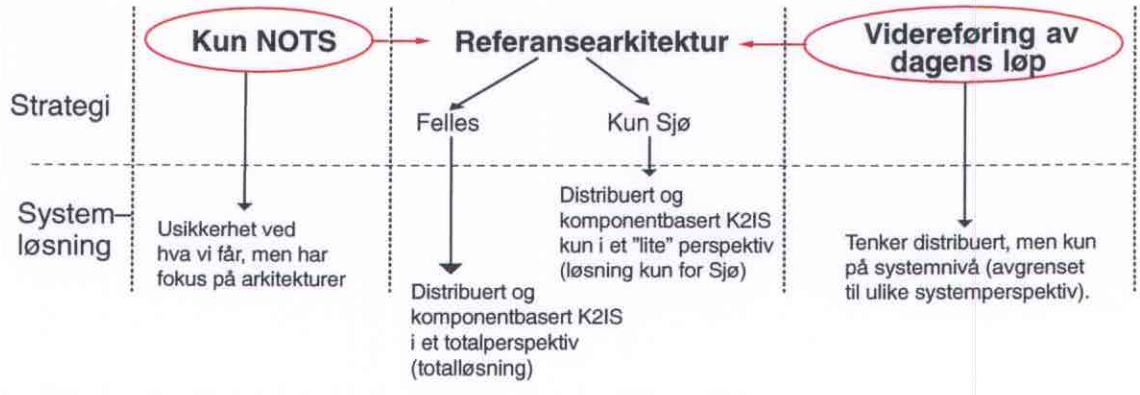


Figur 4.1 NATO C3S Architectural Construct (forenklet) (33)

4.5 Mulige strategialternativer ved bruk av arkitekturer

Ulike strategier legger sterke føringer på hvilke systemløsninger man til slutt ender opp med. Dersom målet er å få et fremtidig sømløst og integrert K2IS, vil valget av strategi være helt avgjørende for i hvor stor grad – og hvor godt man lykkes i å nå målet. Det kan tas utgangspunkt i tre strategialternativer:

1. Basere seg kun på eksisterende (og fremtidige) NOTS–systemer (d v s ikke egenutviklet)
2. Bruk av nasjonal referansearkitektur som fundament i en overordnet K2IS–strategi
3. Fortsette dagens løp og "systemvise tenkning"



Figur 4.2 Enkel illustrasjon av hvordan vi tenker oss mulighetsrommet for veier å gå.

4.5.1 Kun NOTS–systemer

Velger man å basere seg kun på NATO–systemer, og følge de arkitekturer som ligger til grunn der, vil det gi en usikkerhet i forhold til hva vi får av systemløsning(er) og funksjonalitet. Innenfor NATO foregår det arbeid som retter seg inn mot arkitekturer og komponentbasert tankegang, men det er vanskelig å si noe om når det eventuelt får en effekt på systemnivå. Problemet her knytter seg til at mye av det arkitekturarbeidet som hittil har vært gjort i regi av NATO foreløpig kun befinner seg på papiret. Vi har så langt ikke sett at noe vesentlig av dette har materialisert seg i eksisterende NATO–systemer (vi ser da bort fra produktstandarder). MCCIS er et eksempel på dette. Et annet eksempel på dette er arbeidet innenfor "Allied Command Europe Automated Command, Control and Information System" (ACE ACCIS). F eks er samvirke med andre kommando og kontroll systemer utenfor ACE ACCIS lite beskrevet – noe som skaper stor usikkerhet i forhold til fremtidig interoperabilitet mellom ACE ACCIS og nasjonale systemer (21). ACE ACCIS bygges dessuten over en tradisjonell klient–tjener arkitektur og ikke en komponentbasert arkitektur.

Ved å gå for kun NOTS–systemer, vil vi også være underlagt det NATO bestemmer seg for og vi vil få liten mulighet til å påvirke utviklingen. Dette vil bl a innebære at Forsvaret blir avhengig av NATO som eneste "leverandør".

En forutsetning for i det hele tatt å vurdere dette alternativet er at man må være aktiv pådriver overfor NATO for å sikre nasjonale behov. Det kan bli vanskelig ettersom et av de store problemene og risikomomenter som fremkommer ved valg av en slik strategi, er at NATO i liten grad tar hensyn til alle nasjoners nasjonale behov i utvikling av systemer. Skal en f eks ha noen reell mulighet til å påvirke arkitekturabeidet i NATO, må en ha en egen arkitekturaktivitet gående her hjemme som kan knyttes opp til konkrete systemer.

Etter vår oppfatning er det ingen grunn til å tro at NOTS kan dekke alle våre nasjonale behov. I fremtiden vil det fremdeles være nødvendig å ivareta særbehov i forhold til krav om støtte og funksjonalitet, eller tjenester, som informasjonssystemet bør tilby. Dette er en oppfatning som deles av flere deltagere i NATO. Dette gjelder spesielt i f m ulike nasjonale oppgaver som f eks suverenitetshevdelse og myndighetsutøvelse. Da vil man ha behov for å utvikle egne nasjonale (del)systemer. I slike tilfeller vil problemet med en ren NOTS–strategi være at eventuelle nasjonale tilpasninger må enten skje ved å påvirke utviklingen av NOTS–systemene, eller at NOTS–systemene i utgangspunktet åpner for lokale tilpasninger. Dette forutsetter bl a betydelig økt nasjonal kompetanse på NOTS–systemer i forhold til utvikling, bruk og drift.

Fordelen ved å gå for en NOTS–strategi vil være at det vil være minimale utviklings–kostnader av disse systemene knyttet til våre nasjonale budsjetter (utenom de utviklings–kostnader som allerede inngår som vår andel i NATO). Ulempen er at det vil være høye utskiftningskostnader – spesielt i tilknytning til oppgraderinger. Ved å gå til anskaffelse av kun NOTS–systemer vil man også risikere å måtte kjøpe mer enn man trenger. Det innebærer bl a at vår andel av kostnader knyttet til utvikling av disse systemene går til å betale for utvikling av funksjonalitet som vi ikke har bruk for. Også kostnader forbundet med nasjonal kompetanseoppbygging og opplæring på NOTS–systemer kan bli betydelig.

På kort sikt kan det muligens fortone seg relativt billig å gå for en NOTS–strategi, men på lang sikt kan det bli kostbart. Etter vår oppfatning er det for mange usikkerhetsmomenter knyttet til dette alternativet.

4.5.2 Nasjonal referansearkitektur

En referansearkitektur kan brukes som et planverk for samordning og koordinering av K2IS–løsninger.

I Forsvaret foregår det aktiviteter både på arkitekturer og distribusjonsteknologi, men det er liten koordinering mellom de ulike arkitektur– og systemmiljøene. Dette er en svært ugunstig utvikling. I dag finnes det utviklingsmiljøer knyttet opp til enkeltsystemer (f eks Norwegian Command and Control Information System (NORCCIS)–miljøene, Geographic Information System (GIS)–miljøene, o s v) som riktignok tenker arkitekturer – men *hver for seg*. Å fastholde en slik inndeling og "systemvis tenkning" byr på problemer. Først og fremst i forhold til kostnader knyttet til overlappende funksjonalitet og bruk av "proprietære" løsninger, men også ikke minst i forhold til høye infrastrukturkostnader[5] knyttet til enkeltsystemer. De pågående arkitekturarbeidene vil mest sannsynlig ikke gi gevinster eller synergieffekter på tvers av systemene, men kun innenfor sine egne spesifikke områder.

En referansearkitektur kan enten realiseres kun for Sjøforsvaret, eller ved en koordinering med de to andre forsvarsgrenene (Hær og Luft). Ved å utvikle en referansearkitektur kun for Sjøforsvaret, vil alle kostnadene for dette arbeidet tilfalle Sjøforsvaret. Og hvis i tillegg Hæren (i f m med arbeidet med Minimal Architecture for Command and Control Information Systems (MACCIS))[6] og Luftforsvaret utfører tilsvarende arbeider, vil det utgjøre mye dobbelt/trippel arbeid om mange av de samme tingene. Faren er også stor for at man kan bidra til å opprettholde mangel på interoperabilitet mellom forsvarsgrener i stedet for å øke den. Man kan selvfølgelig oppnå et distribuert og komponentbasert maritimt K2IS forankret i en maritim referansearkitektur, men det fremmer ikke nødvendigvis interoperabilitet mellom grenene. Det vil være svært ugunstig hvis hver forsvarsgren utfører arkitekturarbeid hver for seg.

Ved å etablere en felles referansearkitektur for hele Forsvaret vil en kunne få synergi–effekter ved felleskoordinering og samordning, samt en kostnadsfordeling mellom de ulike forsvarsgrenene. Ved å spesifisere en felles referansearkitektur for et felles K2IS vil man på denne måte ivareta og nedfelle en overordnet felles strategi, for å veilede utvikling av både felles og grenspesifikke delsystemer. Dette vil bidra til å øke interoperabilitet på tvers av forsvarsgrener (horisontal interoperabilitet) og vertikalt.

For å øke og sikre interoperabilitet mot internasjonale samarbeidspartnere og mot NATO er det en forutsetning at man koordinerer arkitekturarbeidene mot det som foregår av arkitektur og interoperabilitetsarbeid i NATO. Skal en f eks ha noen reell mulighet til å påvirke

5. F eks ved utskiftning av infrastruktur med moderne COTS mellomvare "tvinges" systemleverandører over på en felles infrastrukturplattform. Ved da å fokusere de økonomiske midlene mot sentrale og nødvendige tjenester, vil man på sikt få en innsparing.

6. MACCIS beskrives nærmere i kapittel 5.3.

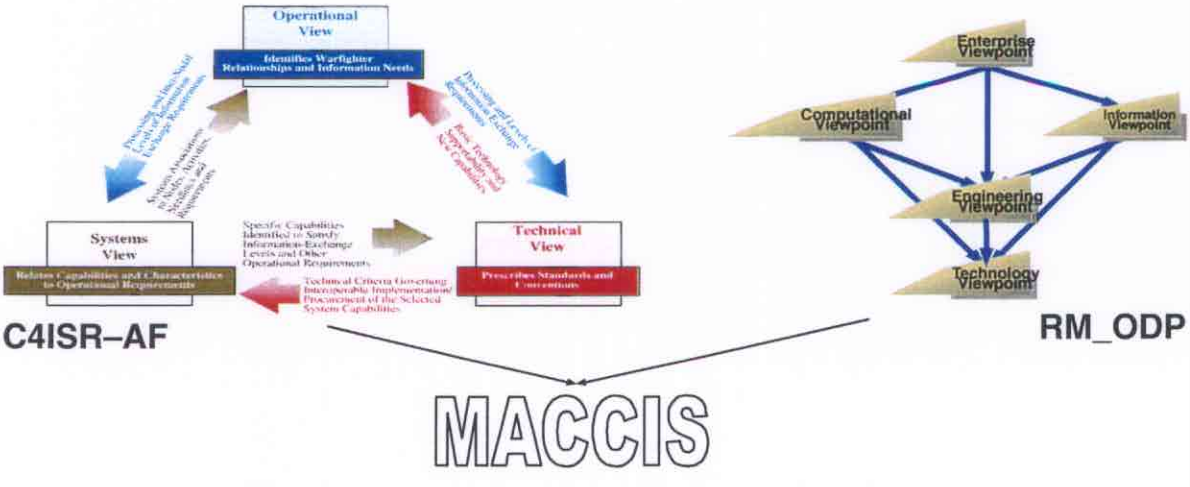
arkitekturabeidet i NATO, må en ha en egen arkitekturaktivitet gående her hjemme som vi kan knytte opp til konkrete systemer. En felles nasjonal arkitekturaktivitet vil styrke en slik mulighet.

5 ARKITEKTURRAMMEVERK

I dette kapittelet beskrives kort de tre mest sentrale arkitekturrammeverkene som til nå er utviklet og som vil ha betydning for utviklingen av fremtidige maritime K2ISer.

- C4ISR–AF er et arkitekturrammeverk utviklet av det amerikanske Forsvaret, spesialdesignet for kommando og kontroll informasjonssystemer gjeldende innenfor militære organisasjoner. Beskrivelsen av C4ISR–AF i denne rapporten er hovedsaklig basert på (16).
- RM–ODP arkitekturrammeverket er en ISO–standard med fokus på åpne distribuerte systemer. RM–ODP beskrives fordi det har et spesielt fokus på distribuerte systemer og interoperabilitet, og er det eneste, som frem til nå kan sies å være et helt ferdigstilt arkitekturrammeverk. Beskrivelsen av RM–ODP i denne rapporten er hovedsaklig hentet fra (13) og (32).
- Norskutviklede Minimal Architecture for Command and Control Information Systems (MACCIS) har som mål å bli en norsk hærspesifikk referansearkitektur, men er likevel valgt beskrevet under dette kapittelet. Det er fordi MACCIS, slik den foreligger i dag, er såpass generell at den mer kan betraktes som et arkitekturrammeverk enn en referansearkitektur. MACCIS er en kombinasjon av C4ISR–AF og RM–ODP. Beskrivelsen av MACCIS i denne rapporten er basert på beskrivelsen gitt i (17).

Felles for alle tre rammeverkene er et sterkt fokus på interoperabilitet.



Figur 5.1 Et arkitekturrammeverk beskriver de modeller som skal brukes for å spesifisere forskjellige aspekter ved et system.

5.1 C4ISR–AF

Hovedmålsettingen for utviklingen av C4ISR–AF var å få definert en felles og samordnet tilnærming for C4ISR innenfor militære organisasjoner. Målet med C4ISR–AF var at det skulle være en felles tilnærming for *alle militære enheter* hvor C4ISR–AF er et arkitekturrammeverk som beskriver regler, retningslinjer og produkter for utvikling av arkitekturer innenfor kommando og kontroll. Målet er å sikre felles forståelse av og interoperabilitet mellom ulike arkitekturer, både horisontalt og vertikalt – fra laveste til høyeste nivå i en militær organisasjon.

C4ISR–rammeverket betrakter et K2–system ut fra tre adskilte arkitekturperspektiver ("viewponts"). Hvert perspektiv fokuserer på bestemte aspekter av det samme systemet:

1. **Operativt perspektiv**: Inneholder beskrivelser av oppgaver, aktiviteter og informasjonsflyt som kreves for å gjennomføre eller støtte militære operasjoner. D v s det gis definisjoner av hvilke typer informasjon som skal utveksles, hvor ofte informasjonen skal utveksles (informasjonsutvekslingsfrekvensen), hvilke oppgaver og aktiviteter som støttes av denne informasjonen og gjeldende interoperabilitetskrav.

2. **Systemperspektiv**: Beskriver systemer og forbindelser mellom dem for å tilrettelegge, eller for å støtte krigførende funksjoner. D v s for et gitt domene (funksjonsområde) vises det hvordan de underliggende systemer er koblet sammen og hvordan systemene samarbeider. I tillegg er det en beskrivelse for hvert av systemene. For hvert system vil beskrivelsen inkludere fysisk tilknytning, lokalisering, oversikt over nøkkelnoder, nettverk, o s v, samt spesifikasjon for systemets ønskede egenskaper (systemkvaliteter, som f eks tilgjengelighet, o s v)

3. **Teknologisk perspektiv:** Beskriver et sett av regler for styring av tilretteleggelse, interaksjon og avhengighet mellom systemer, og mellom deler av systemer. Dette for å sikre at systemer som skal samvirke med hverandre tilfredsstiller et spesifisert sett av definerte krav. D v s at her beskrives retningslinjer for systemimplementasjon i tillegg til en samling av teknologiske standarder, konvensjoner, regler og kriterier for styring av systemtjenester og grensesnitt. Spesielle relasjoner beskrives også for å sikre sammenheng mellom de teknologiske sidene av systemene opp mot det operative perspektivet og systemperspektivet.

I tillegg til beskrivelser av felles definisjoner og referanser, beskrives et sett av produkter som i hovedsak utgjør disse tre arkitekturperspektivene. Imidlertid gis det lite eller ingen veileding i hvordan man bør produsere disse produktene (29).

Arbeidet med dette rammeverket ble startet i 1993 og den første versjonen til felles standard ble publisert 7 juni 1996. Andre versjon ble publisert 18 desember 1997 (16) og er den som blir referert ovenfor. Den tredje versjonen skulle ha vært publisert i 1998 men er fremdeles ikke ferdigstilt.

I NATO er man inspirert av C4ISR–AF, og arkitekturarbeider innenfor NATO er hovedsaklig fundert på dette rammeverket (kapittel 4.4.1.4). I NC3S–arbeidet (33) er f eks

NC3TA basert på teknologisk perspektiv, og NATO C3 System Architecture (NC3SA) er basert på systemperspektivet.

5.1.1 Vurdering

I konteksten kommando og kontroll er det en del fordeler ved C4ISR–AF. For det første er rammeverket spesialdesignet for militære systemer. For det andre gis det gode produktbeskrivelser, og det utheves bra hvilke produkter som er de mest sentrale. I tillegg gis det gode eksempler.

Imidlertid er det noen svakheter ved C4ISR–rammeverket, som etter vår mening reduserer brukbarheten slik det fremstår i dag.

Først og fremst så er hele rammeverket fundert på en funksjonsorientert tilnærming, som er en mer tradisjonell måte å tenke på innenfor systemutvikling. En funksjonsorientert tilnærming resulterer ofte i systemer med liten fleksibilitet. F eks hvis kravene til systemet endres, må det ofte gjøres omfattende restruktureringer av hele systemet. Dette problemet er spesielt aktualisert i forhold til dagens komponentorienterte teknologiutvikling. En funksjonsorientert tilnærming har klare svakheter i å beskrive distribuerte systemer nettopp fordi den ikke fokuserer på objekter[7].

Det er også uheldig at rammeverket ikke tilbyr en prosessbeskrivelse for arkitekturdesign. Noe som bl a betyr at de tre arkitekturperspektivene og deres produkter er konstruksjoner som pr i dag ikke direkte støttes av noe eksisterende systemformalisme. Det er opp til organisasjonene selv å utvikle slike prosessbeskrivelser. Denne problematikken blir forøvrig adressert av Alexander H. Levis og Lee W. Wagenhals i (29) hvor to alternative tilnærminger til en slik prosessbeskrivelse blir presentert. Den ene funksjonsorientert (30), den andre objektorientert (31).

I (24) påpekes det at C4ISR–AF mangler retningslinjer for å opprettholde konsistens i arkitekturen, d v s relaterbarhet mellom arkitekturperspektivene. I tillegg er det, etter vår oppfatning, en svakhet at informasjonsaspektet ikke er fremhevet som et eget arkitekturperspektiv som kan behandles separat.

5.2 RM–ODP

RM–ODP er et objektorientert arkitekturrammeverk og fokuserer på åpne distribuerte systemer. Distribuerte systemer har typiske karakteristiske fellestrekk som f eks heterogenitet, autonomitet, mobilitet og foranderlighet. For å kunne håndtere karakteristikker av denne type har RM–ODP hatt som mål spesielt å støtte:

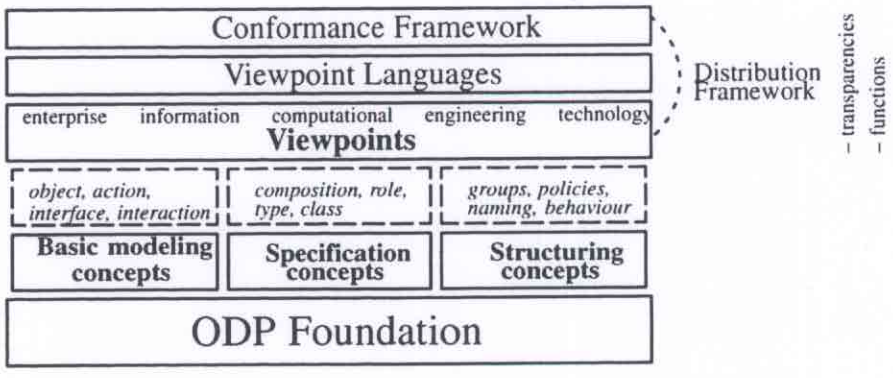
- distribusjon og transparens (gjennomsiktighet) i systemer
- utveksling av informasjon og tilpasset bruk av funksjonalitet mellom distribuerte systemer

7. Funksjonsorientert og objektorientert tilnærming er nærmere beskrevet i kapittel 9.1.

- interoperabilitet
- portabilitet, d v s applikasjoners flyttbarhet mellom heterogene plattformer

RM–ODP er en "referansemodell" – ikke en implementasjonsstandard. Med det menes at RM–ODP har som hensikt å gi det "store bilde" av hvordan man kan organisere delene i et distribuert system slik at det utgjør et samvirkende hele. RM–ODP er abstrakt i den forstand at den skal kunne brukes til å beskrive dagens systemer såvel som fremtidige systemer. F eks så er det ikke i RM–ODP gjort noe forsøk på å standardisere systemkomponenter, og den influerer heller ikke på valg av teknologi.

RM–ODP har et veldefinert begrepsapparat og definerer et sett av grunnleggende konsepter ("Basic modeling concepts", Specification concepts", Structuring concepts"). Modellkonseptene definerer RM–ODPs objektmodell som danner grunnlaget for standardens objektorienterte tilnærming ("ODP Foundation" i figur 5.2). Spesifikasjonskonseptene definerer elementer i forhold til ulike typer, klasser, roller, o s v. De strukturelle konseptene definerer strukturer i distribuerte systemer (f eks oppførsel og kommunikasjon). Videre defineres transparenser som konkrete standarder.



Figur 5.2 RM–ODP struktur (32)

I RM–ODP belyses et distribuert system ut fra fem adskilte, men relaterte arkitekturperspektiver ("viewpoints") som tilsammen beskriver hele systemet:

1. *"Enterprise":* beskriver formålet, omfanget og policy for systemet i organisasjonen.
2. *"Computational":* beskriver komponentene i et distribuert system, samt interaksjonen og grensesnitt mellom kompontene.
3. *"Information":* beskriver informasjons–innhold, –flyt og –prosessering i systemet.
4. *"Engineering":* beskriver distribusjonsaspektet, f eks hvilken infrastruktur som kreves for å støtte distribusjon (f eks transparensmekanismer, prosessorer, hukommelse, kommunikasjonsnett, o s v).
5. *"Technology":* beskriver den underliggende infrastrukturen – d v s de fysiske komponentene (maskinvare– og programvarekomponenter), som utgjør et distribuert system som er relatert til de modellene som er definert i de andre arkitekturperspektivene.

Hver av disse spesifikasjonene (perspektivene) uttrykkes i et såkalt arkitekturperspektiv-språk ("Viewpoint Language"). Hvert av språkene innehar et spesifikt sett av begreper og grammatikk, og er kun en beskrivelsesteknikk for å skille spesifikasjonen fra selve modellen som skal konstrueres. Dette er et abstrakt språk brukt kun til dette formål og er således ingen konkret notasjon.

5.2.1 Interoperabilitetsnivåer

RM–ODP modellen ser interoperabilitet relativt til disse fem perspektivene:

1. *Enterprise – Organisatorisk interoperabilitet:* For at to forskjellige organisasjoner skal kunne interoperere er organisatorisk interoperabilitet nødvendig. Mer spesifikt må organisasjonene bli enige om hva systemene skal gjøre, har lov til og ikke har lov til å gjøre.

2. *Computational – Komponent– og tjenesteinteroperabilitet:* To systemer er komponent– og tjenesteinteroperable hvis det er enighet om settet av tjenester som tilbys av komponenter i de to systemene og grensesnittet.

3. *Information – Informasjonsmodellinteroperabilitet:* krever at to systemer må være syntaktisk og semantisk interoperable. To systemer er syntaktisk interoperable hvis de bruker den samme strukturen for å utveksle informasjonen i systemet. To systemer er semantisk interoperable hvis de har en felles oppfatning av informasjonen som de utveksler. F eks ses databasereplikering ofte som en løsning på interoperabilitet mellom datasystemer. Denne måten å betrakte interoperabilitet på, tvinger systemene som ønsker å være interoperable, å bruke en felles/standard datamodell. I et heterogent miljø vil dette være et altfor sterkt krav. Forskjellige systemer vil sannsynligvis ha sine egne datamodeller, og interoperabilitet på informasjonsnivå bør heller skje ved en felles/ standard informasjonsutvekslingsmodell. Det er ellers verdt å merke seg at RM–ODP definerer replikering som en transparensmekanisme (se punkt 4 nedenfor) for å øke robustheten (redundans) og ytelsen til et system, og ikke et middel for å oppnå interoperabilitet.

4. *Engineering – Transparensmekanismeinteroperabilitet:* fokuserer på distribusjons-aspektet, d v s den infrastrukturen som kreves for å støtte distribusjon. Transparens skjuler konsekvensene av distribusjon. Distribuerte komponenter oppfattes av brukere og applikasjonsprogrammer som ett enhetlig integrert system i stedet for en samling av samarbeidende komponenter. En "engineering"–spesifikasjon definerer en nettverksinfrastruktur som støtter systemstrukturen i "computational"–spesifikasjonen og tilbyr de distribusjonstransparensene som kreves. To systemer kan sies å være transparensmekanismeinteroperable hvis de begge har interoperable mekanismer som støtter de samme distribusjonstransparensene. Transparensmekanismene er som følger:

   - *Aksesstransparens* skjuler datarepresentasjon og påkallingsmekanismer mellom kommuniserende objekter. Aksesstransparens har spesielt fokus på grensesnitt.
   - *Feiltransparens* muliggjør feiltolerante objekter ved at feil og gjenopprettelser skjules. Dette tillater brukere og applikasjoner å fullføre deres oppgaver selv om andre

komponenter feiler (i distribuerte systemer kan komponenter feile uavhengig av hverandre. Dette kalles "delvis feiling").

- *Lokasjonstransparens* sørger for at objekter kan nås uten å ha kunnskap om deres fysiske plassering.
- *Migrasjonstransparens* tillater at objekter endrer fysisk plassering innen et system uten å påvirke operasjoner til brukere og applikasjonsprogrammer. Eksempel: internettsider som flyttes fra en maskin til en annen.
- *Relokasjonstransparens* skjuler relokeringen av et objekt fra andre objekter som refererer til det.
- *Replikeringstransparens* muliggjør at flere instanser av objekter kan brukes for å øke pålitelighet og ytelse uten kunnskap om replikaene hos bruker og applikasjonsprogrammer.
- *Persistenstransparens* skjuler ovenfor et objekt deaktivering og aktivering av andre objekter, inklusive seg selv.
- *Transaksjonstransparens* skjuler koordinering av aktiviteter tilhørende en konfigurasjon av objekter for å oppnå konsistens.

5. *Technology – Infrastrukturinteroperabilitet:* For å oppnå interoperabilitet på dette nivået krever man en infrastruktur som tillater komponentene i et distribuert system å interoperere. Common Object Request Broker Architecture (CORBA), Distributed Component Object Model (DCOM) og Java Remote Method Invocation (RMI) er eksempler på programvareomgivelser som tilbyr en slik infrastruktur. Disse teknologiene støtter i varierende grad distribusjonstransparensene beskrevet ovenfor.

5.2.2 Vurdering

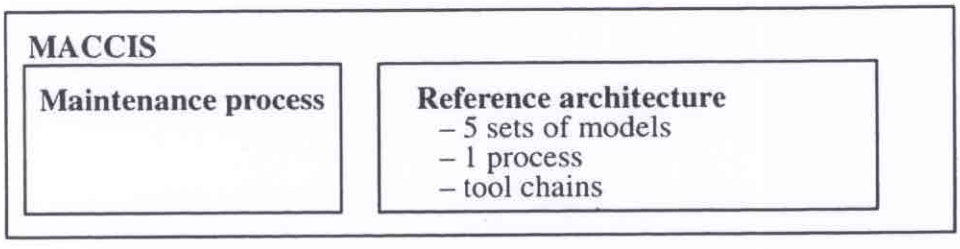
Etter vår oppfatning er RM–ODP et konsistent rammeverk og tilbyr et godt fundament for å beskrive distribuerte informasjonssystemer (modellene er objektorienterte og kan beskrives i Unified Modelling Language (UML)). Rammeverket har et veldefinert begrepsapparat og belyser et distribuert system ut fra fem forskjellige perspektiver, hvor bl a informasjonsperspektivet er gjort eksplisitt.

Det som imidlertid kan sies å være en svakhet ved RM–ODP (i forhold til vårt utgangspunkt), er at den hovedsaklig er programvareorientert. RM–ODP er generell i den forstand at den fokuserer på et avgrenset men likevel generelt teknologiområde (hvordan beskrive distribuerte systemer) og ikke så mye på i hvilke omgivelser disse systemene skal implementeres. RM–ODP er med hensikt gjort så generisk som mulig nettopp for å kunne anvendes innenfor de aller fleste områder. Dette i kontrast til C4ISR–AF som har sitt fokus på beskrivelse av systemer innenfor en militær kommando og kontroll organisasjon.

5.3 MACCIS

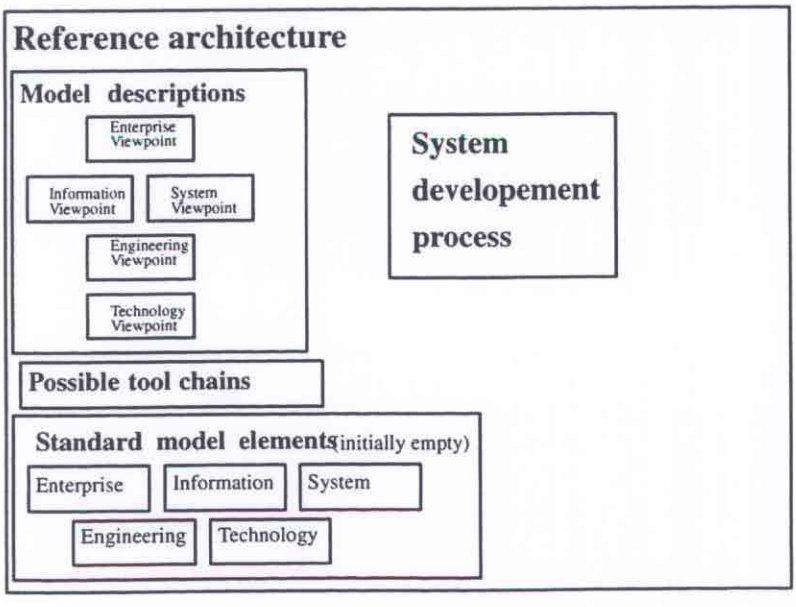
MACCIS ble utviklet av SINTEF Telecom & Informatics i perioden 1997 – 1999 på oppdrag fra Hæren i forbindelse med Hærens K2IS–program. Målsettingen er at MACCIS skal bli et sentralt verktøy (referansearkitektur) for tilretteleggelse av interoperabilitet i et hærspesifikt K2IS. Hovedkonseptene i MACCIS er basert på RM–ODP–rammeverket. Ved å bruke disse konseptene har man brukt C4ISR–AF rammeverket for å identifisere hvilke deler av et K2IS som skal beskrives. MACCIS fokuserer spesielt på robusthet, fleksibilitet, foranderlighet og interoperabilitet, og består av to deler (figur 5.3):

1. en *referansearkitektur* som beskriver hvordan systemer som følger denne arkitekturen skal beskrives og bygges.
2. en beskrivelse av en *vedlikeholdsprosess* for vedlikehold av denne referansearkitekturen inkludert prosedyrer for hvordan man skal håndtere forandringer i arkitekturen.



Figur 5.3 MACCIS består av en referansearkitektur og en vedlikeholdsprosess (17).

Referansearkitekturen (figur 5.4) består av en beskrivelse av fem modeller, en utviklingsprosess som beskriver hvordan man skal utvikle disse modellene og hvordan iterativ utvikling kan benyttes for dette.



Figur 5.4 Reference architecture (17)

Når modellene er utviklet vil disse representere de sentrale delene i systemet. I refereranse-arkitekturen er kun disse modellene og deres egenskaper beskrevet. Hensikten er at systemutviklere kan bruke disse modellbeskrivelsene når reelle systemer skal modelleres og utvikles. I referansearkitekturen er disse beskrivelsene av modellene gruppert inn under fem arkitekturperspektiver: "Enterprise", "Information", "System", "Engineering" og "Technology". Disse arkitekturbeskrivelsene er basert på RM–ODP sine arkitekurperspektiver, og der hvor RM–ODP refererer til "Computational" refererer MACCIS til "System". MACCIS vektlegger spesielt relaterbarhet mellom disse perspektivene.

I tillegg til en systemutviklingsprosess som beskriver hvordan modellene skal utvikles, inneholder referansearkitekturen også forslag til mulige verktøy som kan benyttes ("tool chains"). Disse verktøyene kan brukes for utvikling av modellene og av selve systemene.

Til sist er det meningen at referansearkitekturen skal inneholde et antall standard modell-elementer tilhørende enkelte av de fem settene av modeller (relatert til arkitekturperspektivene). Disse er ennå ikke beskrevet i MACCIS slik referansearkitekturen foreligger i dag. Derav navnet "Minimal Architecture".

MACCIS foreslår bruk av komponentstandarder og metodikk som er lojale overfor "Object Management Groups" (OMG) "Object Management Architecture" (OMA), som har til formål å oppnå:

- høy kvalitet, robusthet og produktivitet: Gjenbruk av kode, komponenter og modeller
- åpne systemer og interoperabilitet: Klart definerte grensesnitt mellom objekter og komponenter, en veldefinert arkitektur og bruk av objektstandarder og verktøy som støtter disse standardene
- fleksibilitet, utvidbarhet og vedlikeholdbarhet: Veldefinert arkitektur hvor funksjonalitet og data er lokalisert til objekter og komponenter og disse aksesseres kun gjennom veldefinerte grensesnitt.

OMGs OMA er en generell referansearkitektur for distribuerte objektsystemer som applikasjoner kan bygges over. OMA blir nærmere beskrevet i kapittel 6.1.

5.3.1 Vurdering

MACCIS er et eksempel på hvordan man kan kombinere styrken til to rammeverk: RM–ODP har sin styrke i fokuseringen på distribusjon og interoperabilitet og støtter objekttankegangen, og C4ISR–AF har sin styrke i forståelsen av militære systemer og identifiseringen av sentrale deler i et K2IS (produkter).

5.4 Konklusjon arkitekturrammeverk

Hovedfokus for alle tre rammeverkene er interoperabilitet. Sammenlikning med NATOs fire kategorier av interoperabilitet (kap 4.4.1) vil RM–ODPs og MACCIS' "Enterprise"–,

"Information–" og "Computational"–perspektiv spesielt støtte horisontal og vertikal interoperabilitet. I C4ISR–rammeverket støttes dette kun av operativt perspektiv.

Slik prosjektet ser det, vil det være naturlig å foreslå at man i arbeidet med en felles referansearkitektur ser nærmere på det arbeid som allerede er gjort i f m MACCIS. Skal det anbefales å utvikle en felles nasjonal referansearkitektur som fundament i en overordnet K2IS–strategi for Forsvaret, ville det mest fornuftige være å ta utgangspunkt i et arkitekturrammeverk som allerede eksisterer og som har tatt hensyn til norske forhold. Selv om MACCIS er påtenkt å bli en referansearkitektur for Hæren, er den foreløpig ikke hærspesifikk og er derfor et godt utgangspunkt for videre arkitekturarbeider både innenfor hær, sjø og luft. Fordelene er:

- MACCIS er utviklet på basis av RM–ODP som er tuftet på moderne teknologi–forståelse, og C4ISR–AF med innsikt i militære systemer, NATO–krav og standarder, samtidig som det er tatt hensyn til norske forhold.
- Ved å ta utgangspunkt i MACCIS unngår man kostnadene ved å starte helt fra "scratch".

Imidlertid vil en viktig forutsetning i et slikt arbeid være at man koordinerer nasjonale arkitekturarbeider med det som foregår av arkitektur– og interoperabilitetsarbeid i NATO.

6 REFERANSEARKITEKTURER

I denne rapporten fokuserer vi hovedsaklig på anvendelsen av en referansearkitektur som et overordnet planverk for samordning og koordinering av K2IS–løsninger.

Eksempler på referansearkitekturer er:

- Norskutviklete MACCIS
- OMGs OMA.

MACCIS og planene for denne kan i denne sammenheng sies å være et eksempel på en slik arkitekturtankegang. MACCIS er imidlertid i sin nåværende form mer å betrakte som et arkitekturrammeverk enn en referansearkitektur. MACCIS er derfor valgt beskrevet under arkitekturrammeverk (kapittel 5) og blir derfor ikke beskrevet videre i dette kapittelet. I dette kapittelet vil kun OMA bli beskrevet, hovedsaklig fordi det pr i dag er vanskelig å finne ferdigutviklede referansearkitekturer som kan gis som gode eksempler i henhold til vårt fokus nevnt ovenfor.

Når det gjelder komponentbaserte systemer kan en referansearkitektur også være en modell som beskriver komponentforbindelser, protokoller og kontroll. OMA er en slik referansearkitektur. Den består generelt av et delvis spesifisert system delt opp i et sett med generiske og abstrakte komponenter i tillegg til standarder for komponenter. En referansearkitektur av denne type er en kjernearkitektur som kan utvides med domenespesifikke

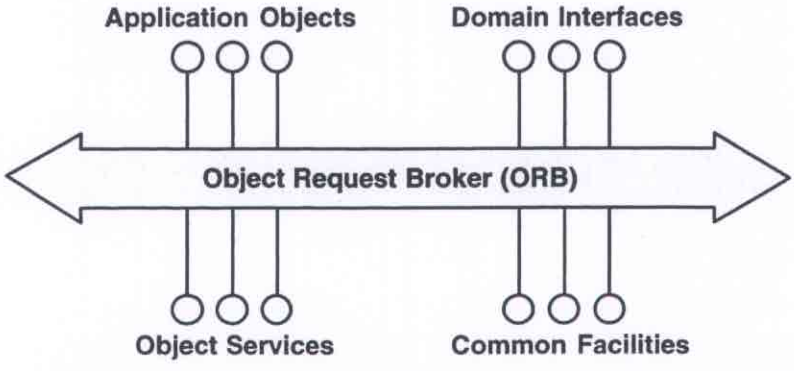
komponenter. F eks ved utviklingen av et distribuert og komponentbasert K2IS for Sjøforsvaret vil en gjennom bruk av en slik arkitektur sikre at komponenter utvikles ifølge bestemte retningslinjer, og at komponentene som introduseres i systemet blir en del av systemet ved at de er i stand til å interoperere med eksisterende komponenter.

OMA er en industristandardisert objektmodell for distribuerte objektsystemer som applikasjoner kan bygges over. OMA er todelt i den forstand at den fokuserer på generelle distribuerte objektsystemer i tillegg til at det er utviklet standard domenespesifikke grensesnitt for applikasjonsomgivelser som f eks finans, produksjon, telekommunikasjon, helsevesen, transport, elektronisk handel, o s v. F eks (som nevnt i kapittel 5.3) foreslår MACCIS bruk av komponentstandarder og metodikker som er lojale overfor OMA.

6.1 OMA

Utviklingen av OMA administreres av OMG. OMG er et ikke–kommersielt foretak som består av mer enn 700 industrimedlemmer. OMG har som målsetting å redusere kostnader, kompleksitet og tidsforbruk i f m utvikling av programvareapplikasjoner. Målet er et programvaremarked med interoperable, gjenbrukbare og portable programvarekomponenter basert på åpne standardiserte objektorienterte grensesnitt. Dette gjøres gjennom en "Object Management Arcitechture" og et omfattende standardiseringsarbeid på grensesnittspesifikasjoner.

OMA bygger på en "Core Object Model" som definerer konsepter som objekt– og ikke–objekttyper, operasjoner, signaturer, parametre, returverdier, grensesnitt, arv og subtyping. OMA er satt sammen av objekter definert ved et objektorientert grensesnitt. Implementasjonen av komponenter trenger ikke å være objektorientert. Interaksjon gjøres gjennom objektorienterte grensesnitt. Grensesnitt mot ikke–objektorienterte og "legacy"– systemer kan utvikles vha objektorienterte innpakninger eller adaptere. Objekter kan være både klienter som ber om tjenester, og tjenere som responderer på forespørsler.



Figur 6.1 OMGs "Object Management Architecture" (39)

OMA består av komponenter, grensesnitt og protokoller (figur 6.1) som deler opp et objektsystem i fem områder:

1. *"Object Request Broker" (ORB).* ORB er en kommunikasjonsinfrastruktur og har mekanismer som objekter kan benytte seg av for transparent å sende ut eller motta forespørsler og svar, og på denne måten sørger for interoperabilitet mellom applikasjoner på forskjellige maskiner i et heterogent, distribuert miljø.

   CORBA spesifiserer et standard grensesnitt mot ORB–komponenten ("ORB to ORB communication"). I OMA–spesifikasjonen defineres også et "Interface Descripting Language" (IDL). IDL er et programmeringspråknøytralt grensesnittspesifiseringsspråk med separate spesifiserte mappinger mot bl a C++, Smalltalk, Ada, COBOL og Java. "Object Services", "Common Facilities", "Domain Interfaces" og "Application Objects" har IDL–grensesnitt og kommuniserer med hverandre via ORBen.

2. *"Object Services".* Denne tjenesten bidrar med grunnleggende operasjoner for logisk modellering og fysisk lagring av objekter. Den definerer et sett av grunnleggende operasjoner som alle klasser bør implementere eller arve. Operasjonene skal være tilgjengelig gjennom ORB, men kan også gjøres tilgjengelig gjennom andre grensesnitt. Spesifikasjonene for "Object Services" består typisk av et sett av IDL grensesnitt–definisjoner og en beskrivelse av oppførsel og forespørselssekvens.

   De operasjonene "Object Services" bidrar med, er:

   - *"Class management"*. Opprette, endre, fjerne, kopiere, distribuere og beskrive klasser. Kontrollere definisjonen av klasser, grensesnittet til klasser og relasjoner mellom klasse–definisjoner
   - *"Instance management"*. Opprette, endre, fjerne, kopiere, flytte og kontrollere objekter og relasjoner mellom objekter
   - *"Storage"*. Permanent og transient lagerplass for objekter og deres tilstand
   - *"Integrity"*. Sikre konsistens og integritet mhp objekttilstand både innen enkle objekter og mellom objekter
   - *"Security"*. Definere og håndheve aksessbegrensninger på objekter og deres komponenter.
   - *"Query"*. Velge objekter eller klasser fra en implisitt eller eksplisitt identifisert samling av objekter eller klasser
   - *"Versions"*. Lagre, korrelere og administrere varianter av objekter

3. *"Common Facilities".* En samling støttefunksjoner som er nyttig i mange applikasjonsdomener og som er gjort tilgjengelig gjennom et grensesnitt som er samsvarende med OMA. Støtte for "Common Facilities" er vilkårlig. Ikke alle standardiserte støttefunksjoner vil være tilgjengelig på alle plattformer, men dersom de er tilgjengelige, skal de være i h t OMGs spesifikasjon. For at en tjeneste skal bli en "Common Facility", må den kommunisere vha ORB, implementere en støttefunksjon som OMG velger å adoptere og ha et OMA–samsvarende grensesnitt. "Common Facilities" blir gruppert i følgende kategorier:

- "User Interface Common Facilities"
- "Information Management Common Facilities"
- "System Management Common Facilities"
- "Task Management Common Facilities"
- "Vertical Market Facilities"

4. *"Domain Interfaces."* Standard domenespesifikke grensesnitt for applikasjonsdomener som f eks finans, produksjon, telekommunikasjon, helsevesen, transport, elektronisk handel etc. OMG har nå også etablert en C4I "Domain Special Interest Group" (C4I DSIG) for å etablere et grensesnitt og rammeverk for C4I–domenet. Arbeidet her er imidlertid kommet mye kortere enn for de andre domenene. OMG har nå ca 2/3 av sin totale innsats på å utvikle domenespesifikke grensesnitt.

   Forskjellen på "Common Facilities"–klasser og "Domain Interfaces"–klasser, er at "Common Facilities"–klasser representerer vanlig funksjonalitet (adoptert av OMG som standard) og "Domain Interfaces"–klasser representerer standard grensesnitt som er spesifikt for et applikasjonsdomene.

5. *"Application Objects."* Denne delen av arkitekturen omfatter de applikasjonene som utfører spesifikke oppgaver for sluttbrukerne og er ikke underlagt standardisering.

6.1.1 C4I DSIG

OMGs C4I DSIG har startet en prosess for å spesifisere domenespesifikke grensesnitt for systemer som inngår i "Command, Control, Communications, Computers, and Intelligence" (C4I) –domenet med utgangspunkt i krav, standarder, produkter og pågående arbeider innen dette området.

Målsettingen for C4I DSIG er å:

- arbeide innad i OMG for å utvide CORBA–teknologien der det er nødvendig for å oppfylle kravene fra C4I–miljøet
- arbeide for anvendelse av CORBA–teknologien i C4I–miljøet
- arbeide sammen med andre OMG–grupper for å sikre at OMG–definerte teknologier dekker C4I–kravene
- gi retningslinjer og støtte til C4I–miljøene mhp bruk av OMG–teknologier
- definere et CORBA–basert rammeverk for C4I–domene

OMGs grunnleggende oppgave er å etablere en arkitektur og et sett av spesifikasjoner for å muliggjøre distribuerte integrerte applikasjoner. Primære mål er gjenbrukbarhet, flyttbarhet og interoperablitet hos objektorienterte programvarekomponenter i et distribuert hete-

rogent miljø. En stor del av OMGs innsats har vært rettet inn mot det å etablere en infrastruktur basert på åpne og standard grensesnitt–definisjoner som muliggjør dette. I senere tid har OMG også startet en prosess for å standardisere felles grensesnitt innen applikasjonsdomener. C4I DSIG representerer et slikt domene og noen av områdene som forsøkes dekket er (23):

- kommando, kontroll og kommunikasjon
- rapportering og distribusjon
- situasjonsbevissthet, presentasjon og vurdering
- målfølging
- operativ planlegging
- støtte for distribuerte, samspillende prosesser og distribuert simulering
- informasjonshåndtering og datalagring
- håndtering av etterretningsdata
- data/informasjons–analyse, –anvendelse og –fusjon
- multinasjonale C4I–aktiviteter, f eks NATO
- "Battlefield Digitization"
- informasjonskrigføring
- logistikk
- CSCW.

6.2 Konklusjon referansearkitekturer

Utviklingen av et distribuert og komponentbasert K2IS for maritime operasjoner vil være avhengig av en arkitektur for å sikre at komponenter utvikles ifølge bestemte retningslinjer og at nye komponenter som introduseres i systemet blir en del av systemet ved at det er i stand til å interoperere med eksisterende komponenter. Det er viktig at arkitekturen for informasjonssystemet også fokuserer på interoperabilitet mot tilsvarende NATO–systemer og norske systemer. I den forbindelse anbefaler vi å beskrive en felles referansearkitektur som tilfredsstiller alle tre forsvarsgrenene i Norge. Dersom en felles referansearkitektur ikke er oppnåelig, vil det med stor sannsynlighet være en rekke fellestrekk som gjør at man bør koordinere arbeidene.

Foreløpig er det kun en del industriforetak som høster gevinst av OMA siden de over OMAs infrastrukturtjenester utvikler komponenter som gjenbrukes i flere systemer som de utvikler. Et marked av komponenter for K2IS tilsvarende komponentmarkedet for f eks helsesektoren, vil imidlertid ikke utvikle seg før en referansearkitektur er på plass. En kan

derfor ikke vente seg et slikt marked de første 2–4 årene. Man bør imidlertid følge med og delta på arbeidet i C4I DSIG i f m utvikling av en referansearkitektur.

7 PROGRAMVAREARKITEKTURER

En distribuert og komponentbasert systemløsning forutsetter en overordnet systemarkitektur som setter komponenter inn i en sammenheng, og sikrer at viktige militære systemegenskaper ivaretas ved (videre)utvikling og vedlikehold av komponenter i systemet. En systemarkitektur for et K2IS bør ta utgangspunkt i en referansearkitektur. En systemarkitektur inneholder beskrivelsen av *ett eller flere* programvaresystemer med sine egne spesifikke programvarearkitekturer.

Det finnes flere generelle typer av arkitekturer som programvaresystemer kan bygges over. Innenfor hvert av disse generelle arkitekturtypene finnes det ofte flere varianter. Tre generelle typer av programvarearkitekturer er:

- *"Mainframe arkitektur":* I en "mainframe"–programvarearkitektur er all system–intelligens lokalisert i en sentral tjenermaskin og brukere har interaksjon med tjener–maskinen via en enkel terminal.
- *Klient–tjener–arkitektur:* En klient–tjener–arkitektur er en generell spesifikasjon av et nettverkssystem der et klientprogram initierer kontakt med et separat tjenerprogram (som ofte kjører på en annen maskin) for å bruke en spesifikk funksjon eller tjeneste som tjeneren tilbyr. Moderne klient–tjener–arkitekturer utvikles i forhold til en *n–lags arkitektur* ved hjelp av distribuert objektteknologi. En n–lags arkitektur er en logisk lagdelt programvarearkitektur som fritt kan distribueres over et nettverk. Utvikling av klient–tjenersystemer ved bruk av distribuert objektteknologi tilbyr en rekke fordeler som maskin–, plattform– og språkinteroperabilitet samt større håndterbarhet og adaptabilitet i systemet. For eksempel kan ett og samme objekt både være en klient og en tjener.
- *Agentarkitekturer:* Agentarkitekturer bygges typisk over kommersielt tilgjengelige mellomvareteknologier. En viktig egenskap som skiller agenter fra objekter er at de har lokal kontroll og søker aktivt å nå de målene de er blitt tildelt. Det vil si at de er autonome og eksisterer uavhengig av sitt opphav. De er dermed velegnet for desentraliserte problemer.

7.1 Klient–tjener–arkitekturer

En klient–tjener arkitektur er en generell spesifikasjon av et nettverkssystem der et klientprogram initierer kontakt med et separat tjenerprogram (som ofte kjører på en annen maskin) for å bruke en spesifikk funksjon eller tjeneste som tjeneren tilbyr.

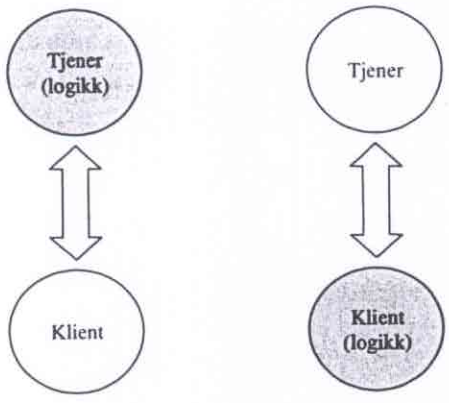
De første klient–tjenersystemene ble laget som en 2–lags arkitektur hvor klientprogrammene presenterte et grafisk grensesnitt til brukeren, og brukte brukerens datainput og handlin-

ger for å utføre forespørsler mot en databasetjener som kjørte på en annen maskin. I en 2–lags arkitektur er applikasjonslogikken typisk direkte bundet opp til klientprogrammet.

Moderne klient–tjenerimplementasjoner legger ofte til et tredje lag og oppnår dermed en 3–lags arkitektur. Generelt kan vi ha en n–lags arkitektur hvor applikasjonslogikken blir partisjonert.

7.1.1 2–lags arkitektur

Den mest vanlige implementasjonen av en 2–lags arkitektur er å plassere applikasjonslogikken i klienten, som gir en såkalt "fet" klient – "tynn" tjenerarkitektur. Databasetjeneren rapporter da bare spørringer tilbake til klienten og klienten prosesserer disse selv.



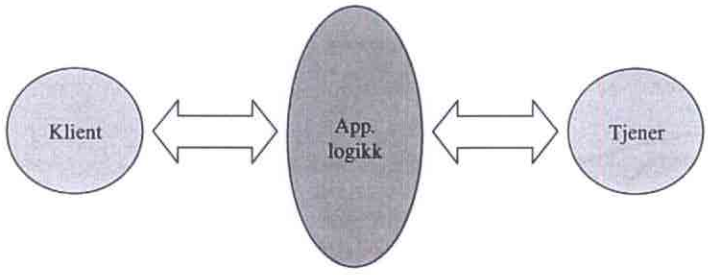
Figur 7.1 Alternative 2–lags akitekturer

En annet alternativ er å ha en "tynn" klient – "fet" tjenerarkitektur hvor klienten kaller på prosedyrer som er lagret hos databasetjeneren.

I begge tilfellene brukes "remote" (fjern) databasetransportprotokoller, som f.eks. SQL–Net ("Structured Query Language") og ODBC ("Open Database Connectivity"), for å utføre transaksjonen. Dette krever høy bruk av nettverket så utnyttelsen av båndbredde, og således antall brukere som kan utnytte nettverket, blir redusert. Ikke bare nettverkstransaksjonsstørrelse, men også forespørselstransaksjonshastighet blir redusert ved denne krevende interaksjonen.

7.1.2 3–lags arkitektur

En merkbar økning i ytelse vil være mulig med en n–lags klient–tjener–arkitektur. F eks kan et mellomliggende lag blir satt inn mellom en "tynn" klient og en "tynn" tjener, og dermed oppnå en 3–lags struktur. Klienten vil da ha interaksjon med det mellomliggende laget med standard protokoller som DLL ("Dynamic Link Library"), API ("Application Programming Interface") eller RPC ("Remote Procedure Call"). Det mellomliggende laget interagerer mot databasetjeneren via standard databaseprotokoller.

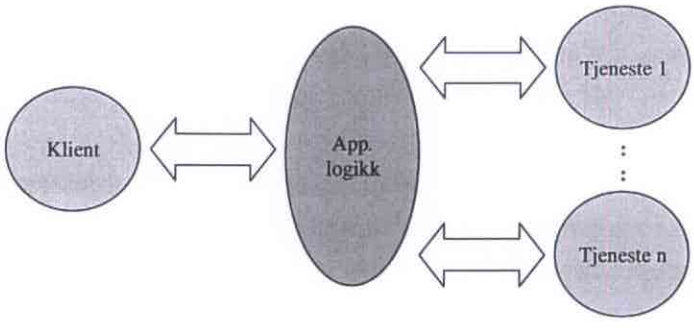


Figur 7.2 3–lags arkitektur

Det mellomliggende laget innholder mesteparten av applikasjonslogikken og oversetter klientforespørsler til databaseforespørsler og andre operasjoner, samt oversetter data fra databasen til klientdata. Hvis det midterste laget er plassert på samme maskin som databasen kan disse bli tett integrert og man kan oppnå sterk kontroll. Men det kan også bli distribuert på andre maskiner for å oppnå økning i prosesseringskapasitet.

7.1.3 N–lags arkitektur

En 3–lags arkitektur kan utvides til en n–lags arkitektur, hvor det midterste laget tilbyr forbindelser til flere typer av tjenester, integrerer og kopler dem opp mot klienten og til hverandre (figur 7.3).



Figur 7.3 n–lags arkitektur

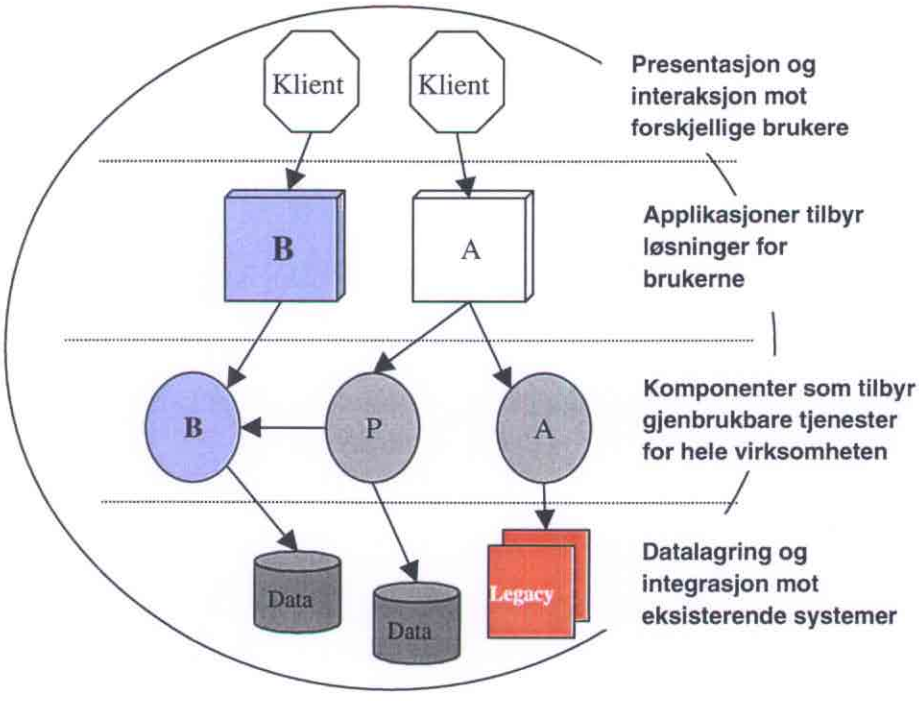
En n–lags struktur kan oppnås ved å skille ut domenelogikk fra applikasjonslogikk i egne lag (illustrert i figur 7.4). Disse lagene kan partisjoneres mellom flere maskiner i en distribuert løsning.

En applikasjon er et selvstendig program som utfører en spesifikk funksjon direkte overfor brukeren. En applikasjon består av tjenester som sluttbrukeren anvender for å løse et bestemt problem. En applikasjon eksisterer kun innenfor en systemkontekst.

7.1.4 Sammenlikning

En n–lags klient–tjener–arkitektur gir modulære gjenbrukbare tjenester over et nettverk som gir høyere ytelse og fordeler sammenliknet med den tradisjonelle 2–lags arkitekturen. Men det er verdt å merke seg at store distribuerte løsninger kan bli svært komplekse og

dette stiller større krav til "management". Det er også viktig at man kan teste en distribuert løsning for å kunne avdekke eventuelle flaskehalser og løse opp i disse.



Figur 7.4 Eksempel på en 4–lags arkitektur

7.2 Agentarkitekturer

Betydningen av agentbegrepet varierer fra enkle forhåndsprogrammerte reaktive objekter/ komponenter (kapittel 9.2) til autonome agenter som har sofistikerte evner som læring og planlegging, og som er proaktive. Begrepet ble opprinnelig lansert i kunstig intelligens miljøet, men har i dag en stor utbredelse i programvareutviklingsmiljøer. I disse miljøene benyttes det som en nyttig metafor i analyse, konstruksjon og implementasjon av såkalte agentbaserte applikasjoner. Agenter slik de oppfattes i disse miljøene har i forhold til tradisjonelle programvareobjekter følgende tilleggsegenskaper:

- Autonome. De eksisterer uavhengig av andre agenter
- Proaktive. De har egne mål og samarbeider med andre agenter for å nå disse målene
- Kommunikasjon. De benytter en felles terminologi i kommunikasjon med andre agenter. Agenter kan både tilby og anmode om tjenester. D v s en agent kan i en sammenheng være en klient og i en annen sammenheng være en tjener
- Kontroll er logisk og ofte geografisk distribuert
- Kan være mobile. Det vil si at de har evnen til å transportere seg selv i en infrastruktur. Mobile agenter reduserer behov for kommunikasjon. De kan potensielt gi mer robuste og feiltolerante systemer, og de kan sendes til andre steder i et nettverk og utføre ope-

rasjoner uavhengig av om maskinen de ble sendt fra er operativ eller ei. De kan også tilpasse seg endringer i nettverk med dynamisk rekonfigurering.

Agenter slik de oppfattes i kunstig intelligens miljøet har i tillegg følgende egenskaper:

- Har kunnskap og intensjoner
- Henter aktivt informasjon fra sine omgivelser
- Er målsøkende
- Resonnerer. Tolker situasjoner, tar beslutninger og planlegger
- Handler. Handler gjennom selv å påvirke sine omgivelser eller gjennom å sende informasjon til andre agenter
- Lærer
- Kommuniserer ved hjelp av et felles språk.

Agenters egenskaper varierer typisk fra applikasjon til applikasjon og er avhengig av kravene som settes.

Siden agenter er autonome må de koordinere sin oppførsel med andre agenter. Konflikter kan oppstå og det finnes et sett av konfliktløsningsstrategier som kan benyttes. Disse strategiene kan f eks benyttes for å løse opp konflikter som kan oppstå i en distribuert bildeoppbyggingsprosess.

Agentarkitekturer kan deles inn i mikro– og makronivå. Mikronivå dekker arkitekturen til den individuelle agent og makronivå dekker arkitekturen til en samling av agenter som til sammen danner et multi–agent–system. På mikronivå er typisk følgende arkitekturer blitt benyttet:

- "Blackboard"
- "Actor" eller objektorientert
- Produksjonssystemer.

På makronivå er agentene knyttet til hverandre via kommunikasjonsmekanismer og interaksjonsprotokoller. Kommunikasjonsmekanismer er typisk meldingsbaserte. Alternativt baseres de på direkte tilgang til en global datastruktur, for eksempel et "blackboard". Blackboarden fungerer da som en slags felles "oppslagstavle" for alle agentene i systemet.

En tavle er en datastruktur som deles av flere agenter og benyttes for å koordinere problemløsningsprosessen som er distribuert mellom agentene. Hver enkelt agent løser et delproblem og legger resultatet på tavla. Eksekveringen av en agent trigges når informasjonen som legges på tavla av en annen agent oppfyller spesifiserte kriterier. En tavle kan dermed benyttes for å støtte asynkron informasjonsutveksling mellom agenter. Blackboard–tilnærmingen ble først benyttet i HERSEY II som var et taletolkningsprosjekt (37).

Agenter kan defineres på forskjellige detaljnivå, fra et enkelt objekt eller en regel til et komplett konvensjonelt eller kunnskapsbasert system med en innpakning som gjør at det for et eksternt system fortoner seg som en agent.

Typen arkitektur som benyttes, er i stor grad avhengig av om agentene er proaktive og tilpasningsdyktige eller reaktive med forhåndsbestemt oppførsel. Egenskapene til agentene er igjen styrt av hvilke krav som settes til systemet. For at agentene i et multiagent system skal arbeide effektivt mot et felles mål eller mot separate individuelle mål som er internt avhengige, er det nødvendig at de utveksler kunnskap, planer og mål. I denne sammenheng er det viktig at agenter har kunnskap om andre agenters intensjoner og behov for informasjon.

Designprosessen for distribuerte multiagentsystemer er kort fortalt som følger (38):

- Gitt et sett av mål som skal nås eller oppgaver som skal utføres av en samling agenter og et sett av designvariable som for eksempel:
  - arkitekturtype
  - koordineringsmekanismer
  - kommunikasjonsmekanismer og interaksjonsprotokoller
  - oppsplitting av oppgaver og tildeling av disse til agenter
  - organisasjonsstruktur
  - konfliktløsningsstrategi,

  finn passende verdier for designvariablene som overholder båndbreddebegrensninger og andre føringer.

8 ARKITEKTURER PÅVIRKER SYSTEMKVALITETER

Krav til kvalitet (eller ønskede egenskaper) er et meget sentralt og viktig aspekt ved utforming av informasjonssystemer. Hvilke egenskaper som prioriteres avhenger av hvilket behov man ønsker systemet skal dekke og hvilket fokus man har (d v s i hvilken sammenheng systemet skal virke). Prioriterte egenskaper påvirker valg av programvarearkitekturer for distribuerte løsninger.

8.1 Funksjonelle og ikke–funksjonelle krav

I en systemutviklingsprosess vil en analyse– og konseptfase innvolvere identifisering av funksjonelle og ikke–funksjonelle krav til et system. Funksjonelle krav relaterer seg til hvilke konkrete funksjoner eller tjenester som systemet skal tilby brukerne. Ikke–funksjonelle krav handler om *kvaliteter* eller *egenskaper* ved systemet. Ikke–funksjonelle krav lar seg ikke så lett implementere som konkrete og målbare deler av et system. Systemegenska-

per av denne type er heller mer "globale" og "overalt nærværende" i den forstand at de *karakteriserer* hele systemet. Ikke–funksjonelle krav har derfor en stor påvirkning for hvilke arkitekturer som velges.

De mest typiske ikke–funksjonelle kravene som ofte leder til valg av arkitekturer for distribuerte løsninger er krav til (35):

- skalerbarhet
- åpenhet
- heterogenitet
- ressursdeling
- feiltoleranse.

Dette er krav som øker kompleksiteten til systemer. Fordelen ved sentraliserte systemer er ofte at utviklingen av disse er enklere og derfor også ofte billigere enn ved utvikling av distribuerte systemer. På den måten kan man si at det ikke nødvendigvis alltid er mest gunstig å utvikle distribuerte systemer. Det er imidlertid slik at mange av de egenskaper vi ønsker skal karakterisere systemer i dag, ikke kan oppnås gjennom sentraliserte systemer. Mer om distribuerte systemer og egenskaper ved disse er gitt i kapittel 9.2.

I prosjektets konseptarbeider er det utledet noen kvaliteter som anses som kritiske for et K2IS. Disse presenteres i det følgende.

8.2 Kvaliteter ved et fremtidig K2IS

Egenskapene identifisert i prosjektets konseptarbeider, er av både dynamisk og statisk karakter. Dynamiske egenskaper kan kun oppfattes ved "run–time", eller operativ bruk, fordi de er relatert til den dynamiske oppførselen av systemet under kjøring. Egenskaper som går på selve designstrukturen, kan sies å være statiske. Vi vil kunne observere statiske egenskaper ved å se systemet ut fra et utviklings– og forvaltningsperspektiv. *Utviklings– og forvaltningsegenskaper* er altså egenskaper relatert til designstrukturer, utvikling og vedlikehold. *Kjøretidsegenskaper* er egenskaper relatert til den dynamiske oppførselen av et informasjonssystem.

8.2.1 Utviklings– og forvaltningsegenskaper

Utviklings– og forvaltningsegenskapene fokuserer på designstrukturer, utvikling og vedlikehold. Dette er viktige kvaliteter som angår utviklere og vedlikeholdere av informasjonssystemet. Et K2IS må være realiserbart i forhold til de krav som stilles og må fungere i den tiltenkte rollen som en del av et ledelsessystem.

- **Modifiserbarhet:** Modifiserbarhet ser på endringsevnen til systemets struktur, og vil være viktig for levedyktighet og evolusjon av et K2IS over tid. Det er viktig å spørre

seg om systemets struktur er slik at utvidelser, reduksjoner og påbygninger enkelt og effektivt lar seg gjøre? Stadig nye krav og brukssituasjoner vil føre til at man hele tiden må regne med å gjøre endringer i informasjonssystemet. Målet er da å begrense omfanget av de forandringer som må gjøres. I et distibuert system vil en godt gjennomtenkt separering av funksjonalitet og teknologi, samt en god fordeling og organisering av komponenter være en fornuftig måte å innrette seg på. En slik inndeling, samt det å vektlegge bruk av standardiserte grensesnittspesifikasjoner (gjennom bruk av objektorientert mellomvare), vil være viktige hjelpemidler for å oppnå et slikt mål.

- **Gjenbrukbarhet:** Et mål på gjenbrukbarhet er at det finnes implementasjons- og designelementer i systemet som vil være gode kandidater for gjenbruk i andre systemer. I denne sammenheng vil bruk av åpne standarder være fordelaktig. Gjenbrukbarhet og modifiserbarhet kan sees i sammenheng. Systemer som er designet med tanke på modifiserbarhet, vil lettere tilpasses nye krav og funksjonalitet og kan dermed også gjenbrukes i andre sammenhenger.

- **Testbarhet:** Hvor enkelt er det å identifisere defekter i systemet ved simuleringer og testdata? Formålet med testbarhet er å kunne verifisere systemet eller deler av systemet på en enkel tilgjengelig måte. Systemet må kunne vise at det fungerer i henhold til gitt spesifikasjon. Eventuelle feil som avdekkes under uttesting, må rettes opp. For at testingen skal være gjennomførbar, kreves det at spesifikasjonen må være klar og presis slik at misforståelser kan unngås.

- **Konfigurerbarhet:** Et konfigurerbart system i denne sammenheng betyr at systemet vil kunne tilby funksjoner (eller tjenester) i form av separate, påbyggende komponenter. Dette gir større fleksibilitet ved at man kan kjøre en konfigurasjon av komponenter som tilsammen tilbyr nødvendig og ønsket funksjonalitet. På denne måten vil man unngå installasjoner av store og komplekse systemer hvor man kun tar i bruk deler av systemet. En høy konfigurerbarhet tillater en å installere nøyaktig kun de komponentene man trenger.

8.2.2 Kjøretidsegenskaper

Med kjøretidsegenskaper fokuseres det på den dynamiske oppførselen av et informasjonssystem.

- **Pålitelighet:** Med pålitelighet menes den tiden systemet er operativt og fungerende, målt i "Mean Time Before Failure" (MTBF). Høy systemtilgjengelighet stiller krav til oppetid og feiltoleranse.

- **Tilgjengelighet:** Tilgjengelighet kan oppfattes på to måter:

  1. *Tilgjengelighet av informasjon:* Brukerne av informasjonssystemet må ha tilgang til informasjonen som ligger i systemet. Det er viktig at rett bruker får rett informasjon til rett tid.

2. *Tilgjengelighet av funksjonalitet:* Brukerne av systemet må ha tilgang til de nødvendige funksjonene, bestemt ut ifra behov, som informasjonssystemet tilbyr for å kunne understøtte ledelse.

- **Ytelse:** Informasjonssystemet bør kunne respondere i sann tid, d v s prosessere og gi nødvendig informasjon innenfor spesifiserte tidsrammer. De spesifiserte tidsrammene vil være situasjonsavhengige. Følgelig må informasjonssystemet støtte krav til og endringer i tidsrespons. Dette stiller krav til tilpasningsevne og effektivitet av flyt og prosessering av data. Ofte vil det være en avveining ("trade–off") mellom tidsbruk og kvalitet på informasjonen for tidskrevende/–kritiske applikasjoner. Det er bedre med en tidsnok ikke–optimal løsning enn en for sen optimal løsning.
- **Tilpasningsevne:** Ved å se på informasjonssystemet i stort (ytre egenskap) vil tilpasningsevne være en egenskap som går både på struktur og prosess. Informasjonssystemet må, på tross av økt heterogenitet, både på informasjonsinnhold, realisering og tilhørende ledelsesprosesser, kunne støtte endringer i ledelsesorganisasjonen. Denne tilpasningsevne som egenskap må også gjenfinnes som en indre egenskap ved systemet – som f eks gjennom dynamisk tilpasningsdyktighet til programvarekomponenter.
- **Robusthet:** Robusthet er systemets motstandsdyktighet overfor ytre påvirkninger. Hvis deler av systemet faller ut, bør ikke dette sette ut andre viktige deler av systemet som vil kunne fungere uavhengig. Informasjonssystemet bør kunne tåle at deler av det faller ut og samtidig opprettholde nødvendig tilgjengelighet av informasjon og tjenester for å understøtte ledelse, dog kanskje med en dårligere kvalitet.
- **Mobilitet:** Mobilitet klassifiseres i forhold til a) Kategori av bruker (en bruker kan være en terminal, en person eller en applikasjon) og b) Tilgjengelighet av tjenester:

a) Kategori av bruker:

   - *Terminal mobilitet* tillater en terminal å flytte lokasjon mens tjenestene opprettholdes
   - *Personlig mobilitet* tillater en person å aksessere eller bli aksessert av nettverket uavhengig av aksesspunkt eller hvor terminalbrukeren befinner seg i nettverket
   - *Applikasjonsmobilitet* tillater en programvarekomponent å flytte fra en maskin til en annen
   - *Sesjonsmobilitet* blir definert som en tilleggsfunksjon for de tre typene mobilitet nevnt ovenfor. Denne mobiliteten sørger for at aktive sesjoner ikke blir avbrutt mens terminaler, personer eller applikasjoner flytter seg.

b) Tilgjengelighet av tjenester:

   - *Kontinuerlig mobilitet* betyr kontinuerlig tilgjengelighet av tjenester mens brukeren er på flyttefot
   - *Diskret mobilitet* betyr at tilgjengeligheten av tjenester er avgrenset til bestemte dekningsområder eller aksesspunkter.

- **Skalerbarhet:** Er systemet skalerbart i forhold til å håndtere økning i bruk, som f eks større datavolum, flere brukere og økt frekvens av forespørsler? Skalerbarhet oppnås primært ved replikering av ressurser (både i maskinvare og programvareprosesser).

- **Koordinering:** Koordinering kan defineres som en prosess som har som målsetting å optimalisere den samlede effekten til en samling komponenter. Komponentene kan sies å være koordinerte dersom deres samlede ytelse er i henhold til spesifiserte ytelseskrav (f eks kvalitet på løsninger eller tid). Koordinering mellom komponenter er fokusert på hvem gjør hva, når og med hvem.

- **Synkronisering:** Distribuerte komponenter som er delaktige i en operasjon samarbeider og utveksler informasjon. Det eksisterer implisitte og eksplisitte skranker på hvordan komponenter eksekverer relatert til hverandre. Disse skrankene blir sikret og utført av forskjellige typer for synkronisering.

- **Brukervennlighet:** Brukervennlighet er effekten, effektiviteten og tilfredsstillelsen som brukeren oppfatter ved å bruke systemet til å utføre sine oppgaver. Hvordan understøtter informasjonssystemet ledelsesprosessen? Er menneske–maskin– grensesnittet utformet slik at det assisterer og støtter brukeren i å oppnå sine mål?

- **Sikkerhet:** Hvordan forhindre uautorisert bruk og misbruk av systemet? For et K2IS er det viktig at brukeren kan stole på informasjonen som ligger der, og at overføring av gradert informasjonen ikke kan snappes opp. Stallings (9) definerer fire kategorier av angrep som fokuserer på overføring av informasjon:

  1. *Avbrytelse:* Informasjonen blir ødelagt, utilgjengelig eller ubrukelig
  2. *Oppsnapping:* En uautorisert part får tilgang til informasjon
  3. *Modifisering:* En uautorisert part får aksess og modifiserer informasjon
  4. *Fabrikkering:* En uautorisert part setter inn falsk, fabrikkert informasjon.

  Informasjonssystemet bør bruke velkjente og for Forsvaret godkjente sikkerhetsarkitekturer og mekanismer.

9 KOMPONENTBASERT SYSTEMUTVIKLING

Inkrementell og komponentbasert systemutvikling blir en meget sentral utfordring fremover. Ønsket om kortere utviklingstider og økt kvalitet vil kunne oppfylles gjennom økt gjenbruk i form av mer standardiserte og dels distribuerte komponenter. Selv om teknologier for komponentbaserte systemer i dag kan sies å være relativt godt etablert, er imidlertid kunnskapen, erfaringen og metodene for å utvikle dem fortsatt begrenset. I dag snakker man om 9 viktige prinsipper for moderne systemutvikling (25):

1. *Arkitekturorientert tilnærming*. Det betyr at fokus for utvikling først legges på systemets grunnstrukturer, d v s de store trekkene. Deretter ser man på detaljene.

2. *Iterativ livssyklusprosess:* En bearbeidingsstrategi hvor tid blir satt til side for å revidere og forbedre deler av systemet. Man itererer over utviklingsfaser og modeller.

3. *Modellbasert utvikling:* D v s hele utviklingen av systemet drives på basis av de modellene som utvikles (virksomhetsmodell, analysemodell, designmodell, implementasjonsmodell, o s v). Det mest kjente og brukte modelleringsspråk er UML.

4. *Komponentbasert utvikling:* Metodikker som støtter komponenttankegangen i systemutvikling.

5. *Endringsorientert miljø:* D v s fokus på endringer og vedlikehold av systemene.

6. *"Round–trip" engineering:* At "alle nivåer henger sammen" (fra virksomhetsnivå ned til implementasjon– og testnivå). Denne sammenhengen beskrives gjennom modellene.

7. *Objektiv kvalitetskontroll:* Testing av produkter (ofte på basis av modellene).

8. *Økende detaljeringsgrad:* Kontinuerlig forfining av modellene (iterativ).

9. *Konfigurerbar prosess:* At man kan endre systemutviklingsprosessen underveis. F eks introdusere nye roller, nye teknikker, o s v.

Bruk av komponentbasert teknologi forutsetter bruk av arkitekturer. Målsettingen innenfor utviklingen av programvarearkitekturer for distribuerte systemer, er å gjøre konfigureringen av større programvaresystemer ut fra enkeltkomponenter mer presist og effektivt. I den forbindelse er det viktig å kjenne til hvordan framtidige distribuerte informasjonssystemer vil kunne bygges. Dette er viktig både med tanke på egenutvikling og for å få en forståelse av hvilken fleksibilitet man får overfor eventuelle komponentleverandører.

I dette kapittelet presenteres først noen sentrale systemutviklingsperspektiver. Deretter gis en kort orientering om hva som ligger i begrepet systemutviklingsmetodikker, hvor bl a sammenhengen mellom arkitektur og metodikk er beskrevet. Til slutt presenteres to kjente objektorienterte og komponentbaserte utviklingsmetodikker som brukes i dag. Felles for disse er at de er modelldrevet og brukersentrert gjennom bruk av UML.

9.1 Objektorientert systemutviklingsperspektiv

Ved utvikling av informasjonssystemer har man tidligere fokusert på funksjonsorientering og/eller dataorientering. Det siste tiåret har det imidlertid skjedd et paradigmeskifte hvor den objektorienterte tankegangen er blitt rådende. Funksjonsorienterte, dataorienterte og objektorienterte systemutviklingsperspektiver representerer grunnleggende forskjeller i måten å "betrakte verden" på i forhold til hvordan man modellerer og utvikler et system.

Funksjonsorientert hentyder til en tradisjonell måte å tenke på innenfor systemutvikling. Ved et funksjonsorientert fokus ser man på hvilke funksjoner (arbeidsoppgaver) som blir utført i organisasjonen og modellerer en løsning som en sekvens med funksjoner. Denne måten å utvikle systemer er effektiv, men resulterer ofte i systemer med liten fleksibilitet.

F eks dersom kravene til systemet endres, må det ofte gjøres omfattende restruktureringer av hele systemet.

Dataorientering fokuserer på hva slags informasjon som representeres og lagres i systemet. Fokuset er på datamodellering. Denne måten å tenke på er typisk innenfor tradisjonell databaseutvikling, hvor dataene står i sentrum og hvor funksjoner eller applikasjoner sees på som behandlere av data i databasen.

Objektorientering refererer til en måte å betrakte verden på som er enklere og mer intuitiv for mennesker å forstå. I motsetning til funksjonsorientering og dataorientering betrakter man i objektorientering data og operasjoner som likeverdige. Man fokuserer på hva slags informasjon som skal representeres og hva som skal kunne gjøres med denne informasjonen.

Objekter er abstraksjoner som representerer entiteter i den virkelige verdenen eller systemet som utvikles. Et objekt er innkapslet og består av data og operasjoner. Et objekt spesifiseres ved attributter og metoder. Attributter inneholder data og metoder er operasjoner som utføres på attributter. Objekter er manipulerbare og gir en større fleksibilitet sammenliknet med tradisjonell utvikling.

Objektorientering har en rekke fordeler sammenliknet med tradisjonell funksjonsorientert og dataorientert systemtenkning. Objekter gir meningsfulle abstraksjoner for å representere virkelige entiteter. Innkapsling av informasjon og operasjoner i objekter gir oss et verktøy for bedre å være i stand til å håndtere kompleksitet og forandringer.

Objektorientert systemutvikling tillater en evolusjonær systemdesign og støtter en inkrementell utvikling hvor deler av systemet kan bygges til forskjellig tid og hastighet. I motsetning til funksjonsorientering og dataorientering, er objektorientering en måte å tenke på som kan brukes for å beskrive alle aspekter av et system med det samme begrepsapparatet (fra organisatorisk bruksperspektiv helt ned til konkrete implementasjonsmodeller).

9.2 Komponentbasert systemutviklingsperspektiv

I de siste årene har fokuset skiftet over på komponentbasert utvikling. Komponentbasert utvikling bygger videre på prinsippene fra objektorientering, men mens objekter er interne systemspesifikke deler, er komponenter uavhengige (autonome), kjørbare systemdeler som tilbyr et veldefinert sett av tjenester. En programvarekomponent defineres ved (14):

En programvarekomponent er en identifiserbar programvaredel som beskriver og/eller leverer en meningsfylt tjeneste som bare er tilgjengelig via et veldefinert grensesnitt.

Karakteristikker ved programvarekomponenter:

- *Identifiserbar:* En komponent har en klar identitet.
- *Sporbar*: En komponent ivaretar sin identitet og er sporbar når den brukes i en annen komponent eller en applikasjon.

- *Erstattbar:* En komponent kan erstattes av en ny versjon eller annen komponent som tilbyr de samme tjenestene uten at funksjonalitet i applikasjonen berøres.
- *Aksesserbar kun gjennom grensesnitt*: Komponenter aksesseres kun gjennom veldefinerte grensesnitt som er uavhengige av den underliggende implementasjon.
- *Uforanderlige grensesnitt*: Selv om implementasjonen av komponenten kan forandres, må ikke tjenestene som komponenten tilbyr, forandres.
- *Dokumentert tjeneste*: For å kunne gjenbruke komponenter, er det nødvendig at grensesnittet beskrives slik at man får en forståelse for hvilke tjenester som tilbys og hvordan disse brukes.

Komponenter er ofte gjenbrukbare og kan brukes som byggesteiner for å bygge og konfigurere et system. Det gjøres ved at tjenestene (funksjonene) i et komponentbasert system kan tilbys i form av separate påbyggende inkrementer (komponenter).

Ved utvikling av komponentbaserte systemer bør det vurderes bruk av tilgjengelige COTS–komponenter der hvor dette er hensiktsmessig. Utvikling med tanke på gjenbruk forutsetter en høyere initiell kostnad, men vil føre til kostnadsbesparelser på sikt ved at komponenter systematisk gjenbrukes hvor dette er mulig. Eksempler på etablerte komponenteknologier i dag er Microsoft's ActiveX components, Distributed Component Object Model (DCOM/COM+), CORBA og Enterprise Java Beans.

9.2.1 "Nivåer" av komponenter

Komponenter, som nevnt ovenfor, aksesseres kun gjennom veldefinerte grensesnitt som er uavhengig av den underliggende implementasjon. F eks er en "skrollbar", eller et annet Graphical User Interface (GUI)–objekt, en programvarekomponent dersom det har et veldefinert grensesnitt. Slike GUI–komponenter blir ofte omtalt som "små–skala"–komponenter og brukes ofte i utvikling av brukergrensesnitt innenfor samme applikasjon.

På "mellom–skala" nivå kan komponenter, som kommuniserer med hverandre, utvikles i forskjellige språk og hver komponent kan kjøres (eksekveres) på hver sin maskin. CORBA er et eksempel på en teknologi brukt på dette nivået. På "mellom–skala" nivå blir bl a valg av programvarearkitekturer, mellomvare og informasjonsbehandlingsmekanismer (f eks tjenester som transaksjons– og databasetjenester) viktig. Disse setter føringer for hvordan komponenter på dette nivået implementeres og passes inn i sammenhengen.

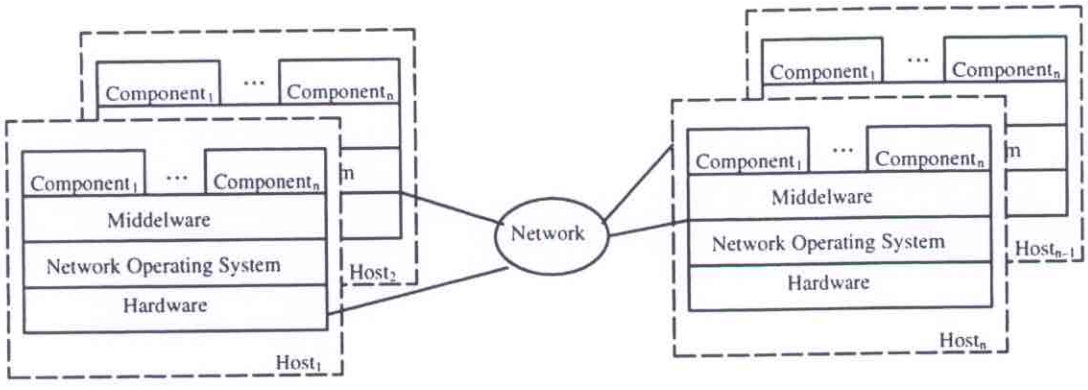
På såkalt "stor–skala" nivå snakker man om komponenter som støtter bestemte organisasjonsroller. En organisasjonsrolle kan være en rolle som innehas av et bestemt individ eller en avdeling, med ansvar for en bestemt funksjon.

9.2.2 Moderne komponentbaserte distribuerte systemer

Et distribuert system er et system hvor deler av systemet ofte fordeles (distribueres) på flere maskiner som kan være spredd geografisk. I et komponentbasert systemutviklingsperspektiv kan et distribuert system beskrives slik (35):

A distributed system consists of components on networked hosts that interact via middleware so that they appear as an integrated facility.

I figur 9.1 vises et distribuert system som en samling av tjenermaskiner som står i forbindelse med hverandre gjennom et nettverk. På hver maskin kjøres komponenter som betjenes av en mellomvare. Mellomvaren gjør komponenter i stand til å koordinere aktiviteter på en slik måte at en bruker oppfatter systemet som ett enhetlig integrert system.



Figur 9.1 Prinsippmodell av et distribuert system (35)

Distribuerte systemer har en rekke fordeler sammenliknet med sentraliserte systemer. F eks ved utvikling av et sentralisert system benyttes ofte homogen teknologi (samme maskinvare, programvareplattform, programmeringsspråk, o s v). Distribuerte systemer er typisk tilrettelagt for, og satt sammen av, heterogen teknologi. Et sentralisert system kan også settes sammen av flere deler. Imidlertid er disse delene, som ofte da er klasser i et objektorientert språk, ikke–autonome. Det betyr at systemet har full kontroll over disse til enhver tid. Konsekvensen er at alt i et sentralisert system blir aktivert når systemet starter, og tilsvarende hvis systemet deaktiveres eller slettes, deaktiveres eller slettes alt i systemet også. I et distribuert system vil deler av systemet kunne slås av uten at dette nødvendigvis influere på de øvrige delene av systemet.

En annen fordel er at i et distribuert system håndteres f eks skalerbarhet primært ved påbygging av flere tjenermaskiner sammen med flere komponenter. I denne sammenheng snakker man om en skalerbar arkitektur. Et konkret eksempel på dette er utbredelsen av internett. Internett har en skalerbar arkitektur fordi det kan enkelt bygges ut uten at det får konsekvenser for resten. En arkitektur sies generelt å være skalerbar dersom den kan håndtere en eventuell fremtidig økning i belastning ("load"), enten den er forventet eller ikke. I et distribuert system vil prosesseringer også kunne gjøres lokalt. Det reduserer dermed behovet for overføringskapasitet, og man kan bygge robuste systemer med lav sårbarhet og høy effektivitet.

Målet med åpen og distribuert prosessering er altså å muliggjøre interaksjon med tjenester fra hvor som helst i en distribuert omgivelse uten tanke på den underliggende *heterogene* omgivelse. Likevel, det er utfordringer knyttet til distribuerte systemer:

- Upålitelig kommunikasjon: tap av forbindelse og meldinger.
- Usikker kommunikasjon: mulighet for uautorisert avlytting og modifikasjon av meldinger.

En bruker av et distribuert system kan være en person eller en applikasjon. Et distribuert system karakteriseres typisk ved seks egenskaper (10):

- *Ressursdeling*: Muligheten til å benytte tilgjengelig maskinvare, programvare eller data hvor som helst i systemet (flere personer kan benytte de samme ressursene).
- *Åpenhet*: Betyr at systemet enkelt kan utvides og modifiseres (f eks å legge til nye programvarekomponenter). Åpenhet oppnås gjennom kommunikasjon via veldefinerte grensesnitt.
- *Samtidighet*: Flere oppgaver kan utføres samtidig (fordi komponenter kjøres i samtidige prosesser).
- *Skalerbarhet*: Et system kan tilpasses for å håndtere flere brukere (personer eller applikasjoner) og flere samtidige oppgaver uten at f eks svartider reduseres.
- *Feiltoleranse*: En feil i en del av et distribuert system bør ikke påvirke de resterende delene i systemet.
- *Transparens*: Underliggende distribusjonsmekanismer, som f eks ressursdeling og skalerbarhet, er skjult for brukeren. Det betyr at et distribuert system oppfattes av brukere som *en* enhet i stedet for en samling av samarbeidende komponenter. Transparens skjuler konsekvensen av distribusjon. Dette oppnås primært ved hjelp av objektorientert mellomvare.

Tradisjonelle distribuerte systemer har i stor grad fulgt en 2– eller 3–lags klient–tjener–arkitekturer. Moderne distribuerte systemer følger ofte en n–lags arkitektur hvor distribuerte objekter og komponenter inngår (beskrevet i kapittel 7).

9.2.2.1 Distribuerte objekter

Et objekt har følgende karakteristikker:

- En *klasse* brukes for å beskrive et sett av like objekter og definerer attributtene og metodene. Et objekt er en instans av en klasse.
- Objekter kommuniserer med hverandre gjennom *meldinger*. En melding påkaller (invokerer) en metode på et objekt.
- *Innkapsling* refererer til det å kapsle inn attributter og implementasjonen av metodene i objekter.
- *Polymorfisme* er den egenskapen at ulike objekter kan reagere forskjellige på den samme meldingen.

- *Arv* er en mekanisme som tillater at utvalgte fellesdeler av bestemte klasser kan deles.

Distribuerte objektmodeller og verktøy (f eks objektorientert mellomvare) er slik at objekter kan fordeles på forskjellige maskiner i et nettverk, hvor de kan inngå i lokale applikasjoner, men samtidig være en del av et enhetlig system. CORBA innehar en slik distribuert objektmodell og er en teknologi for distribuerte objekter. Fordeler ved en slik distribuert objektteknologi inkluderer bl a:

a) Mulighet til å distribuere objekter i et system til maskiner som best egner seg for spesifikke gjøremål uten å gjøre forandringer i systemet som bruker disse objektene.

b) Forenkler systemintegrasjon. Den overordnede tekniske målsettingen ved distribuerte objektsystemer er å "hjelpe frem" distribuerte informasjonsteknologier slik at de er mer effektive og mer fleksible, og samtidig mindre komplekse.

9.3 Organisasjonperspektiv i systemutvikling

Det er en økt bevisstgjøring rundt ulike brukeraspekter og organisasjonsaspekter i all IS–utvikling i dag. Utfordringene er store spesielt fordi disse aspektene er såpass komplekse og sammensatte at de ikke lar seg så lett formalisere og modellere. Teknologi og systemutvikling i organisasjoner er blitt et tverrfaglig forskningsområde som er i stor utvikling. Dette gjenspeiler seg i mye av den nyere faglitteraturen på området. I dag er det ikke tilstrekkelig bare å lære om teknologien, fordi det er ikke bare funksjonaliteten som bestemmer hvordan en gitt teknologi eller system blir brukt. Dette er viktig, fordi *hvordan* en gitt teknologi blir brukt innvirker på virksomheten.

Forskjellige synsvinkler og perspektiver på systemutvikling er:

a) Forandring av organisasjoner ved hjelp av informasjonsteknologi (Ledelses–/organisasjonsperspektiv)

b) Utvikle og innføre datastøtte i arbeidet (sluttbruker–/arbeidsperspektiv)

c) Designe og realisere tekniske løsninger (ingeniør/konstruksjonsperspektiv).

Innføring og bruk av teknologi i organisasjoner medfører forandring. Innenfor moderne systemutvikling i organisasjoner betraktes derfor systemutvikling og organisasjonsutvikling som to sider av samme sak. I organisasjoner som f eks har krav til samarbeid og samhandling, er endringsfokusert, og/eller hvor teknologi skal spille en strategisk rolle bør det skapes organisasjonsendringsprosesser hvor design og utvikling av teknologi blir en naturlig integrert del. Spesielt i f m teknologistøtte til sentrale virksomhetsfunksjoner vil dette være viktig. Innenfor moderne systemutvikling er det derfor mye større variasjon i arbeidsarenaer og nivåer for systemutviklere nå enn før:

- systemutviklere som tekniske rådgivere

- systemutviklere som har kunnskaper om, og griper inn i sosiale og organisatoriske prosesser
- systemutviklere som har oversikt og kontroll over helheten i informasjonssystemet.

Spesielle utfordringer innenfor systemutvikling i organisasjoner er:

- produkter som integreres med eksisterende systemer, og generasjoner av teknologi som må spille sammen,
- brukerbegrepet utvides ved at brukere er både individ og gruppe (og organisasjon), d v s det opereres med flere klasser av brukere,
- økt fokus på samhandling og informasjonsdeling som prinsipper for organisering av arbeidet
- økt kunnskap om det sammenvevde mønsteret av teknologiske og sosiale forhold som omslutter hele systemutviklingsprosessen (design, innføring og bruk),
- metoder og verktøy som støtter opp om de aspekter nevnt ovenfor.

Det er nødvendig med økt *flerperspektivisk* systemutvikling, d v s å sikre at flere sider ved et system blir belyst og lagt til grunn ved utvikling. Organisasjonsperspektivet er et viktig perspektiv i denne sammenheng. Flerperspektivisk utvikling er bl a det arkitekturer kan bidra til gjennom sine såkalte arkitekturperspektiver (kapittel 3).

9.4 Systemutviklingsmetodikk

Generelt er begrepet metodikk for de fleste mest kjent som læren om metoder innenfor et visst arbeidsområde eller fagfelt. Slik begrepet metodikk ofte blir brukt i moderne systemutvikling og i denne rapporten er basert på definisjonen gitt i (26):

Methodology: *A series of related methods or techniques.*

I systemutvikling kan man i stort si at en systemutviklingsmetodikk handler om hvordan en organisasjon produserer og leverer et system. En metodikk omfatter bl a sentrale elementer som mennesker, roller, kvalifikasjoner, aktiviteter, teknikker, verktøy, kvalitetsmål, leveranser (produkter) og standarder (26). En metodikk i stort omfatter:

- *Organisasjon og mennesker*: Enkeltindividenes roller og kvalifikasjoner er viktige. Normer og verdier (kulturen) i bedriften spiller en rolle der gruppesamarbeid og gruppeoppgaver står sentralt. Hvordan mennesker kommuniserer og hvordan de velger å organisere arbeidet kan variere fra prosjekt til prosjekt avhengig av ressurser og utviklingsmål.
- *Utviklingsmodell:* Utviklingsmodeller i systemutvikling beskriver hva en utviklingsprosess skal omfatte og prinsipper for hvordan arbeidet skal organiseres. Modellene gir

retningslinjer for hvilke trinn/faser eller løp arbeidet deles opp i, hva som skal oppnås i de enkelte faser/trinn eller løp (av leveranser og modeller), hvem som skal delta, og hvordan arbeidet skal ledes, o s v. Anbefalte/mulige teknikker kan være listet, men blir sjelden gjennomgått i detalj. Eksempler på utviklingsmodeller er evolusjonær prototyping, eksperimentell prototyping, fossefall og spiralmodellen.

- *Teknikker:* Omfatter konseptene og notasjon (modelleringsspråk) man bruker i systemutviklingen. Teknikker er en innarbeidet arbeidsform eller en beskrevet fremgangsmåte som kan benyttes for å utføre et avgrenset stykke arbeid, eventuelt ved hjelp av et bestemt hjelpemiddel (verktøy).

- *Metoder:* Bruk av begrepene metode, utviklingsmodell og teknikker går ofte over i hverandre. Vanligvis blir metode oppfattet som en kombinasjon av en utviklingsmodell og teknikk. D v s både *hva* som skal oppnås, *prinsipper* for organisering – og utførelse (teknikker). Der utviklingsmodellen ofte er lite detaljert på beskrivelsen av teknikker er metoder ofte mye mer detaljert. I praksis viser det seg at metode ofte blir brukt som synonym for teknikk.

- *Teknologier og standarder*: Utviklingen av et system må forholde seg til de teknologiene (f eks programmeringsspråk, operativsystemer, nettverkssystemer, utviklingsomgivelser) som er tilgjengelige, og førende standarder (f eks RM–ODP, militære standarder, o s v) i den tiden system skal utvikles.

- *Verktøy:* Verktøyene en bruker i systemutviklingen er meget sentrale. Verktøyene brukes for å analyse, spesifisere, designe og implementere systemet. Det er derfor viktig at verktøyene understøtter prosessen, metodene og/eller teknikkene, og teknologiene som er valgt.

En metodikk er prosessorientert og skal sørge for å koordinere samspillet mellom disse forskjellige elementene. Valg av f eks teknologi, teknikker og utviklingsmodell bør derfor ikke betraktes som separate valg. Bruker en f eks objekt– og komponentteknologi og sikter på å lage et distribuert system bør en velge teknikker og utviklingsmodeller som er rettet inn mot denne type utvikling. Eksempelvis vil man måtte bruke forskjellige teknikker og utviklingsmodeller ved bygging av et agentsystem til forskjell for å bygge et klient– tjenersystem. En metodikk bør også være så fleksibel som mulig slik at den kan endres underveis og tilpasses bestemte utviklingsformål. Enkelte av elementene i en metodikk vil være faste, mens andre bør velges ut ifra det utviklingsmålet en har. Innenfor komponentorientert systemutvikling blir dette svært sentralt.

Et hovedpoeng er at man ikke bare kan "kjøpe og ta i bruk" en metodikk som vil føre en frem til målet. En metodikk må *designes* for å tilpasse:

- *det* man skal lage (produkt)
- *hvem* som skal lage (systemutviklingsprosjekt)
- *hvem* man lager *for* (kunde/organisasjon).

Ulike prosjekter trenger ulike metodikker. En metodikk er derfor ofte såpass generell slik at den kan tilpasses forskjellige typer systemer, organisasjoner og prosjektstørrelser. En slik metodikk blir ofte omtalt som et metodikkrammeverk (som f eks Rational Unified Process (RUP)[8]). Et metodikkrammeverk brukes da ofte som en referanse for å spesialdesigne og tilpasse mer spesifikke metodikker.

9.4.1 Inkrementer og iterasjoner

Inkrementer og iterasjoner er to viktige *utviklingsstrategier* som brukes i systemutvikling. I tradisjonell systemutvikling (typisk fossefall) konsentrerte man seg for mye om detaljer på et tidlig tidspunkt (analyse og design fasen). Problemet med denne fremgangsmåten var ofte at man la inn alt for mye arbeid veldig tidlig i prosessen for deretter, og ofte for sent, å finne ut at man i starten gjorde noen uheldige valg. Ved en metode som er iterativ og inkrementell, konsentrerer man seg om systemets hovedlinjer først, og deretter settes detaljene på plass.

Iterativ og inkrementell utviklingsstrategi fokuserer på hvordan et system skal bygges:

- Inkrementell utvikling er en *trinnvis strategi* hvor deler av systemet blir utviklet til forskjellig tid og hastighet, og integrert når de er ferdige. Mellom de forskjellige inkrementene har en rom for å reparere og forbedre utviklingsprosessen.
- Iterativ utvikling er en *bearbeidingsstrategi* hvor tid blir satt til side for å revidere og forbedre deler av systemet. Iterasjoner lar deg reparere og forbedre kvaliteten til systemet. Man itererer over utviklingsfaser og modeller.

9.4.2 Arkitekturfokus i komponentbasert systemutvikling

Ettersom distribuerte systemer får større omfang, blir krav til kommunikasjon og samarbeid mellom komponenter viktigere. Komponentbaserte systemer forutsetter arkitekturer slik at nye komponenter kan settes inn i en sammenheng og følger bestemte retningslinjer. En arkitektur for et informasjonssystem kan litt forenklet betraktes som et høynivå design som beskriver hvordan de ulike komponentene skal designes og utvikles over tid. Et utvidet arkitekturbegrep definerer de store retningslinjene for hvordan systemet skal bygges og passes inn i en bestemt situasjon og organisasjon. Et informasjonssystem vil ha mange forskjellige funksjoner hvorav de fleste designes som egne komponenter og har sin egen interne arkitektur. Hver av de ulike komponentene må passe inn i hovedarkitekturen, d v s at deres eksterne kommunikasjon må være forenlig med andre komponenter, men deres interne arkitektur kan designes uavhengig.

Det finnes i dag ingen generelle arkitekturbaserte metodikker som har et slikt utvidet arkitekturbegrep for utvikling av større systemer. Enkelte systemutviklingsmetodikker har arkitekturbegrepet innbakt, men da i en snevrere betydning. Systemutviklingsmetodikker fokuserer på analyse, design, implementasjon og testing av et system, hvor da arkitekturbegrepet er knyttet til disse fasene.

8. RUP blir nærmere beskrevet i kapittel 9.5.

I forhold til prosjektets arkitekturfokus vil de ulike komponentene (eller system/ delsystemene) i en arkitektur fritt kunne utvikles i forhold til en eller flere passende valgte systemutviklingsmetodikker. En overordnet arkitektur vil *legge føringer* på systemutviklingsprosessen. Det er derfor viktig at dette støttes i de systemutviklings– metodikkenene og verktøyene man velger å bruke.

9.5 "Rational Unified Process"

"Rational Unified Process" (RUP) er et metodikkrammeverk som spesielt har fokus på metode og prosess (27). Imidlertid påpekes også viktigheten av sammenhengen mellom mennesker, organisasjon og verktøy. RUP brukes av utviklere for å utføre både prosjektledelsesoppgaver og utviklingsoppgaver. RUP beskriver seg selv som et generisk metodikkrammeverk som kan tilpasses forskjellige typer systemer, organisasjoner og prosjektstørrelser.

RUP er modelldrevet og benytter seg av UML. UML er et modelleringsspråk som brukes til å spesifisere, visualisere, konstruere og dokumentere programvaresystemer (28). Både prosessen (RUP) og modelleringsspråket (UML) er utviklet av de samme personene og har derfor følgelig vektlagt en tett integrasjon mellom disse. I november 1997 ble UML adoptert av OMG som et standard objekt modelleringsspråk og har etterhvert fått en stor utbredelse. UML brukes i dag i mange forskjellige sammenhenger.

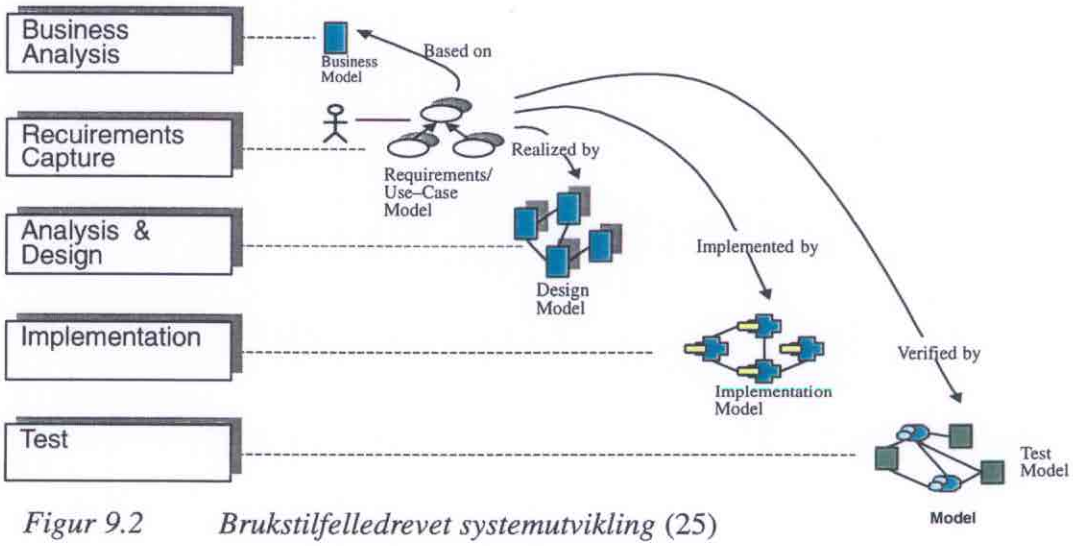
RUP kan beskrives gjennom tre viktige nøkkelbegreper:

1. *"Use case" basert tilnærming:* En "use case" beskriver et *bruksmønster* eller et *brukstilfelle*. Et brukstilfelle er en måte en brukertype bruker systemet på. Hver bruk danner ett brukstilfelle. En *brukstilfelledrevet* ("use case driven") systemutvikling betyr at alt som skal gjøres i utviklingsprosessen gjøres for å kunne realisere ett eller flere brukstilfeller. En brukstilfellebeskrivelse beskiver en funksjonalitet i systemet som gir en brukertype av systemet et resultat eller en verdi. Et brukstilfelle er atomært, og en brukertype kan være en person eller en applikasjon. Alle brukstilfellebeskrivelsene vil tilsammen utgjør beskrivelsen av de funksjonelle kravene til systemet og samles i en "use–case" modell (aggregering). Basert på denne modellen blir hvert av brukstilfellene detajert ned i konkrete analyse–, design–, implementasjon– og testmodeller. Figur 9.2 viser prinsippene for brukstilfelledrevet systemutvikling.

2. *Arkitektursentrert prosess:* Arkitekturfokus i RUP må ikke misforstås med arkitekturbegrepet slik det er definert i forrige kapittel. I RUP siktes det til et høynivå design, hvor de viktigste *brukstilfellene* utdypes og de mindre viktige skjules. Arkitektursentert prosess betyr at arkitekturen utvikles og brukes aktivt under hele utviklingsforløpet og at brukstilfellene driver utviklingen av arkitekturen. Arkitekturen omfatter det som brukstilfellene ikke omfatter. Det gjelder f eks organisering av systemet, struktur, grensesnitt, oppførsel og samarbeid mellom elementer i systemet. I RUP utvikles det en arkitektur for hver modell som lages. Det utvikles en arkitektur for "use–case"–modellen, en arkitektur for analysemodellen, en arkitektur for designmodellen, o s v. Disse arki-

tekturene vil etterhvert utgjøre de forskjellige arkitekturperspektivene ("views") i den endelige arkitekturbeskrivelsen av systemet som helhet. I RUP betyr bruk av arkitekturer at fokus for utviklingen først legges på systemets grunnstrukturer, d v s de store trekkene. Deretter ser man på detaljene. Ved å fokusere på de viktigste og mest sentrale aspektene i designet vil man rette søkelyset på de viktigste målene som forståelse, modifiserbarhet og gjenbruk ved et design. Hvert av de sentrale brukstilfellene spesifiseres i detalj ved hjelp av subsystemer, klasser og komponenter.

3. *Iterativ og inkrementell utvikling:* I RUP utvikles systemet i en serie av iterasjoner hvor systemets funksjonalitet utvides i hver iterasjon. Målet med hver iterasjon er en kjørbar versjon av systemet, men hvor all funksjonalitet ennå ikke er på plass. Hver iterasjon kjøres som et lite "miniprosjekt". Hvert miniprosjekt resulterer i et inkrement. Hver iterasjon (eller miniprosjekt) inneholder varianter av trinnene *planlegging–analyse–design–koding–test–evaluering*. Hele utviklingsprosessen består da av en serie med iterasjoner med disse trinnene, hvor hver iterasjon leverer ett inkrement som resultat. I RUP refererer iterasjoner seg til steg i arbeidsprosessen og inkrementer til påbygning av systemet. Dette er litt annerledes enn vår definisjon (kapittel 9.4.1). I forhold til vår definisjon kan flere iterasjoner skje innenfor ett inkrement for å forbedre kvaliteten til systemet vi bygger.



Figur 9.2 *Brukstilfelledrevet systemutvikling* (25)

9.6 "Catalysis"

Catalysis er en utviklingsmetodikk som har spesialisert seg på komponentbaserte systemer, og omfatter metode og prosess (34). Den bruker objektorienterte og komponentbaserte metoder og en variant av UML som modelleringsspråk.

Catalysis beskriver forskjellige prosessmønstre som kan brukes (og gjenbrukes) i forskjellige typer prosjekter. Kort fortalt har utviklingsprosessen i Catalysis følger fire hovedfaser. Ikke alle faser er obligatoriske:

1. *Virksomhetmodellering:* Etablere forståelse for problemområdet (prosess, roller og samarbeid) og terminologi. Denne fasen er ikke obligatorisk. Det er fordi man f eks noen ganger "bare" skal utvikle en komponent. I slike tilfeller kan neste fase (2) være et bedre sted å starte.

2. *Systemkontekst og kravspesifikasjon:* Etablere forståelse for systemets rolle i virksomheten og spesifisere krav til systemet som skal bygges. Denne fasen er obligatorisk enten man skal designe en komponent eller et helt system. Beskrivelsen skal være helt uten referanse til hvordan systemet eller komponenten er tenkt implementert. Spesifikasjonene blir presist definert ved hjelp av aksjoner og pre– og postbetingelser i UMLs "Object Constraint Language" (OCL).

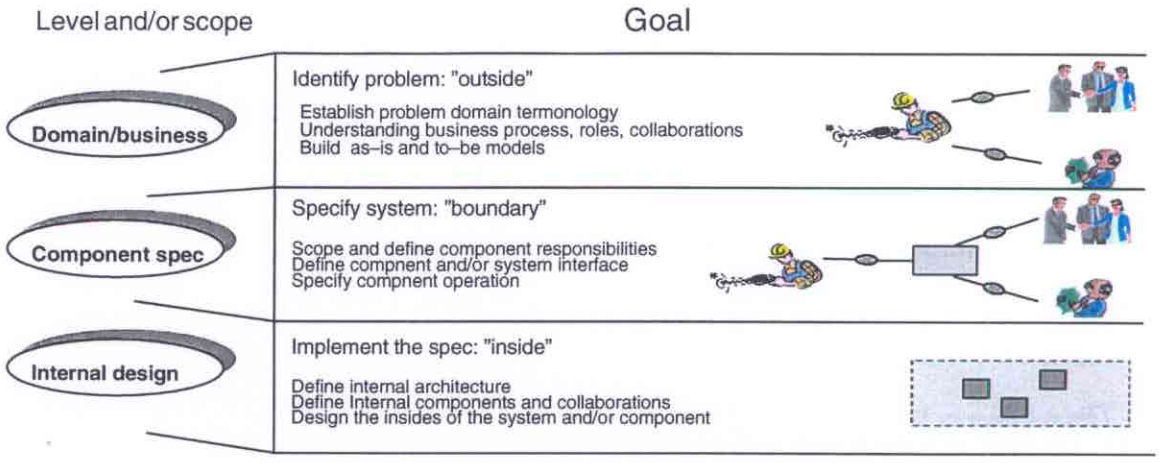
3. *Komponentdesign og komponentspesifikasjon:* Komponentdesignet beskriver hvordan hovedkomponentene i systemet samarbeider. Designet har også med referanse til kravspesifikasjonene for hver enkelt komponent. Komponentmodeller brukes for å beskrive ekstern oppførsel og grensesnitt til komponentene. Komponentarkitekturen i Catalysis definerer en såkalt "kit architecture" som beskriver felles elementer for hvordan en samling komponenter samarbeider, f eks standard interaksjonsprotokoller. Arkitekturen definerer her såkalte *porter* ("ports") og ulike typer *koblingsforbindelser* ("connectors") for å beskrive pluggbare komponenter. Portene er grensesnittene til komponentene som definerer tilknytningspunkter mot andre komponenter. Koblingsforbindelsene er kommunikasjonen mellom portene og brukes for å binde komponenter sammen. En koblingsforbindelse kan være alt fra enkle funksjonskall til en mer kompleks dialog over en API. Kit'et brukes av utviklere for å bygge interoperable komponenter som kan settes sammen med liknende produkter.

4. *Objektorientert design og implementasjon av komponenter:* Denne fasen beskriver hvordan et system eller en komponent virker, helt ned til programmerinsspråk, klasser eller et annet nivå som kan bli direkte kodet (eller omgjort til kode via en kodegenerator). Denne fasen er obligatorisk.

Det spesielle ved Catalysis er at den beskriver komponenter rekursivt (en komponent består av komponenter, o s v). F eks så betraktes et programvaresystem som et "objekt" i en forretningsomgivelse på samme måte som en forretningsomgivelse kan betraktes som et "objekt" i en større omgivelse igjen. Betraktet fra "utsiden" samarbeider et programvaresystem med andre programvaresystem eller objekter. Betraktet fra "innsiden" består et programvaresystem av en samling samarbeidende objekter. Figur 9.3 viser tre rekursive nivåer av modeller i Catalysis. På denne måten dekomponeres komponenter helt ned til implementasjon.

Catalysis har eksistert siden 1992 og har influert på UML–standarden og Microsoft sin komponentmodell som er implementert i Microsoft Repository.

Catalysis støtter komponentteknologier og standarder basert på Java, COM+, CORBA og RM–ODP. Catalysis støtter også brukstilfelledrevet systemutvikling.



Figur 9.3 Catalysis' tre rekursive nivåer av modeller (34)

10 OBJEKT- OG KOMPONENTTEKNOLOGI

Det er nye momenter innenfor objekt- og komponentteknologi som nå introduseres i markedet. Gitt at de nye teknologiene slår igjennom, kan man vente seg et stort marked med tilgjengelige og konkurrende produkter hvor bruk av denne type teknologi vil være vanlig i systemutviklingen om få år. En bør kjenne til utviklingen av disse teknologiene og se hvilke muligheter de kan gi med tanke på framtidige militære operative informasjonssystemer.

10.1 Objektorienterte databaser

Objektorienterte databaser ble allerede omtalt på slutten av 80–tallet og er følgelig ingen ny teknologi. Tidligere har man stort sett klart seg med tradisjonelle relasjonsdatabaser og det er først nå at objektorienterte databaser har fått nytt fokus med tanke på nye anvendelser og krav f eks innen multimedia.

To typer objektorienterte databaser finnes på markedet i dag:

1. *Persistent lagring* for objektorienterte programmeringsspråk, f eks C++ og Java, som "mapper" objekter til persistente datastrukturer.
2. *Objektorienterte databasesystemer* som minimum består av et databasesystem (inkluderer forespørsler, transaksjoner, o s v) og en objektorientert datamodell.

10.1.1 ODMG 3.0

Gruppen Object Database Management Group (ODMG) ble etablert i 1991 for å fremme standarder for objektlagring. Deres første objektdatabasestandard ble publisert i 1993. Den nåværende standarden er ODMG 3.0 (22).

ODMG 3.0–standarden spesifiserer en objektmodell, et objektdefinisjonsspråk (ODL), et objektspørrespråk (OQL) og språkbindinger for programmeringsspråkene C++, Smalltak og Java:

- *Object Model:* Objektmodellen sørger for samsvar mellom representasjonen av data i objektdatabasen og representasjonen av data i det objektorienterte programmeringsspråket som støttes av ODMG.
- *Object Definition Language (ODL):* Objektdefinisjonspråket beskriver databaseskjemaet, d v s hierarkiet av objekttyper, deres egenskaper og metoder.
- *Object Query Language (OQL):* Objektspørrespråket er basert på SQL 92. OQL kan brukes for å spørre om komplekse datatyper.
- *Language interface bindings:* Språkbindingene sørger for at systemer/applikasjoner implementert i et ODMG–støttet programmeringsspråk kan kommuniserere med databasen.

For at en database kan sies å være *ODMG–konform* holder det å overholde kun ett av følgende fire punkter: a) ODL, b) OQL, c) C++–språkbinding, eller d) Smalltalk–språkbinding. En database er *ODMG–sertifisert* hvis den overholder alle de fire kriteriene. Per dags dato finnes ingen ODMG–sertifiserte databaser og dette betyr at ODMG–standarden egentlig bare er en retningslinje for hvordan objektorienterte databaser bør bygges.

10.1.2 Fordeler ved objektorienterte databaser

Hovedargumentet som taler for bruk av objektorienterte databaser i stedet for tradisjonelle relasjonsdatabaser dreier seg om datakompleksitet. Behovet for objektorienterte databaser fremtvinges ved økende datakompleksitet som:

- eksistens av mange–til–mange relasjoner,
- eksistens av flerfoldige ulike datatyper,
- komplekse relasjoner mellom datatyper og
- evolusjonære datastrukturer.

Tradisjonelle relasjonsdatabaser ble designet for å representere "enkle" datatyper som f eks navn (strenger) og beløp (tall) og enkle relasjoner som kan representeres i tabeller (matriser). Objektorienterte databaser er designet for å representere komplekse og "naturlige" datastrukturer som bedre samsvarer med virkelige entiteter i verden. Objektorienterte databaser klarer bedre å skalere lineært med økende kompleksitet i objektstruktuer mens relasjonsdatabaser ofte krever eksponensiell økning i antall "join"–operasjoner av tabeller. Relasjonsdatabaser er lite egnet for å håndtere komplekse datatyper. Dette kommer til uttrykk i form av

- urimelig stort minneforbruk for å representere komplekse datarelasjoner i "join"–tabeller,

- betydelige ytelsesproblemer ved databasespørringer og
- eksponensiell økende vanskelighetsgrad med å håndtere og utvide relasjonstabeller etterhvert som datakompleksiteten øker.

Eksempler på objektdatabaser på markedet i dag inkluderer ObjectStore, Versant og Objectivity.

10.2 Applikasjonstjenere og komponenter

Gjennom 90–tallet har utviklingen av tradisjonelle informasjonssystemer gått fra en 2–lags klient–tjener arkitektur til å følge en mer fleksibel 3–lags eller n–lags arkitektur. De nye logiske flerlagsarkitekturene separerer domene–/virksomshetslogikk fra systemtjenester og brukergrensesnitt. Utviklingen av mellomvareteknologi som tilbyr generiske systemtjenester, transaksjonsmonitorer, meldingsorientert mellomvare, ORBer (objektorientert mellomvare) o s v har sørget for at n–lags arkitekturen har slått igjennom.

N–lags arkitekturen forenkler utvikling og deployering, samt håndtering av virksomshetsapplikasjoner. En n–lags arkitektur hjelper utviklere å fokusere på domenespesifikke programmeringsaspekter ved å bruke eksisterende mellomvareteknologi for å fremskaffe en infrastruktur, samt klientapplikasjoner for å tilby brukerinteraksjon. Virksomhetslogikk implementeres i tjenestekomponenter som kan deployeres og distribueres på tjenermaskiner i forhold til organisasjonens krav.

Inntil nylig har utviklingen av n–lags arkitekturer blitt implementert med en rekke forskjellige standarder, og har begrenset mulighetene for utviklerene å bygge applikasjoner ved hjelp av standardiserte og kommersielt tilgjengelige programvarekomponenter. Både Javasoft, Microsoft og OMG har jobbet med å komme fram til en skalerbar standard komponentmodell. Javasoft har spesifisert en Enterprise JavaBeans komponentteknologi som inngår i deres Java 2 Enterprise Edition (J2EE) omgivelse. Microsoft har utviklet sin egen Windows Distributed interNet Application (DNA) omgivelse over Windows 2000–plattformen hvor COM+/DCOM komponentteknologien står sentralt. OMG har lansert en språkuavhengig CORBA Component Model.

10.2.1 J2EE

J2EE ble utviklet for å løse problemene rundt utvikling, deployering og håndtering av skalerbare flerlagssystemløsninger. J2EE–spesifikasjonen er et resultat av et industrisamarbeide ledet av Sun Microsystems. Blant de viktigste industripartnere involvert i spesifikasjonsarbeidet, finner vi IBM og Oracle.

J2EE er en evolusjon av flere Java–teknologier:

- *Enterprise JavaBeans (EJB):* Arkitektur for å bygge gjenbrukbare tjenerkomponenter.

- *Java Database Connectivity (JDBC):* Java–grensesnitt til relasjonsdatabaser.
- *Java Naming and Directory Interface (JNDI):* Lokaliserer ressurser over et nett–verk.
- *Java Remote Method Invocation over the Internet Inter ORB–Protocol (RMI–IIOP):* Fjernmetodekall mellom virtuelle Java–maskiner. Den OMG–definerte IIOP protokollen tillater integrasjon mot andre språk.
- *Java Message Service (JMS):* Asynkron kommunikasjon ved hjelp av meldinger.
- *Java Interface Definition Language (Java IDL):* En Java CORBA ORB som implementerer et subsett av CORBA–spesifikasjonen.
- *Connectors (framtidig utvidelse):* Aksessintegrasjon mot virksomshetsinformasjonssystemer som CICS, Tuxedo, SAP R/3 og PeopleSoft.
- *JavaServer Pages (JSP):* Dynamisk generering av websider.
- *Java Servlets:* Servlets er komponenter som deployeres på en webtjener.
- *Java Transaction API (JTA):* Transaksjonstjeneste.
- *eXtensible Markup Language (XML):* En Java XML API som gir et grensesnitt mot en XML "parser" (tolker).
- *JavaMail:* Elektronisk post.
- *JavaBeans Activation Fremework (JAF):* Automatisk aktivering av Java–komponenter for å håndtere ulike objekter.
- *Java 2 Platform Standard Edition (J2SE):* Java 2 plattformen som inkluderer kjernespråktjenestene.
- Andre virksomhetstjenester (f eks Load–balancing, data caching, transparent failover): J2EE–spesifikasjonen skiller ut disse transparente tjenestene slik at mellomvareleverandører kan tilby disse.

EJB definerer en arkitektur for utvikling og deployering av gjenbrukbare *Java tjenerkomponenter.* Tjenerkomponenter er komponenter som kjører på en applikasjonstjener. En applikasjonstjener tilbyr en optimalisert eksekveringsomgivelse for applikasjonskomponenter. En applikasjonstjener automatiserer håndtering av mer komplekse egenskaper knyttet til distribuerte systemer som f eks "load balancing" og distribusjon på flere maskiner. En applikasjonstjener tilbyr også infrastrukturtjenester som navne– og katalogtjeneste, transaksjoner, persistens og sikkerhet.

10.2.1.1 Teknologi– og markedsvurdering

Fordelene ved J2EE er at den er en åpen spesifikasjon ("åpen spesifikasjon" i betydningen av "de–facto standard") som er maskin– og plattformuavhengig, og som fremmer programvareleverandøruavhengighet.

Ulempen er at komponentmodellen er bundet til et bestemt språk, nemlig Java. RMI–IIOP tillater imidlertid tjenesteinteroperabilitet mot biblioteker/komponenter implementert i andre språk.

10.2.2 Windows Distributed interNet Application

Windows DNA er som J2EE også utviklet for å løse problemer tilknyttet utvikling, deployering og håndtering av flerlagssystemløsninger.

Windows DNA har utviklet seg fra de ulike mellomvaretjenestene tilbudt i Windows NT. Windows DNA omfatter følgende teknologier:

- *Windows 2000:* Underliggende operativssystemet som Windows DNA applikasjoner må kjøres på.
- *COM+:* Arkitektur for å utvikle gjenbrukbare tjenerkomponenter.
- *Distributed COM (DCOM):* Teknologi for grensesnitt/implementasjonsseparering og språkuavhengighet som tillater distribuerte komponenter.
- *Microsoft Transaction Server (MTS):* Transaksjonsmonitor og en Object Request Broker (ORB) som håndterer tjenerkomponenter.
- *Microsoft Message Queue (MSMQ):* Asynkron kommunikasjon mellom komponenter.
- *Microsoft Cluster Server ("Wolfpack"):* Redundans og "backup"–mekanismer
- *Network Load–Balancing:* "Load–balancing" av IP–trafikk mellom webtjenere.
- *Component Load–Balancing (future):* "Load–balancing" av tjenere som kjører COM+ –kompontenter.
- *Microsoft Active Data Objects (ADO) and OLE DB:* API for aksess til relasjonsdatabaser.
- *Microsoft "Babylon" Integration Server:* Integrasjon mot eksisterende virksomhetsinformasjonssystemer.
- *Microsoft Internet Information Server:* Webtjener som tilbyr en Internet Server API (ISAPI) for webapplikasjoner.
- *Active Server Pages (ASP):* Dynamisk generering av websider.
- *Microsoft Active Directory:* Navne– og katalogtjeneste som brukes for å lagre informasjon om brukere, datamaskiner og andre ressurser (komponenter) i et nettverk.

10.2.2.1 Teknologi– og markedsvurdering

Microsoft er en dominerende part i IT–industrien hvis teknologi man er nødt til å forholde seg til. Den meste åpenbare fordelen ved satsing på Windows DNA vil være en tett integrasjon mot operativssystemet. DCOM/COM komponentteknologien er også språknøytral.

Imidlertid er denne fordelen også den største ulempen. DNA er en proprietær spesifikasjon utviklet av Microsoft som er bundet til et spesifikt operativssystem (Windows 2000). Man blir altså bundet til en bestemt arkitektur, en bestemt leverandør og et bestemt operativsystem. Siden Windows operativssystemet finnes kun for Intel maskinvareplattformen vil man også binde seg til en bestemt plattform. En satsing på denne proprietære arkitekturen begrenser framtidige teknologimuligheter. Slik markedet er i dag, er Microsoft så dominerende at de er i stand til å beholde sine propritære standarder og fortsatt være teknologiledende. I motsetning til åpne standarder som ofte må gjennom en lengre "byråkratisk" spesifikasjonsprosess, har Microsoft muligheten til å være først på markedet.

Ved hjelp av broteknologi kan man imidlertid oppnå interoperabilitet mellom forskjellige arkitekturer og plattformer. Et K2IS vil sannsynligvis operere i et heterogent miljø og det kan være hensiktsmessig at enkelte områder kan utvikles og kjøres vha Windows–teknologi og bruke broteknologi for å sikre interoperabilitet.

10.2.3 CORBA Component Model

CORBA Component Model (CCM) er en språk– og maskinuavhengig komponentmodell som bygger på etablerte standarder og konsepter fra EJB–standarden. CCM har utvidet EJB–standarden for Java med støtte for andre språk. CCM er dermed et superset av EJB.

10.2.3.1 Teknologi– og markedsvurdering

Fordelen ved CCM er at det er en åpen spesifikasjon som er språk–, maskin– og plattformuavhengig og som fremmer leverandøruavhengighet. Ulempen er at den er unødvendig kompleks hvis man begrenser seg i forhold til f eks språkvalg.

Det er vanskelig å spå hvordan markedet vil utvikle seg. CORBA er en byråkratisk standard hvor implementasjoner lar vente på seg. Imidlertid har CORBA et stort marked når det gjelder "legacy wrapping". Dette vil også med stor sannsynlighet være situasjonen i årene som kommer.

11 MELLOMVARETEKNOLOGI

Mellomvareteknologi brukes til å integrere programvareapplikasjoner i en distribuert omgivelse. Den lar programvareutviklere konsentrere seg om organisasjons– og domenerelevante aspekter under utvikling av applikasjoner, istedenfor å bruke mye tid på å bygge infrastruktur.

Bruk av mellomvareteknologi vil være sentralt i utviklingen av et maritimt K2IS fordi mellomvare tilbyr tjenester som moderne applikasjoner vil trenge. En evaluering av forskjellige mellomvareteknologier vil derfor være hensiktsmessig for valg av løsninger som best tilfredsstiller militære krav og ønsker:

- Et militært operativt informasjonssystem antas å bestå av mange forskjellige delsystemer og komponenter, som vil kunne utvikles i forskjellige programmeringsspråk av-

hengig av behov. Framtidige informasjonssystemer antas å være distribuerte, som betyr at komponentene i systemet distribueres på flere maskiner i et heterogent miljø. D v s komponentene deployeres på forskjellige maskinplattformer (maskinvarearkitekturer/ operativssystemer, o s v). Av denne grunn er det viktig at den valgte mellomvareteknologien gir maskin–/plattform–/ programmeringsspråk– uavhengighet og interoperabilitet.

- Tilgjengelighet av standardiserte og innkapslede tjenester som f eks sikkerhet, samt domenespesifikke kommando– og kontrollkomponenter vil også være viktig for valg av mellomvareløsning som et K2IS skal bygges over.

Mellomvareteknologien tilbyr ulike fundament som basis for å bygge distribuerte applikasjoner og tjenester. Dette inkluderer dokumentorienterte (f eks Web) og objekt/komponentorienterte systemer. Mellomvare har også en rolle i informasjonsbehandling gjennom generiske tjenester som transaksjons– og databasetjenester, spørring– og relasjonstjenester, o s v.

Nedenfor gis en kort oversikt over forskjellige mellomvareteknologier som en bør kjenne til i lys av et framtidig K2IS.

11.1 Oversikt over grunnleggende mellomvareteknologier

De grunnleggende mellomvaretjenestene inkluderer:

- Fjernkommunikasjonstjenester som fjernprosedyrekall (Remote Procedure Call (RPC)), fjerndataaksess (Remote Data Access (RDA)) og meldingsorientert mellomvare (Message–Oriented Middelware (MOM)).
- Administrasjons– og støttetjenester som sikkerhet, katalogisering, tid og feilhåndtering.

Fjernkommunikasjonstjenester tilbyr grensesnitt for utveksling av informasjon for klient–tjener programmer i et nettverk. Administrasjons– og støttetjenester er viktige for å kunne håndtere kompleksiteten i et distribuert system.

En forståelse av de grunnleggende klient–tjener–mellomvaretjenester er viktige i utvikling av distribuerte applikasjoner siden mange andre høyereliggende mellomvaretjenester (f eks objektorientert mellomvare) er bygget over disse. De tre fjernkommunikasjonstjenestene RPC, RDA og Queued Message Processing (QMP) beskrevet nedenfor har sine fordeler og ulemper. Ideelt bør derfor et høyereliggende mellomvareteknologi støtte alle tre modellene for å tilby utviklere av distribuerte systemer størst mulig fleksibilitet. Det gjør bl a CORBA.

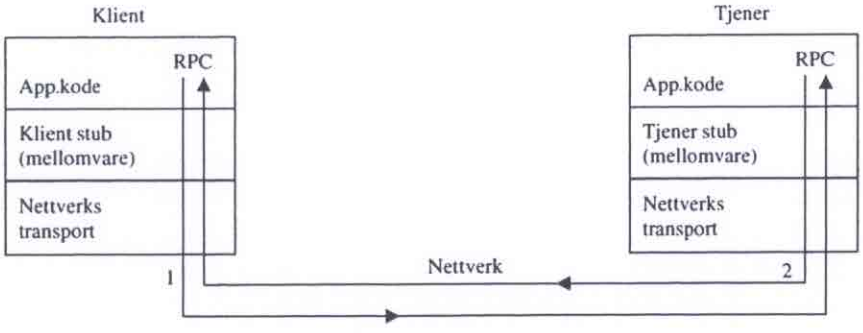
11.1.1 Fjernprosedyrekall

Fjernprosedyrekall (RPC) ble utviklet på tidlig 80–tallet av SUN Microsystem, som en del av deres Open Network Computing (ONC) plattform. RPC er operasjoner som kan anro-

pes på tvers av maskinvare– og operativsystemplattformer. SUN forela RPC som en standard til X/Open Consortium, og den ble adoptert som en del av Distributed Computing Environment (DCE). RPC er opprinnelsen til det vi i dag kjenner som objektorientert mellomvare, eller også kalt ORB mellomvare. Eksempler på disse er CORBA, DCOM og Remote Method Invocation (RMI)). Objektorientert mellomvare blir nærmere beskrevet i kapittel 11.3.

RPC fungerer ved at klienten kaller en fjernprosedyre lokalisert på tjeneren. Prosedyren eksekveres og tjeneren gir tilbake en respons til klienten. Klientkallet kalles en forespørsel og resultatet fra tjeneren kalles en respons. RPC gjemmer alle nettverksrelaterte detaljer og gir en transparent aksess over nettverket.

Figur 11.1 viser stegene i et fjernprosedyrekall. Klient– og tjenerrutinene er i to forskjellige prosesser. RPC–mellomvaren lager et lokalt "dummyobjekt", kalt en klient–stub. Klient–stub'en har samme navn som fjernprosedyren. Ved en forepørsel vil klientkoden anrope klient–stub'en. Klient–stub'en sørger for å pakke forespørselen i et format egnet for nettverksoverføring ("marshalling"). En standard transportprotokoll brukes for å sende den pakkede førespørselen over til tjeneren. En mottaker–stub (tjener–stub) opprettet på tjeneren, pakker ut forespørselen ("unmarshalling") og kaller den reelle fjernprosedyreimplementasjonen slik at den blir eksekvert på tjeneren. Deretter sendes en respons tilbake til klienten.



Figur 11.1 Fjernprosedyrekall fungerer ved at klienten kaller en fjernprosedyre i tjeneren som eksekveres og gir tilbake en respons til klienten

Grensesnittet for fjernprosedyren (som brukes av klienten) spesifiseres i et grensesnittdefinisjonsspråk (IDL). Gjennom dette grensesnittdefinisjonsspråket får man (ved IDL–kompilering) automatisk generert en klient–stub og tjener–stub som, litt enkelt sagt, sørger for å "gjemme" kompleksiteten i lav–nivå kommunikasjonen mellom klienten og tjeneren, d v s pakkingen og overføringen av data.

Fjernprosedyrekall er velkjent og mye brukt. Bruk av RPC kan gi en enkel overgang fra sentraliserte til distribuerte programmer. De fleste RPC–implementasjonene, deriblant SUN RPC og DCE RPC, er implementert over TCP/IP transportprotokollene. En ulempe med dagens RPC implementasjoner er at de kun støtter synkrone forespørsler. Ved en synkron forespørsel vil klienten blokkeres helt til den får en respons tilbake fra tjeneren. RPC er heller ikke spesielt egnet for overføring av store mengder data.

11.1.2 Fjerndataaksess

RDA, eller det som også blir omtalt som transaksjonsorientert mellomvare, brukes hovedsaklig i f m distribuerte databaseapplikasjoner. En transaksjonsorientert mellomvare støtter transaksjoner på tvers av distribuerte databasesystemer. Objektorientert mellomvare har tatt opp i seg slike transaksjonsmuligheter.

RDA tillater klientapplikasjoner å sende forespørsler, f eks i form av Structured Query Language (SQL), til databasetjenere (relasjonsdatabaser). Hovedforskjellen mellom RPC og RDA er at i RDA er ikke omfanget på resultatet av spørringen kjent på forhånd. Et resultat av en databaseforespørsel kan resultere i en tabell med et ukjent antall rader og et ukjent antall kolonner, avhengig av dataene i databasen. Hver parameter i RPC må være forhåndsdefinert og RPC fungerer derfor dårlig med tanke på dataoverføring.

Klientgrensesnittet i RDA er radbasert (rader i tabellen), men typisk tillater mellomvareløsningen å transportere blokker av rader for å øke ytelsen.

RDA har bred støtte blant databaseleverandører. Standardene for fjerndataaksess faller i to kategorier:

- Standarder for APIer som lar klientapplikasjoner være portable mellom databaser så lenge samme API brukes. Eksempler på standard fjerndatabasegrensesnitt er Microsoft ODBC, Sun Java Database Connectivity (JDBC) og SQL Access Group Command Level Interface (SAG CLI).
- Standarder for utvekslingsprotokoller som lar databasetjenere fra en leverandør interoperere med en klient fra en annen leverandør. Eksempler på slike utvekslingsprotokoller er ISO Remote Database Access (ISO RDA).

Den største fordelen ved RDA er at ingen tjenerprogramvare trenger å utvikles (databasetjeneren tar seg av kall av fjerndatabasespørringer i SQL). Imidlertid så bør bruken av RDA evalueres grundig. Det har vist seg at RDA i mange tilfeller kan generere unødvendig mye trafikk på nettverket p g a overføring av store tabeller.

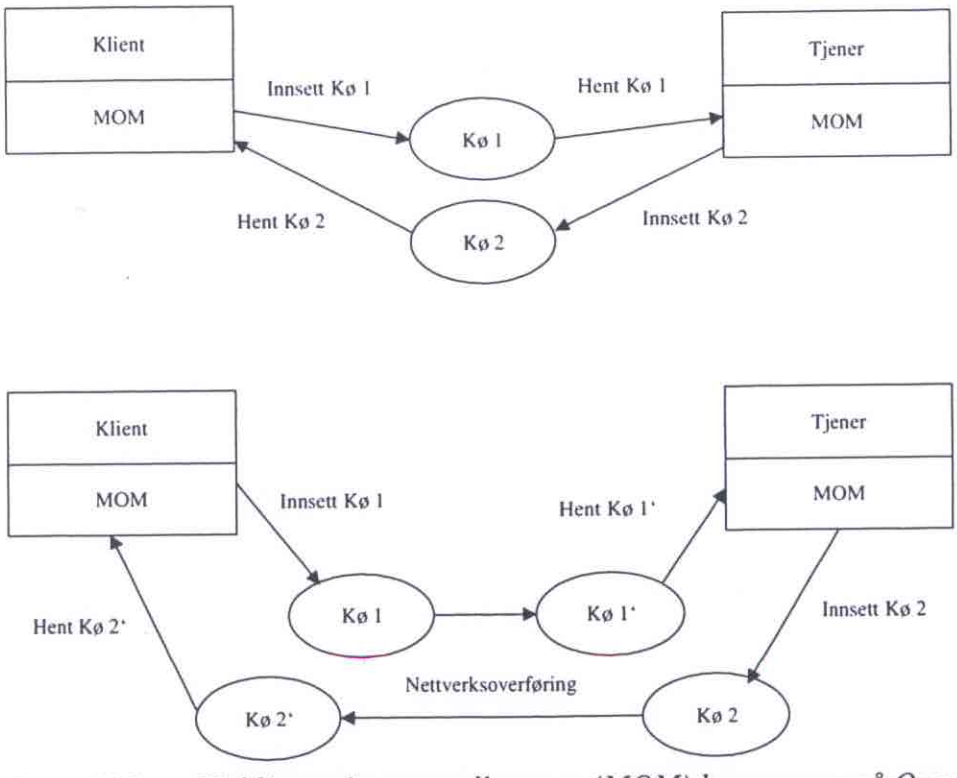
11.1.3 Meldingsorientert mellomvare

Meldingsorientert mellomvare (MOM) baserer seg på QMP hvor klient og tjener utveksler meldinger som blir lagret i køer. QMP tillater klienter å sende asynkrone forespørsler til tjeneren. Når en forespørsel er stilt i kø, blir forespørselen prosessert, selv om klienten skulle miste forbindelsen. Etter å ha satt meldingen i køen kan klienten fortsette med videre prosessering. MOM implementerer QMP og tilbyr programmeringsgrensesnitt og mekanismer for å sette inn og hente meldinger ut av køer.

Figur 11.2 viser hvordan MOM fungerer mellom en klient og en tjener. Man kan ha en kø for avsending og en kø for mottagelse av meldinger. Man kan også tenke seg at hver av disse køene blir implementert som to køer, hvorav den ene befinner seg på klientsiden og den andre befinner seg på tjenersiden.

- Applikasjoner sender datameldinger til mellomvarelaget. Disse meldingene kan være datasett eller forespørsler.
- Datameldingene blir plassert i eksekveringskøer av mellomvarelaget.
- Applikasjoner kan hente meldinger fra meldingskøene når de trenger de.

Hovedfordelen ved MOM er altså støtte for asynkron prosessering. I motsetning til RPC som krever direkte eller synkron kommunikasjon, vil MOM sørge for at klienter og tjenere ikke blokkeres når de venter på respons fra hverandre. Denne egenskapen tillater utvikling av samarbeidende distribuerte komponenter. En annen fordel ved MOM er gjenoppretting av feilsituasjoner, siden meldingskøene som lagres persistent kan brukes i gjenopprettingsarbeidet etter en feilsituasjon.



Figur 11.2 Meldingsorientert mellomvare (MOM) baserer seg på Queued–Message–Processing (QMP) hvor klient og tjener utveksler meldinger som blir lagret i køer

MOM kan også brukes til å integrere eksisterende applikasjoner enkelt uten modifisering av kode på begge sider ved at MOM brukes for å redirigere "input" og "output". MOM kan også være en bra løsning for distribuerte transaksjoner ved å ha transaksjonsbaserte køer. Hovedulempen ved MOM er overheadet med å skrive/lese fra persistente køer som kan degradere hastigheten til distribuerte systemer. Blant MOM produkter på markedet i dag finnes IBM MQ produktfamilie som inkluderer forskjellige tjenester og Digital Message Q. Objektorientert mellomvare har startet integrering mot meldingsorientert mellomvare.

11.1.4 Andre grunnleggende tjenester

I tillegg til de tre mekanismene beskrevet ovenfor som brukes primært for utveksling av data, finnes andre grunnleggende administrasjons– og støttetjenester som er viktige i en distribuert omgivelse. Blant disse tjenestene finner vi:

- *Sikkerhetstjenester:* Distribuerte systemer introduserer mange mulige sikkerhets–lekkasjer, f eks blir en tjener åpen for mange klienter og data overføres over et nettverk. Det er derfor viktig at det finnes sikkerhetstjenester, f eks autentisering av klienter og kryptering av data før overføring på nettverket
- *Feilhåndteringstjenester:* Feilhåndtering dreier seg om detektering, isolering og gjenoppretting av feil i distribuerte systemer
- *Navngiving og katalogtjenester:* Et distribuert system trenger å vite navnet og lokasjonen på objektene som håndteres.

11.2 Distribuerte transaksjonsmonitorer

En transaksjon er en sekvens av dataoperasjoner (lese, skrive og forandre) som transformerer en tilstand av systemet til en ny tilstand. En transaksjon har fire egenskaper, kalt ACID–egenskapene:

- *Atomær ("Atomicity"):* En transaksjon er en atomær operasjonsenhet. Alle aksjonene i en transaksjon fullføres eller ingen av dem. En transaksjon som fullføres sies å være "commited".
- *Konsistens ("Consistency"):* En transaksjon mapper en konsistent (korrekt) tilstand av databasen til en annen.
- *Isolasjon ("Isolation"):* En transaksjon kan ikke avsløre sine resultater til andre parallelle transaksjoner før sin fullførelse.
- *Bestandig ("Durability"):* Når en transaksjon er vellykket (commited), så er resultatene permanente og ikke forandret som en følge av systemfeil.

En transaksjonsmanager, også kalt transaksjonsmonitor, tillater flere maskiner å koordinere utførelsen av en enkel transaksjon. Konsekvensene av ACID–egenskapene overført til transaksjonsmanagerene for distribuerte transaksjoner, er som følger:

- *Serialiserbarhet (samtidighetskontroll):* Tillater transaksjoner å utføres samtidig og oppnå de samme logiske resultatene som om de hadde blitt utført i serie. Samtidighetskontroll tillater multiple transaksjoner å lese og oppdatere data samtidig, og inkluderer transaksjonsskedulering og håndtering av ressurser som trengs i løpet av en transaksjon
- *Commit prosessering:* Tillater "commitment" av transaksjonsforandringer. D v s transaksjonsforandringen blir permanent dersom transaksjonen som er fullført, er vellykket, eller fjernet dersom transaksjonen feiler.

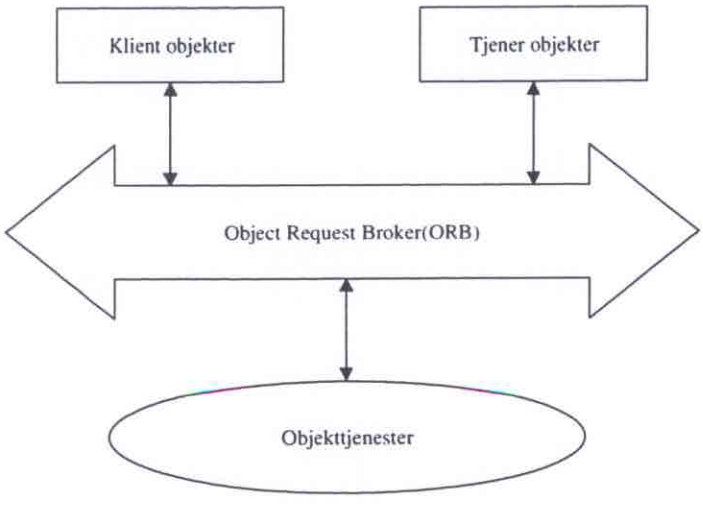
Tilgjengelige produkter på markedet i dag: BEA Tuxedo, Iona OrbixOTM og Microsoft Transaction Service.

11.3 ORB mellomvare

Årsaken til utviklingen av ORB–basert mellomvare (eller objektorientert mellomvare) er å gjøre objektorienterte prinsipper, som f eks identifisering av objekter gjennom referanser og arv, tilgjengelig for utviklingen av distribuerte systemer.

ORB–baserte mellomvareløsninger lar distribuerte objekter kommunisere med hverandre over en programvarebasert objektbuss. Det er denne bussen som kalles en ORB. Objektene snakker sammen over ORBen ved hjelp av en definert protokoll. Objektene knyttet til ORBen er distribuerte, og det er transparent for klienten hvorvidt objektet befinner seg lokalt eller fjernt.

En distribuert programvareapplikasjon kan sees som en samling av distribuerte objekter. Hvert distribuerte objekt har sine egne attributter og har metoder som definerer oppførselen til objektet. Interaksjoner mellom komponenter i systemet skjer gjennom beskjeder som anroper passende metoder ("requests"). I figur 11.3 vises en enkel distribuert objektmodell hvor et klientobjekt anroper et tjenerobjekt gjennom ORBen for å få utført en objekttjeneste.



Figur 11.3 En enkel distribuert objektmodell

- *Distribuerte objekter* er data omgitt med kode med egenskaper som arv, polymorfi og innkapsling. Distribuerte objekter kan være klienter, tjenere eller begge deler.
- *ORBer* tillater objekter å finne hverandre i et distribuert omgivelse og å ha interaksjon med hverandre over et nettverk. ORBer er grunnpilaren i distribuerte objektorienterte systemer.
- *Objekttjenester* tilbyr brukere å opprette, navngi, flytte, kopiere, lagre, slette, gjenopprette og håndtere distribuerte objekter.

All ORB–basert mellomvare har et IDL som støtter konsepter som: objekttyper som parametre, avansert feilhåndtering gjennom "exceptions", og arv.

ORB–orientert mellomvare er hovedsaklig basert på ideen fra RPC. Den første av denne type mellomvare var OMGs CORBA. Deretter tilføyde Microsoft distribusjonsmekanismer til sin Component Object Model (COM), som da ble til Distributed Component Object Model (DCOM), og SUN la til en mekanisme for Remote Method Invocation (RMI) i Java. Disse teknologiene beskrives mer utfyllende nedenfor.

11.3.1 CORBA

CORBA er en spesifikasjon foreslått av OMG og adresserer interoperabilitet mellom forskjellige maskinvareplatformer og programmeringsspråk i distribuerte objektsystemer. CORBA innehar også mekanismer for støtte til interoperabilitet mellom forskjellige mellomvareteknologier.

CORBA spesifiserer mekanismer som gjør at objekter kan sende anrop og motta svar på en transparent måte. Hovedkonseptene til CORBA omfatter:

- Spesifikasjon av mellomvaretjenester som brukes av applikasjoner.
- Enhver applikasjon kan være en klient, tjener eller begge deler.
- Enhver interaksjon mellom objekter skjer gjennom forespørsler.
- Støtter statisk og dynamisk binding mellom objekter.
- Et grensesnittdefinisjonsspråk. Et grensesnitt representerer kontrakten mellom klient– og tjenerapplikasjonen. Grensesnittdefinisjonen viser parametrene som overføres og en unik grensesnittidentifikator. Grensesnitt spesifiseres ved hjelp av en egen IDL, definert av OMG.
- En objektadapter (Portable Object Adapter (POA)) som styrer aktivering og deaktivering av objekter, og generering av objektreferanser.

CORBA har stor støtte i industrien og en rekke leverandører leverer CORBA–løsninger. Teknologien benyttes i større og større grad i f m oppgradering og utvikling av kampsystemer.

11.3.2 DCOM

DCOM er Microsofts ORB–variant for distribuerte objekter. DCOM vokste ut av OLE/COM–teknologien som var desktopløsninger og ble utvidet med distribusjonsmekanismer for å håndtere samspill mellom forskjellige maskiner. DCOM er ofte referert til som COM+.

Grensesnittdefinisjonsspråket i CORBA er vesentlig forskjellig fra grensesnittdefinisjonsspråket i DCOM. I motsetning til CORBA så støtter ikke DCOM konseptet om identifise-

ring av objekter gjennom objektreferanser. DCOM–objekter blir istedet identifisert gjennom pekere til referanser i minnet for å finne ut hvor et objekt er lokalisert. Disse pekerene blir kalt for grensesnittpekere. DCOM–objekter (også kjent som ActiveX–objekter) er komponenter som støtter ett eller flere grensesnitt.

11.3.3 Java RMI

Java Remote Method Invocation (RMI) tilbyr fjernmetodekall på distribuerte Java–objekter mellom forskjellige Java–virtuelle maskiner (JVM). Java RMI er ikke designet for å støtte fjernprosedyrekall mellom Java–objekter og objekter skrevet i et annet språk. Java RMI kan sees på som SUNs CORBA–løsning for Java–plattformen.

I J2EE introduserer Sun RMI i CORBAs kommunikasjonsprotokoll, RMI–IIOP. RMI–IIOP tillater Java RMI–programmer å kommunisere med CORBA–komponenter skrevet i andre språk.

11.4 "Message broker"

En annen type "broker" kalt "message broker" er et alternativt til ORB–arkitekturene beskrevet ovenfor. En "message broker" leverer *meldinger* mellom distribuerte applikasjoner. Ideen er å tilby mulighet for asynkron kommunikasjon mellom klient og tjener, som støtter en "publish/subscribe"–modell. Asynkrone forespørsler og svar mellom klient og tjener blir utvekslet gjennom en meldingskø, som baserer seg på "først–inn–først–ut" prinsippet. Programmer på forskjellige plattformer kan aksessere de samme meldingskøene.

Slike "message brokers" er lett tilgjengelig på markedet idag. Implementasjoner av meldingskøer blir generelt referert til som MOMs. Slike implementasjoner er ikke begrenset til kun å brukes innenfor distribuerte og objektorienterte systemer. Tilgjengelige produkter på markedet i dag er IBM MQSeries og Java Message Service (JMS). DCOM har et meldingskøsystem under spesifisering. CORBA har, i sin CORBA Services, inkludert en meldingstjeneste som støtter køing av meldinger.

11.5 "Legacy" tilgang/integrasjon mellomvare

Mellomvareteknologiene beskrevet ovenfor, brukes i utviklingen av nye og moderne applikasjonsystemer. De nyutviklede applikasjonene må ofte sameksistere og interopere med eldre eksisterende systemer. "Legacy" er et begrep som brukes om gamle programvaresystemer som fortsatt har nytteverdi. "Legacy"–integrasjon dreier seg om å gjøre eksisterende programvaresystemer tilgjengelige for nye systemer og applikasjoner. Man har forskjellige nivåer av integrasjon. Eksempler på ulike typer integrasjon er:

- Legge på nye grensesnitt (f eks Web) til eksisterende systemer
- Bruke de eksisterende systemer som tjener–delen til i et nytt klient–tjener system
- Linke eksisterende systemer til hverandre på programmeringsnivå.

Strategier for å håndtere "legacy"–systemer varierer fra total omskriving av systemet, til gradvis migrasjon og aksess/integrasjon. Aksess/integrasjon tillater "legacy"–applikasjoner å sameksistere med de nye applikasjonene. Det er behov for mellomvare for "legacy"–integrasjon som støtter forskjellige integrasjonsteknikker:

- "Screen scraping"
- "Database gateways"
- "Application gateways"
- "Object wrappers"
- "Integration gateways."

11.5.1 Skjermskraping

Skjermskraping ("screen scraping") tillater klientapplikasjoner å simulere terminal tastatur– og skjermegenskaper og således agere som programmerbare terminalemulatorer. Skjermskrapermellomvare tilbyr et API som kan brukes av utvikleren for å bygge skjermbilder, sende skjermbildet til vertsapplikasjonen, simulere tastetrykk, motta skjermbilder fra verten, og lese ut felter fra skjermbildet ved "skraping" av skjermbildene.

Skjermskrapere er ofte brukt til å aksessere monolittiske "legacy"–applikasjoner. Skjermskraping fungerer stort sett alltid og for en del eldre systemer er terminalemulering og skjermskraping eneste måte å aksessere "legacy"–data på. Skjermskrapingsteknikken lider imidlertid av en del problemer:

- Krever at brukeren må eksplisitt logge seg inn på hver enkelt "legacy"–applikasjon.
- Skjermskraping er tregt og kan føre til ytelsesproblemer.
- "Mainframe workload" øker etterhvert som flere brukere logger inn.

Dette er en teknikk som hovedsaklig benyttes innenfor administrative systemer.

11.5.2 Databasebroer

Databasebroer ("database gateways") er basert på RDA. All kommunikasjon går gjennom en broadapter. Klientapplikasjoner sender SQL–forespørsler til en SQL–tjener (relasjonsdatabase). Databasen prosesserer forespørselen og sender svar tilbake. Broadapteren gir klienten inntrykk av å kommuniserer med en lokal database. SQL–forespørselen kan utføres som gjennom et kommandonivågrensesnitt eller direkte i et programmeringsspråk. For å tilby aksess til ikke–relasjonsdatabaser, f eks IMS, funger enkelte databasebroer også som oversettere, f eks "SQL–til–IMS".

Hovedfordelene ved databasebroer er fleksibilitet og sluttbrukerkontroll. Imidlertid er disse broene best egnet for applikasjoner som er basert på relasjonsdatabaseteknologier

(SQL til ikke–relasjonsformater har ytelses– og funksjonsbegrensninger). En rekke leverandører, deriblant Sybase, Orcale og Informix, tilbyr proprietære databasebroer. De fleste eksisterende databasebroer støtter lesing og modifikasjon på sentrale tjenere. Støtten for distribuert prosessering, spesielt mellom forskjellige databaser, er foreløpig liten.

11.5.3 Applikasjonsbroer

Applikasjonsbroer ("application gateways") gir adgang til fjernfunksjonalitet ved at klientprogrammer kaller på en fjernfunksjon som typisk kan være en COBOL–rutine, prekompilerte SQL–spørringer eller en komplett transaksjon. En applikasjonsbro kan inkludere skjermskraping i tillegg til fjernfunksjonsaksess. Hovedkarakteristikkene til en applikasjonsbro er:

- Gjemmer tjenerapplikasjonsmiljøet for klientene. Applikasjonsbroer fungerer stort sett som oversetterere mellom applikasjonskall, f eks oversette et RPC–kall til et MVS–basert applikasjon.
- En API og støtteprogramvare for fjernkall trengs av klientapplikasjonen. Klientapplikasjonen inneholder klientkode og grensesnittdefinisjon. Ingen tilleggsprogramvare kreves utover dette.
- Flere operasjonsegenskaper som sikkerhet er viktig, siden applikasjonsbroene kan trenge å agere som betrodde partnere.

Applikasjonsbroer er lite støttet og det finnes få produkter på markedet.

11.5.4 Objektinnpakkere

"Legacy"–systemer er, bl a pga manglende dokumentasjon ofte ikke enkle å modifisere slik at de kan integreres med et nytt system. F eks er det ikke sikkert at det går an å linke inn "stub"– og "skeleton"–filer ved bruk av CORBA. Løsningen på problemet er å pakke inn "legacy"–systemet med "proxy"–objekter ("object wrappers") som kan implementeres i et moderne språk (et "proxy"–objekt er en lokal representant for et fjernobjekt).

Fem broteknikker som ofte benyttes med "proxy"–objekter er:

- Koble et "proxy"–objekt til APIen som legacy–systemet tilbyr.
- En filbasert overføring hvor "proxy"–objektet lager en fil ut ifra IDL–kallet den fikk fra klienten. Denne filen blir overført til "legacy"–systemet.
- Et kommandolinjekall ved å bruke tjenestene til operativssystemet for å kjøre "legacy" –tjenester.
- Bruk av en annen type mellomvare er den som brukes til å pakke inn "proxy"–objektet. Eksempler er RPC–mekanismer, MOM, ODBC, SQL–net og transaksjoner.
- "Screen–scraping" hvor layouten til terminalskjermen blir gjort tilgjengelig for proxyobjektet.

Integrasjonsteknikkene listet ovenfor, kan være et steg på veien til full reimplementasjon av "legacy"–systemet i moderne objektorienterte komponenter. En bør ikke ha som filosofi å reimplementere for å reimplementere, men utskiftningen må begrunnes ved f eks nye krav, modifiserbarhet, o s v.

En trinnvis strategi kan benyttes for "legacy wrapping" og CORBA vil kunne støtte en slik strategi på følgende måte. En starter med å pakke den komplette "legacy"–applikasjonen/ systemet i en IDL "object wrapper". Deretter utvikles grensesnittspesifikasjoner vha IDL som pakker inn "legacy"–funksjonalitet i komponenter etter behov. Denne strategien vil tillate en gradvis erstatning av komponenter med nye komponenter etter hvert som dette blir ønskelig.

11.5.5 Integrasjonsbroer

Integrasjonsbroer ("Integration gateways") integrerer objektinnpakkere med forskjellige akesssteknologier (skjermskraping, fileoverføring, databasebroer og applikasjonsbroer) i et enhetlig rammeverk. "Legacy" integrasjonsbroer tilbyr standard objektorienterte APIer for klientapplikasjoner for å kalle skjermskraping, filoverføring, databaseforespørsler og RPC prosedyrekall.

11.6 Strategi for bruk av mellomvareteknologi

Flere ulike mellomvareteknologier er beskrevet her. I og med at det eksisterer flere standarder og leverandører, bør en ha en strategi for bruk av mellomvareteknologi.

Et eget abstraksjonslag over mellomvarelaget vil gi mellomvareteknologiuavhengighet som vil kunne gi følgende fordeler:

- Ingen avhengighet av en bestemt leverandør
- Minimalisere steder hvor koden trengs å forandres.
- Kodegenering vil kunne gjøres mot forskjellige mellomvareplattformer
- Mellomvare vil kunne gjøres erstattbar som vanlige komponenter.

Mellomvare må være maskin–/plattform– og språkinteroperabel, og uavhengig. Vi vet ikke hvordan fremtidens systemer vil utvikle seg. Spesielt når det gjelder hvilket operativsystem som blir dominerende, hvilken maskinvare en ønsker å kjøre på og hvilke programmeringsspråk en vil benytte. Det vil garantert skje en utvikling og derfor er det viktig at man har et mellomvarerammeverk som tillater en å skifte ut gammel maskinvare med ny, og utvikle deler av et system i et nytt og mer fleksibelt programmeringsspråk, o s v.

Det kan diskuteres hvorvidt mellomvare bør være komponenter som kan erstattes med andre mellomvareteknologier. Man vil ikke kunne utnytte de egenskapene som spesielle mellomvareteknologier tilbyr hvis en bygger en felles modell over all mellomvare. Hvis imidlertid de tjenestene som man trenger fra mellomvarelaget tilbys, eller vil bli tilbudt i

fremtiden fra de mest aktuelle leverandørene, bør man sterkt vurdere et abstraksjonslag for å skjule et bestemt mellomvareprodukt.

12 KONKLUSJON

Prosjektets anbefalte strategi for den fremtidige utviklingen av K2Iser hviler på en visjon om at dagens "system av systemer" over tid går over til ett fremtidig distribuert og komponentbasert/tjenesteintegrert informasjonssystem – d v s et sømløst og integrert K2IS. Et av hovedmålene har vært å formidle den rollen arkitekturer bør ha som et viktig fundament i en overordnet helhetlig K2IS–strategi. Hovedmålsettingen med en arkitekturbasert K2IS–strategi er at den skal bidra til en mer helhetlig og kosteffektiv utvikling, og økt støtte for operativ interoperabilitet, både nasjonalt og internasjonalt. Rapporten har vurdert arkitekturrammeverk, referansearkitekturer og programvare– arkitekturer, samt alternative arkitekturstrategier. Rapporten har også vurdert kommersielle og standardiserte mellomvare– og komponentteknologier, som vil stå sentralt i utviklingen av et distribuert og komponentbasert/tjenesteintegrert K2IS.

En viktig konklusjon er at ved å etablere en felles referansearkitektur for hele Forsvarets K2IS vil en kunne dra ut synergieffekter ved felleskoordinering og samordning av de ulike K2IS–miljøene. Ved å utvikle en nasjonal referansearkitektur vil man kunne nedfelle overordnete og felles strategier for å veilede utvikling av både felles og grenspesifikke delsystemer. Dette vil bidra til å øke interoperabilitet både horisontalt og vertikalt i organisasjonen. En viktig forutsetning i dette arbeidet er at man koordinerer nasjonale arkitekturarbeider med det som foregår av arkitektur– og interoperabilitetsarbeid i NATO. En nasjonal arkitekturaktivitet knyttet til konkrete systemer vil også styrke muligheten til å påvirke arkitekturarbeider i NATO.

Etableringen av en nasjonal referansearkitektur er en K2IS–strategi som støtter et helhetlig perspektiv i forholdet teknologiutvikling, systemutvikling og utvikling av operativ virksomhet, samt målet om et koordinert samspill mellom disse. Fra et teknologiperspektiv er hovedmålet systeminteroperabilitet. Fra et operativt virksomhetsperspektiv er målet å oppnå et koordinert samspill mellom teknologiutvikling, operativ virksomhetsutvikling og systemutviklingsprosessene. Hensikten er å:

- oppnå bedre interoperabilitet (både operativ, system og teknisk) og fleksibilitet.
- lette håndtering av kompleksitet
- oppnå økt teknologiuavhengighet og mer kosteffektiv utvikling av systemene
- bedre koordinering av utviklingen av K2ISer i forhold til operative krav
- sikre bedre utnyttelse av teknologiske muligheter, samt integrering og håndtering av "arv" (evolusjon).

En referansearkitektur vil være gjenstand for jevnlige revisjoner. Endringsprosesser i den operative virksomheten vil f eks kunne resultere i nye eller endrede krav til den underlig-

gende teknologien, og endringer i den underliggende teknologien vil kunne gi nye muligheter som vil føre til endringer i den operative virksomheten. Siden en referansearkitektur skiller ut de mer "bestandige" delene fra de mer teknologiavhengige delene vil den sørge for at skifte i valg av underliggende teknologier lettere vil kunne gjennomføres.

Det er imidlertid en del utfordringer knyttet til et arbeid med en overordnet nasjonal referansearkitektur. Først og fremst så er det utviklet få arkitekturrammeverk og referansearkitekturer. Det betyr at vi har lite erfaringsgrunnlag av denne type. Videre er problemet her at selv om teknologier for distribuerte komponentbaserte systemer i dag kan sies å være relativt godt etablert, er imidlertid kunnskapen, erfaringen og metodene for å anvende dem fortsatt begrenset. Krav til relevant kompetanse blir dermed sentralt i etableringen av en referansearkitektur.

For å få etablert en felles referansearkitektur ligger også mange av utfordringene på Forsvaret. Det kreves at Forsvaret tar arkitekturarbeidet inn som en naturlig del av den virksomhetsstrategiske planleggingen. MACCIS, som er et arkitekturrammeverk som allerede eksisterer, anbefales som et utgangspunkt for utviklingen av en felles nasjonal referansearkitektur.

I rapporten er det også påpekt at bruken av kommersielt tilgjengelige og standardiserte mellomvare– og komponentteknologier vil være sentralt i utviklingen av et distribuert og komponentbasert K2IS. Vi vet ikke hvordan fremtidens systemer vil utvikle seg i forhold til hvilke operativsystemer og maskinvare som blir gjeldende og hvilke programmeringsspråk som vil bli benyttet. Det vil skje en utvikling og derfor er det viktig at mellomvaren tillater et heterogent miljø av maskinvare, operativsystemer, programmeringsspråk og nettverksprotokoller.

Hvorvidt det vil etableres et komponentmarked for K2IS tilsvarende komponentmarkedet for f eks helsesektoren er usikkert. Det anbefales å følge med og delta i arbeidet i C4I DSIG innen Object Management Group i fm utvikling av en referansearkitektur. Som et akutt–tiltak anbefales innføring og standardisering på kommersielt tilgjengelig mellomvareteknologi for eksisterende systemer. Dette vil uavhengig av omfanget av komponentmarkedet gi en egen gevinst. Mange av egenskapene til proprietære infrastrukturer er tilgjengelige som tjenester i standardisert og kommersielt tilgjengelig mellomvare. Omsetningen av standardisert og kommersielt tilgjengelig mellomvare gir en pris og utviklingstid i en annen størrelsesorden enn utviklingen og vedlikehold av proprietære infrastrukturer. I utformingen av en referansearkitektur bør det også vurderes hvorvidt en mellomvare bør kunne erstattes med andre.

Denne rapporten har også beskrevet nye objekt– og komponentteknologier. En bør kjenne til utviklingen av disse teknologiene og vurdere hvilke muligheter de kan gi med tanke på framtidige K2ISer.

Visjonen om ett fremtidig distribuert og komponentbasert/tjenesteintegrert K2IS gir grunnlag for å formulere og samordne prinsipper og retningslinjer for videre arbeid. En referansearkitektur som beskriver og fanger opp sammenhengen mellom operativ virksomhet og

teknologiske muligheter vil stå sentralt i realiseringen av visjonen. En forutsetning for at arbeidet skal lykkes er at kompetanse innen operativ virksomhet, systemutvikling og relevante teknologier forenes.

Litteratur

(1) NATO MAS (1995): AAP–6(U) – NATO Glossary of Terms and Definitions (English and French), Military Agency for Standardization, Januar 1995.

(2) NATO MAS (1993): AAP–31(U) – NATO Glossary of Communication and Information Systems, Terms and Definitions, Military Agency for Standardization, June 1993.

(3) NATO MAS (1994): AdatP–2(G) – Automatic Data Processing (ADP) NATO Glossary (English and French), Military Agency for Standardization, November 1994 (NATO Unclassified).

(4) NATO (1999): NATO Policy for C3 Interoperability, AC/322–WP/72, 1999 (NATO Unclassified).

(5) Lødøen S, Bråthen K, Jenssen A C, Kværnø O, Nordø E, Rose K, Størdal J M, Veum K (1998): Forslag til noen systembetegnelser i Sjøforsvaret, FFI/NOTAT–98/01312, Forsvarets forskningsinstitutt.

(6) Malerud S, Bråthen K, Mevassvik O M, Rose K, Thorsen U (1999): (U) Referanseledesessystem – Beskrivelse av dagens ledelsessystem for maritime operasjoner, FFI/RAPPORT–99/03590, Forsvarets forskningsinstitutt (Konfidensielt).

(7) Urdahl M (1999): (U) Kartlegging av maritime K2IS, FFI/NOTAT–99/05859, Forsvarets forskningsinstitutt (Begrenset).

(8) Bråthen K, Mevassvik Ole M, Leere Anton B, Aas Johan H (1996): (U) KKI i Sjøforsvaret – Innledende kartlegging, FFI/NOTAT–96/04258, Forsvarets forskingsinstitutt (Konfidensielt)

(9) Stallings W (1995): Network and Internetwork Security: Principle and Practice, Prentice Hall, Englewood Cliffs, New Jersey, 1995.

(10) Coulouris G, Dollimore J, Kindberg T (1994): Distributed Systems: Concepts and Design, Addison–Wesley, New York, 1994.

(11) Kim W (1995): Modern Database Systems – The Object Model, Interoperability and Beyond, ACM Press and Addison–Wesley, 1995.

(12) Nygaard K, Sørgaard P (1987): The Perspective Concept in Informatics. In: *Computers and Democracy: A Scandinavian Challenge* (Bjerknes G, Ehn P, Kyng M), Avebury Gower Publ. Comp. Ltd, Aldershot, 371–393, 1987.

(13) ISO/IEC (1995): ITU–T X.901 | ISO/IEC 10746–1 ODP Reference Model Part. 1 Overview.

(14) Butler Group (1998): Component–Based Development – Application Delivery and Integration Using Componentised Software, Management Guide, UK.

(15) IEEE (1990): IEEE STD 610.12–1990 – IEEE Standard Glossary of Software Engineering Terminology, The Institute of Electrical and Electronics Engineers, Inc. (IEEE).

(16) C4ISR AWG (1997): C4ISR Architecture Framework Version 2.0, C4ISR Architecture Working Group.

(17) Neple T, Aagedal J Ø, Bjanger B W, Berre A–J (1999): MACCIS – Minimal Architecture for CCIS in the Norwegian Army, ST40 F99016, Sintef Telecom & Informatics (Restricted).

(18) Farhoodi F, Kennedy J (1994): PD4 – Report on Initial Requirements Definition, Technical and Research Objectives and Prioritisation of Techniques, PD4 Issue B, EUCLID RTP 6.1, Logica UK Limited.

(19) FO (2000): Forsvarets fellesoperative doktrine, del A.

(20) FO (2000), Forsvarets fellesoperative doktrine, del B.

(21) Winjum E, Hedenstad O–E, Sletten G (2000): Kartlegging av operative informasjonssystemer, FFI/RAPPORT–2000/02034, Forsvarets forskingsinstitutt (Konfidensielt).

(22) http://www.odmg.org

(23) http://c4i.omg.org

(24) Aagedal J, Berre A–J, Bjanger B W, Neple T, Roark C B (1999): ODP–based Improvements of C4ISR–AF, Sintef Telecom & Informatics.

(25) Aagedal J Ø, Schliemann T, Elvesæther B, Berre A–J (2000): Komponentutviklingsmetodikk med UML, Sintef Telecom & Informatics.

(26) Cockburn A (1999): A Methodology Per Project, Humans and Technology Technical Report, TR 99.04, Oct. 1999, Humans and Technology, Salt Lake City.

(27) Jacobson I, Booch G, Rumbaugh J (1999): The Unified Software Developement Process, MA: Addison–Wesley, 1999, Reading, Massachusetts.

(28) Rumbaugh J, Jacobson I, Booch G (1999): The Unified Modelling Language Reference Manual, MA: Addison–Wesley, 1999, Reading, Massachusetts.

(29) Levis A H, Wagenhals L W (2000): C4ISR Architectures: I. Developing a Process for C4ISR Architecture Design, Inc. Syst Eng 3: 225–247, 2000.

(30) Wagenhals L W, Shin I, Kim D, Levis A H (2000): C4ISR Architectures: II. A Structured Analysis Approach for Architecture Design, Inc. Syst Eng 3: 248–287, 2000.

(31) Bienvenu M P, Shin I, Levis A H (2000): C4ISR Architectures: III. An Object–Oriented Approach for Architecture Design, Inc. Syst Eng 3: 288–312, 2000.

(32) Oldevik J, Berre A–J (1998): UML–Based methodology for distributed systems, SINTEF Telecom and Informatics, EDOC'98.

(33) NC3B (2000): NATO C3 System Architecture Framework, AC/322–WP/0125, 2000 (NATO Unclassified).

(34) D'Souza D F, Wills A C (1999): Objects, Components and Frameworks with UML, The Catalysis Approach. MA: Addison–Wesley, 1999, Reading, Massachusetts.

(35) Emmerich W (2000): Engineering Distributed Objects, John Wiley & Sons, Ltd, Chichester.

(36) NC3B (2000): BI–SC AIS Implementation Strategy, AC/322–N/574, 2000 (NATO Unclassified).

(37) Erman L D, Hayes–Roth F, Lesser V R, Reddy D R (1980): The Hearsay–II Speech–Understanding System: Integrating Knowledge to Resolve Uncertainty, Computing Surveys, Vol. 12, No. 2, June 1980.

(38) Farhoodi F et al (1991): Design of organisations in distributed decision systems, Proceedings of the Workshop on Cooperation Among Heterogeneous Intelligent Agents, AAAI 91.

(39) http://www.omg.org

(40) Bråthen K, et al (2001): Anbefalt fremtidig ledelsessystem for maritime operajsoner, FFI/RAPPORT–2001/02935 (Begrenset).

APPENDIKS

A DEFINISJONER

Under følger en liste med både norske og engelske definisjoner av noen viktige termer som blir brukt i dette dokumentet.

A.1 Norske definisjoner

Kommando: Den myndighet en offiser har til å lede, koordinere og kontrollere militære styrker. (1)

Kontroll: Den myndighet utøvet av en sjef over deler av aktivitetene til en underordnet organisasjon, eller organisasjon som normalt ikke er under hans kommando, som innbefatter ansvaret for å iverksette ordrer og direktiver. Hele eller deler av denne myndighet kan bli overført eller delegert. (1)

Kommando og kontroll (K2): Den myndighet, det ansvar og de aktiviteter som offiserer har ved ledelse og koordinering av militære styrker og ved utførelse av ordre relatert til iverksettelse av operasjoner. (2)

System: Assembly of doctrines, methods, personnel, procedures, equipment or facilities organised to accomplish specific functions. (2), (3)

Data: Formalisert representasjon av informasjon i en form som er egnet for overføring, tolkning og behandling. Operasjoner på data kan utføres av mennesker eller med automatiske midler. (3)

Informasjon: Kunnskap om objekter f.eks. fakta, begivenheter, ting, prosesser eller ideer, inklusive begreper som i en viss sammenheng har en spesiell mening. (3)

Informasjonssystem: Samling av utstyr, metoder og prosedyrer, og hvis nødvendig personell, organisert for å utføre informasjonsbehandlingsfunksjoner. (3)

Kommando og kontroll informasjonssystem (K2IS): Et informasjonssystem som gir militære myndigheter støtte i forbindelse med kommando og kontroll. (2)

Interoperabilitet: Evnen systemer, enheter eller styrker har til å yte tjenester til og aksepere tjenesterer fra andre systemer, enheter eller styrker, og evnen til å bruke de utvekslede tjenestene for å gjøre dem i stand til å operere effektivt sammen. (1).

Arkitektur: Begrepet arkitektur har forskjellig betydning avhengig av konteksten det brukes i. (a) Strukturen av komponenter, deres relasjoner, og prinsipper og retningslinjer som er styrende for deres design og evolusjon over tid. (b) Organisasjonsstruktur av et system eller en komponent. (15)

A.2 Engelske definisjoner

Command: The authority vested in an individual of the armed forces for the direction, co–ordination, and control of military forces. (1)

Control: That authority exercised by a commander over part of the activites of subordinate organizations, or other organization not normally under his command, which encompasses the responsibility for implementing orders or directives. All or part of his authority may be transferred or delegated. (1)

Command and Control (C2): The authority, responsibility and activities of military commanders in the direction and co–ordination of military forces and in the implementation of orders related to the execution of operations. (2)

System: Samling av doktriner, metoder, personell, prosedyrer, utstyr og fasiliteter for å utføre en bestemt funksjon. (2), (3)

Data: A reinterpretable representation of information in formalized manner suitable for communication, interpretation, or processing. (3)

Information: Knowledge concerning objects, such as facts, events, things, processes, or ideas, including concepts, that within a certain context has a particular meaning. (3)

Information System (IS): Assembly of equipment, methods and procedures and if necessary personnel, organized to accomplish information processing functions. (3)

Command and Control Information System (C2IS): Information system which provides military authorities with support for command and control. (2)

Interoperability: The ability of systems, units or forces to provide services to and accept services from other systems, units or forces and to use the services so exchanged to enable them to operate effectively together. (1)

Architecture: Architecture has various meanings, depending upon its contextual usage. (a) The structure of components, their interrelationship, and the principles and guidelines governing their design and evolution over time. (b) Organizational structure of a system or component. (15)

B FORKORTELSER OG AKRONYMER

ACE ACCIS	Allied Command Europe Automated Command, Control and Information System
ACID	Atomicity, Consistency, Isolation, Durability
ADO	Active Data Objects
API	Application Programming Interface
ASP	Active Server Pages
ATCCIS	Army Tactical Command, Control and Information System
Bi–SC	Bi–Strategic Commands
C2	Command and Control
C3	Consultation, Command and Control
C3S	Consultation, Command and Control Systems
C4I	Command, Control, Communications, Computers and Information
C4I DSIG	C4I Domain Special Interest Group
C4ISR	Command, Control, Communications, Computers, Intelligence, Surveillance and Reconnaissance
C4ISR–AF	C4ISR–Architectural Framework
CCM	CORBA Component Model
CIS	Communication and Information Systems
CLI	Command Level Interface
COM	Component Object Model
COM+	Distributed Component Object Model
COMAJF	Commander, Allied Joint Force
CORBA	Common Object Request Broker Architecture
COTS	Commercial Off The Shelf
CSCW	Computer Supported Cooperativ Work
DCE	Distributed Computing Environment
DCOM	Distributed Component Object Model
DLL	Dynamic Link Library
DNA	Distributed interNet Application
EJB	Enterprise JavaBeans
GIS	Geographic Information System
GOTS	Government Off The Shelf
GUI	Graphical User Interface
IDL	Interface Description Language
IIOP	Internet Inter ORB Protocol
IKT	Informasjons– Kommunikasjons Teknologi
IS	Information System
IS–API	Internet Server API
ISO	International Standardisation Organisation
ISSC	Information Systems Sub–Committee
J2EE	Java 2 Enterprise Edition
J2SE	Java 2 Platform Standard Edition
JAF	JavaBeans Activation Framework

JDBC	Java Database Connectivity
JNDI	Java Naming and Directory Interface
JMS	Java Message Service
JSP	JavaServer Pages
JTA	Java Transaction API
K2	Kommando og kontroll
K2IS	Kommando og kontroll informasjonssystem
KKI	Kommando, kontroll og informasjon
MACCIS	Minimal Architecture for Command and Control Information Systems
MCCIS	Maritime Command and Control Information System
MOM	Message–Oriented Middleware
MOTS	Military Off The Shelf
MSMQ	Microsoft Message Queue
MTBF	Mean Time Before Failure
MTS	Microsoft Transaction Server
NAC	North Atlantic Council
NC3	NATO C3
NC3B	NC3 Board
NC3COE	NC3 Common Operating Environment
NC3O	NC3 Organisation
NC3S	NC3 System
NC3SA	NC3 System Architecture
NC3TA	NC3 Technical Architecture
NCIS	NATO C3 Common Interoperability Standards
NCSP	NC3 Commond Standards Profile
NIE	NATO C3 Interoperability Environment
NIETI	NIE Technical Infrastructure
NIF	NATO Interoperability Framework
NIMP	NATO Interoperability Program
NORCCIS	Norwegian Command Control and Information System
NOSWG	NATO Open Systems Working Group
NOTS	NATO Off The Shelf
OCL	Object Constraint Language
ODBC	Open Database Connectivity
ODL	Object Definition Language
ODMG	Object Database Management Group
OLE	Object Linking and Embedding
OMA	Object Management Architecture
OMG	Object Management Group
ONC	Open Network Computing
OO	Object Oriented
OQL	Object Query Language
ORB	Object Request Broker
PCNCEP	Political Consultation and NATO Civil Emergency Planning
POA	Portabel Object Adapter

QMP	Queued Message Processing
RDA	Remote Database Access
RIP	Rolling Interoperability Program
RM–ODP	Reference Model for Open Distributed Processin
RMI	Remote Method Invocation
RPC	Remote Procedure Call
RUP	Rational Unified Process
SAG	SQL Access Group
SINTEF	Stiftelsen for industriell og teknologisk forskning ved NTH
SQL	Structured Query Language
TCP/IP	Transmission Control Protocol / Internet Protocol
UML	Unified Modeling Language
WWW	World Wide Web
XML	eXtensible Markup Language

FORDELINGSLISTE

FFIE **Dato:** 15 september 2001

RAPPORT TYPE (KRYSS AV)	RAPPORT NR	REFERANSE	RAPPORTENS DATO
X RAPP ☐ NOTAT ☐ RR	2000/04582	FFIE/730/134	15 september 2001
RAPPORTENS BESKYTTELSESGRAD		ANTALL EKS UTSTEDT	ANTALL SIDER
UGRADERT		71	96
RAPPORTENS TITTEL		FORFATTER(E)	
ARKITEKTURER FOR KOMMANDO OG KONTROLL INFORMASJONSSYSTEMER Arkitekturer, komponentbasert systemutvikling og teknologier		HAFNOR Hilde, ELVESÆTER Brian, VEUM Kurt A, VIKEN Kjell	
FORDELING GODKJENT AV FORSKNINGSSJEF: Vidar S. Andersen		FORDELING GODKJENT AV AVDELINGSSJEF: Johnny Bardal	

EKSTERN FORDELING

ANTALL	EKS NR	TIL
1		FO/FST
1		FO/SST
1		v/Kom Jon Meyer
1		FO/E
1		v/Udir Kolbjørn Rørnes
3		FO/I
1		v/KK Tor E Wivelstad
1		v/Fkom Torbjørn Sakseide
1		FKN
1		v/KK Arne Morten Grønningsæter
1		FKS
1		v/Kom Lars Fleisje
1		KE
1		v/KK Svein Jacobsen
1		KVINSP
1		v/OK Martin Ramsland
1		FSTS/Sjø
1		v/KK Inge Tjøstheim
1		KNM T/SMOPS
1		v/OK Svein T Sagstuen
1		KNM T/SMOPS/SIS
1		SFK
1		v/KL Knut Morten Hanssen

INTERN FORDELING

ANTALL	EKS NR	
14		FFI–Bibl
1		Adm direktør/stabssjef
1		FFIE
1		FFISYS
1		FFIBM
1		FFIN
1		Elling Tveit, FFIBM
1		Vidar S Andersen, FFIE
1		Karsten Bråthen, FFIE
1		Kjell Viken, FFIE
1		Hilde Hafnor, FFIE
1		Helge Sanden, FFIE
1		Kurt A Veum, FFIE
1		Ole M Mevassvik, FFIE
1		Morten Urdahl, FFIE
1		Anton B Leere, FFIE
1		Kjell Rose, FFIE
1		Bjørn Jervel Hansen, FFIE
1		Ian Bednar, FFIE
1		Ole–Erik Hedenstad, FFIE
1		Arne Cato Jenssen, FFIE
1		Stig Lødøen, FFIE
1		Robert Helseth Macdonald, FFIE
1		Geir Enemo, FFISYS
5		Arkiv, FFIE
		FFI–veven

FFI–K1 Retningslinjer for fordeling og forsendelse er gitt i Oraklet, Bind I, Bestemmelser om publikasjoner for Forsvarets forskningsinstitutt, pkt 2 og 5. Benytt ny side om nødvendig.

EKSTERN FORDELING

ANTALL	EKS NR	TIL
1		LFK
1		FTD
1		v/Sj Ing Per Anders Jørgensen
1		Brian Elvesæter, SINTEF
		www.ffi.no

INTERN FORDELING

ANTALL	EKS NR	