FFI RAPPORT

MELLOMVARE

Rolf Rasmussen

FFI/RAPPORT-2003/00462

FFIE/855/134

Godkjent
Kjeller 31. januar 2003

Vidar S Andersen
Forskningssjef

MELLOMVARE

RASMUSSEN, Rolf

FFI/RAPPORT-2003/00462

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-2003/00462	UNCLASSIFIED	29
1a) PROJECT REFERENCE	**2a) DECLASSIFICATION/DOWNGRADING SCHEDULE**	
FFIE/855/134	-	

4) TITLE

MELLOMVARE

MIDDLEWARE

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

RASMUSSEN, Rolf

6) DISTRIBUTION STATEMENT

Approved for public release. Distribution unlimited. (Offentlig tilgjengelig)

7) INDEXING TERMS

IN ENGLISH:	IN NORWEGIAN:
a) Middleware	a) Mellomvare
b) Information infrastructure	b) Informasjonsinfrastruktur
c) Network Centric Warfare	c) Nettverksbasert Forsvar
d) Architecture	d) Arkitektur
e) Integration	e) Integrasjon

THESAURUS REFERENCE:

8) ABSTRACT

This report is the result of a study on middleware as being one of the essential concepts in the construction of the information infrastructure for Network Centric Warfare.

The purpose of this report is to elaborate on the different aspects of middleware in information systems, under the assumption that future systems development will be architecture-focused, and that the systems developed should fit into an environment of open, distributed systems. The relation between architectual models and middleware, is presented. Some categories containing middleware with similar characteristics, are described. A wide collection of desirable middleware properties, is also presented.

9) DATE	AUTHORIZED BY	POSITION
	This page only	
31. January 2003	Vidar S Andersen	Director of Research

ISBN 82-464-0694-9

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE
(when data entered)

INNHOLD

Side

MELLOMVARE

1 INNLEDNING

Denne rapporten er et resultat av en litteraturstudie høsten 2002 i FFI-prosjekt 855 ”Programstøtte FIS/O”, delprosjekt Informasjonsinfrastruktur. Formålet er å beskrive mellomvare med utgangspunkt i arkitektur-tenkning og generaliserte modeller for konstruksjon av informasjonssystemer. Mellomvare omtales i denne rapporten med en inndeling i følgende kategorier, som dels kan være overlappende:

- Mellomvare som en integrert del av en komponentmodell
- Teknologier som muliggjør ad-hoc nettverk (også benevnt ”Service Discovery”)
- Basis mellomvare (også benevnt ”connector-teknologi”)
- Løsninger for Enterprise Application Integration (EAI)

Videre gis en bred oversikt over ønskede egenskaper ved åpne, distribuerte systemer. Dette er egenskaper som i hovedsak vil kunne realiseres ved hjelp av mellomvare.

Rapporten gir en overordnet beskrivelse av mellomvare til bruk i distribuerte informasjonssystemer. Den tar mål av seg til å gi en forståelig innføring i hva mellomvare-begrepet omfatter og en omtale av sentrale mellomvare-konsepter, samt referanser til kilder for mer utfyllende informasjon.

Som mange har erfart fra andre områder, har vi også her fått bekreftet at ”jo mer man vet, jo mer vet man at man ikke vet”. Studiene har avdekket stadig mer avanserte og kompliserte tekniske beskrivelsesmodeller. Begrepet ”mellomvare” har vist seg å være noe alle har en intuitiv forståelse av hva er, men når man skal være presis og gjøre avgrensninger, blir det vanskeligere. Det samme oppleves for øvrig med begrepet ”arkitektur” i programvare-sammenheng.

Vårt utgangspunkt er en situasjon der følgende elementer står sentralt:

- Fremtidens Forsvar skal være nettverksbasert. Et nettverksbasert Forsvar (NbF) bygger på en forventning om at all nødvendig informasjon er gjort tilgjengelig på samme logiske nett – heretter kalt Infostrukturen. Det forutsettes høy grad av interoperabilitet mellom systemer i Infostrukturen. Aktørene inndeles i rollene som sensor, beslutningstaker og effektor.
- Arkitektur vil være et hovedfokusområde i arbeidet med å utvikle Forsvarets fremtidige informasjonssystemer. Et mål for dette arbeidet er teknologiuavhengig interoperabilitet, som skapes ved hjelp av presise og konsistente modeller på ulike abstraksjonsnivåer, med veldefinerte overganger til kjørbar programkode.
- Fremtidig systemutvikling vil gå i retning av åpne, distribuerte systemer bygget på objektorienterte og komponentbaserte modeller. I denne konteksten kan mellomvarens rolle beskrives som ”limet” som holder applikasjonskomponentene sammen.

Dette arbeidet tar i noen grad utgangspunkt i de vurderinger som er gjort i FFI rapport 2000/04582 ”Arkitekturer for kommando og kontroll informasjonssystemer” (Hafnor). FFI rapport 2002/03973 ”Informasjonsinfrastruktur for NbF” (Hedenstad) er et annet viktig bakgrunnsdokument.

Integrasjon av eksisterende systemer – ofte kalt *legacy* eller ”arven” – er i dette arbeidet vurdert ut fra et overordnet perspektiv, og uten å gå i detalj på de ulike verktøy og produkter som finnes på dette området. Se for øvrig omtale av løsninger for Enterprise Application Integration (EAI).

2 ARKITEKTUR-BETRAKTNINGER

Til tross for at denne rapporten skal handle om mellomvare, er det ønskelig å starte med omtale av arkitektur. Hensikten er å få en ”ovenfra og ned” (top-down) tilnærming til emnet mellomvare. Dette kapitlet vil argumentere for at veien til teknologiuavhengighet går via større grad av modellering og arkitekturfokus.

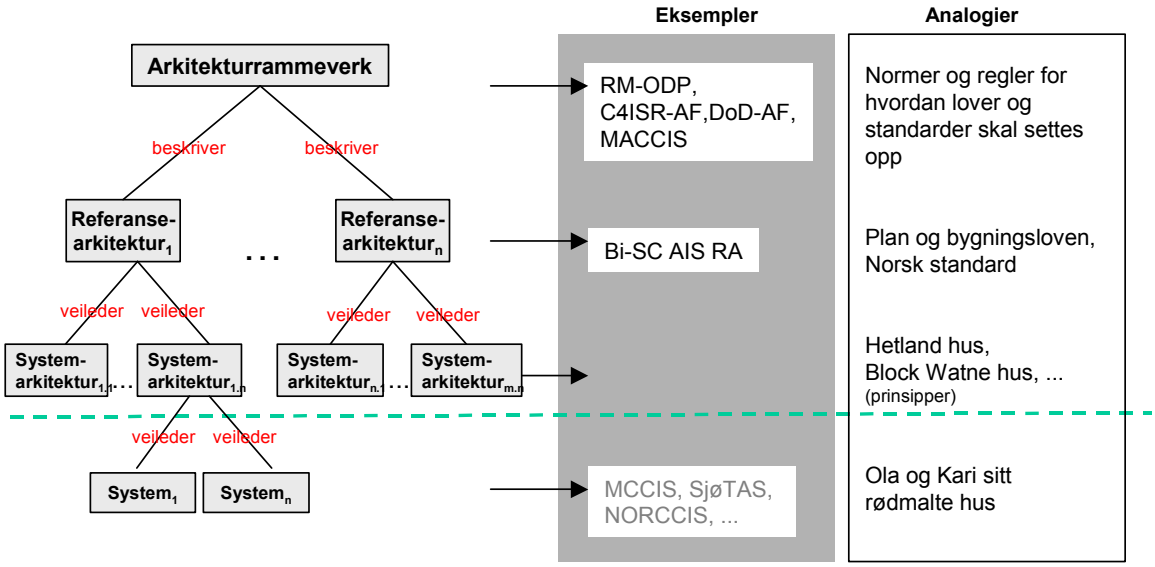
Et viktig aspekt ved arbeidet med arkitekturer for bygging av informasjonssystemer har vært å inkludere en modell av virksomheten som systemene skal støtte. Dette for å sikre at man er i stand til å analysere behovet for informasjons- og kommunikasjonssystemstøtte og tilpasse systembyggingen til det.

Bruk av omforente modelleringsteknikker – som UML – skal sørge for at modellen formidler en felles forståelse av det som er modellert. Detaljeringsmekanismer i modellspråket gjør det mulig å utvikle presise modeller, som i sin ytterste konsekvens vil kunne nærme seg ønskemålet om ”eksekverbare spesifikasjoner”. Det er også viktig å sikre konsistente overganger mellom abstraksjonsnivåene, slik at effekten av fremtidige endringer i overordnede forutsetninger lar seg etterspore på en enkel måte og kan hensyntas på de lavere nivåer i modellen, samt i de ferdig bygde systemene.

2.1 Arkitekturrammeverk og nivådeling

I systemutviklingssammenheng er det enighet om at arkitektur er viktig. Men det er fort å trå feil når man ferdes blant de ulike abstraksjonsnivåene og perspektivene (viewpoints) som finnes innenfor arkitekturområdet. Det er vanskelig å finne konkrete eksempler på systemarkitekturer – det sies at området ennå er umodent.

Som startpunkt på øverste nivå av arkitekturområdet finner vi begrepet arkitekturrammeverk. Rammeverket danner grunnlaget for beskrivelse av referansearkitekturer som igjen legges til grunn for mer spesifikke systemarkitekturer (se nedenstående figur fra Hafnor).



Figur 2.1: Nivåer av arkitekturer (basert på Hafnor)

Som analogiene fra husbyggingsbransjen antyder, vil det kunne være ganske store sprang mellom de ulike abstraksjonsnivåene. Ettersom mye av arkitekturdebatten har foregått på rammeverk-nivå, er det lett å forstå at det for mange kan være hardt å henge med. Det er fristende å sammenligne den typiske programmerers manglende sans for "alt snakket om arkitektur" med håndverkerens nedlatende holdninger til de "byråkratene" som steller med lover, regler og forutsetninger.

Det faktum at figuren ikke viser noen eksempler på systemarkitekturer er en indikasjon på at man i praksis ofte beskriver bare de viktigste delene av arkitekturen for et gitt formål. Komplette beskrivelser – som inkluderer alle perspektiver og tilfeller som kan tenkes å være relevante – er det nok vanskelig å finne. Og de som eventuelt finnes er sannsynligvis fullført i etterkant av gjennomført systembygging.

MACCIS definerer systemarkitektur i form av fem "model views" som hver for seg gir en modell av systemet sett i et gitt perspektiv:

- **Context Model** beskriver systemet sett i forhold til dets omgivelser
- **Requirements Model** har til hensikt å beskrive krav til systemet
- **Component Model** beskriver systemets tjenester, informasjon, komponenter og interaksjoner
- **Distribution Model** beskriver den logiske distribusjonen av systemet
- **Realisation Model** beskriver realiseringen av systemet i form av teknologi og delsystemer

RM-ODP beskriver også fem perspektiver, men med en annen inndeling og andre begreper. Der er perspektivene Enterprise, Information, Computational, Engineering og Technology. DoD-AF opererer med perspektivene Operational View, Technical View og Systems View.

Ideen her er at det ikke er mulig å formidle alle forhold gjennom ett og samme "bilde", men at summen av de ulike perspektivene skal gi en samlet forståelse av helheten.

2.2 Transparens-egenskaper

Fra arkitekturrammeverket Reference Model for Open Distributed Processing (RM-ODP) har vi fått spesifisert et sett med ”distribution transparencies” – altså områder der man ønsker egenskaper som skjuler det faktum at vi har et distribuert system. RM-ODP definerer følgende transparencies, som også trekkes frem i MACCIS:

Transparency	Description
Access	masks data representation and invocation mechanisms for services between systems
Failure	masks possible failure or recovery of objects to enable fault tolerance.
Location	masks the need for an application to know the location of a service in order to call it.
Migration	masks from an object the ability of a system to change the location of that object. Migration is often used to achieve load balancing and reduce latency.
Relocation	masks relocation of an interface from other interfaces bound to it.
Replication	masks the existence of several object copies for the purpose of stability, availability and performance.
Persistence	masks from an object the *deactivation* or *reactivation* of other objects.
Transaction	masks the activities amongst a configuration of objects to achieve consistency.

I tillegg peker MACCIS på følgende relaterte områder som en generell sjekkliste for forhold som bør tas med under ”andre systemkrav”:

- Communication
- Concurrency
- Synchronization
- Configuration
- Naming

Flere områder kan defineres ut fra spesifikke behov, eksempelvis ”sikkerhet”. Poenget her er at slike transparens-egenskaper representerer begreper som kan benyttes i overordnede beskrivelsesmodeller. Når modellene detaljeres og man på grunnlag av dem utvikler en beskrivelse av systemet som skal realiseres, kommer mellomvare-konseptene inn i bildet for å realisere transparens.

2.3 Komponentbaserte distribuerte systemer

Som konseptuelt grunnlag for moderne systemutvikling har den objektorienterte tenkningen fått dominerende gjennomslag. Innenfor det objektorienterte modellkonseptet finnes mekanismer for å gjøre gjenbruk av tidligere utviklet programvare. Gjennom å definere programvarekomponenter ser man for seg et fremtidig ”marked” der ferdige komponenter kan anskaffes og benyttes etter ”plug and play” prinsippet sammen med øvrig egenutviklet programvare.

Moderne systemarbeid handler i stor grad om distribuerte systemer. I en objektorientert komponentmodell ser man for seg at et objekt som har behov for å benytte ressurser som finnes i en gitt komponent, ikke behøver å forholde seg til hvor denne komponenten fysisk befinner seg. Den kan godt eksekvere på en annen datamaskin, som for den sakens skyld kan befinne seg på en internett-forbindelse på et helt annet sted på kloden.

For å få til å realisere distribuerte systemer uten at applikasjonsutviklere skal behøve å forholde seg til systemets "distribuerte" egenskaper, tyr man til fenomenet "mellomvare". Mellomvaren får altså enkelt sagt oppgaven med å realisere utvalgte deler av de tidligere omtalte transparens-egenskapene.

2.4 Gjenbruk - produktfamilier

Gjenbruk av programvare er en av de største utfordringene for systemutviklingsmiljøer verden over. Forventningene var store den gang de objektorienterte prinsippene begynte å bli allment akseptert. Men objektorientering er i seg selv ingen lettvint oppskrift på å få til gjenbruk. I teorien lar det seg greit nok gjøre, men i praksis sliter man med språkavhengige detaljer og ulike miljømessige forutsetninger for øvrig.

Komponent-paradigmet – der man rekursivt helt ned mot det enkelte objekt kan betrakte hver enhet som selvstendige og uavhengig "utførbare" – har betydd et betydelig steg fremover i retning av praktisk gjenbrukbarhet.

Den alminnelige oppfatningen er at komponenter er å betrakte som utførbare moduler. Komponentenes viktighet fokuseres vanligvis ikke før i implementerings- og utrullingsfasene av systembyggingen. For å få maksimalt utbytte av komponent-paradigmet må det gis fokus også i de tidligere fasene av livssyklusen. Hvis man kan identifisere gjenbrukbare komponenter på spesifikasjonstidspunktet så vil det forenkle prosessen betydelig.

Et annet nyttig prinsipp, som med fordel kan kombineres med komponentbasert tenkning, er det å sørge for at fellestrekk ved de ulike systemene som utvikles blir identifisert og definert (gjerne i form av komponenter), slik at de kan fremstå som gjenbrukbare for flere systemer.

En slik kjerne av veldefinert programvare med høy kvalitet vil danne basis for en produktfamilie. Ut fra den kan man ta frem ulike produktlinjer der man utnytter de felles byggestenene i kombinasjon med skreddersydde løsninger. Slik kan man tenke seg en familie av produkter med felles basis funksjonalitet, men som henvender seg til ulike målgrupper med fristende spesialtilpasninger.

I prinsippet kan elementene også kjøpes. Begrepet COTS – Commercial Off The Shelf – brukes for å benevne slike kommersielt tilgjengelige byggestener.

Konseptet med produktlinjer som springer ut av en produktfamilie er i prinsippet svært enkelt. Utfordringen ligger i en ennå manglende eller i beste fall umoden metodestøtte for det å arbeide på en slik måte ut fra komponentbasert tenkning.

Den viktigste diskusjonen i forhold til innholdet i komponent-begrepet ligger i ønsket om å se på komponenter som byggestener på utviklingstidspunktet og ikke bare på kjøretidspunkt (som utførbare moduler). Når vi har et marked for COTS-komponenter som kan brukes til å bygge opp høynivå realiseringsmodeller med en presisjon som gjør det mulig å automatisere deler av det videre arbeid frem mot kjørbar kode, så ser man for seg store fremskritt i overordnet produktivitet på systemutviklingsområdet.

2.5 Komponentteknologier

Vi har tre dominerende komponentmodeller i dagens marked:

- Enterprise Java Beans (EJB)
- Microsoft COM+ (inkludert i .Net)
- CORBA Component Model

De beskrives nærmere i tilknytning til de relaterte mellomvare-teknologiene. Disse tre utgjør alternative valg som basis for tradisjonell systemutvikling basert på en klient-tjener modell. To av dem er avhengig at man baserer seg på henholdsvis Java som programmeringsspråk eller Microsoft som plattform-leverandør. I tillegg har vi den "plattform-nøytrale" CORBA Component Model, som i stor grad bygger på EJB men uten binding til Java.

Disse tre beskrives av noen som tjener-baserte (server-oriented) eller "enterprise" konsepter som bygger på en tradisjonell n-lags klient-tjener arkitektur. Som et alternativ snakkes det stadig mer om tjeneste-baserte (service-oriented) konsepter, som på en bedre måte støtter utvikling av peer-to-peer (P2P) orienterte distribuerte systemer. Et eksempel er komponentrammeverket OpenWings, som er et lovende konsept for integrasjon av ulike teknologier i P2P-orienterte omgivelser.

Peer-to-peer defineres i Internett-sammenheng som en klasse av applikasjoner som utnytter desentraliserte ressurser i ytterkanten av Internett, som opererer under forhold preget av ikke-permanente nettilknytninger og er autonome i forhold til sentrale tjenere i nettverket. Eksempler på slike applikasjoner er Napster, Gnutella og KaZaA. Teknologigrunnlaget for disse er proprietært. JXTA pekes på som "open source" tilnærmelsen til P2P. Mer om dette i Mellomvare-kapitlet.

2.6 Model Driven Architecture (MDA)

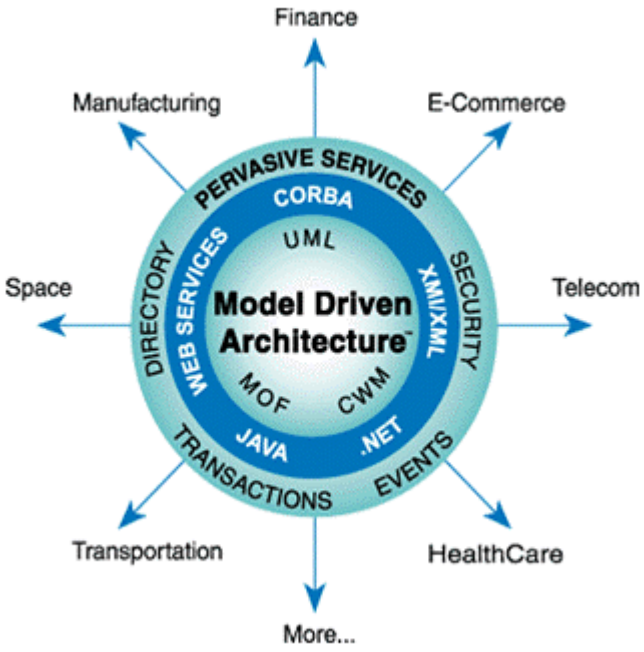
MDA er et initiativ fra Object Management Group (OMG) som fokuserer på bruk av modeller på flere abstraksjonsnivå som grunnlag for å utvikle plattformuavhengige løsninger. Modelleringen legger opp til konsistente overganger fra virksomhetsnivå og ned til realisering av systemer for å understøtte virksomheten.

MDA har som bærende ide å gjøre det mulig å generere mellomvare-spesifikk kode for valgfri plattform, gitt en tilstrekkelig presis modellbeskrivelse av systemet. Viktige begreper i MDA er PIM (Platform Independent Model) og PSM (Platform Specific Model). Med plattform menes "teknologiske detaljer som er irrelevant for den fundamentale funksjonaliteten i systemet".

Ideen er at utviklerne etablerer en PIM med tilstrekkelig presisjon, bl.a. når det gjelder grensesnitt. Den kan så transformeres til en PSM, som i sin tur realiseres som en implementasjon av systemet.

I UML 2.0, som ventes lansert i 2003, regner man med at det vil komme ytterligere støtte for å gi modellene den nødvendige presisjon. I tillegg kreves det at verktøy-industrien utvikler den nødvendige støtten for dette, selv om det allerede finnes eksempler på verktøy som kan bidra med kodegenerering i samsvar med MDA.

Men det er altså viktig å huske at MDA ikke er et implementasjons-rammeverk (slik som for eksempel CORBA), men et rammeverk for å utvikle standarder og verktøy til bruk i utviklingsprosessen. Vi befinner oss altså på et litt høyere abstraksjonsnivå, jfr den innledende omtalen av nivåer i arkitektur.



Figur 2.2: OMG's Model Driven Architecture (MDA)

Figuren illustrerer hvordan man fra en kjerne av UML-modeller (i samsvar med MOF og CWM, se definisjonsliste) gjør bruk av teknologier som for eksempel Java og .Net for å realisere bl.a. de "gjennomgående" (pervasive) tjenestetypene

- events
- transaction
- directory
- security
- persistence (ikke vist på figuren, men inkludert i spesifikasjonen)

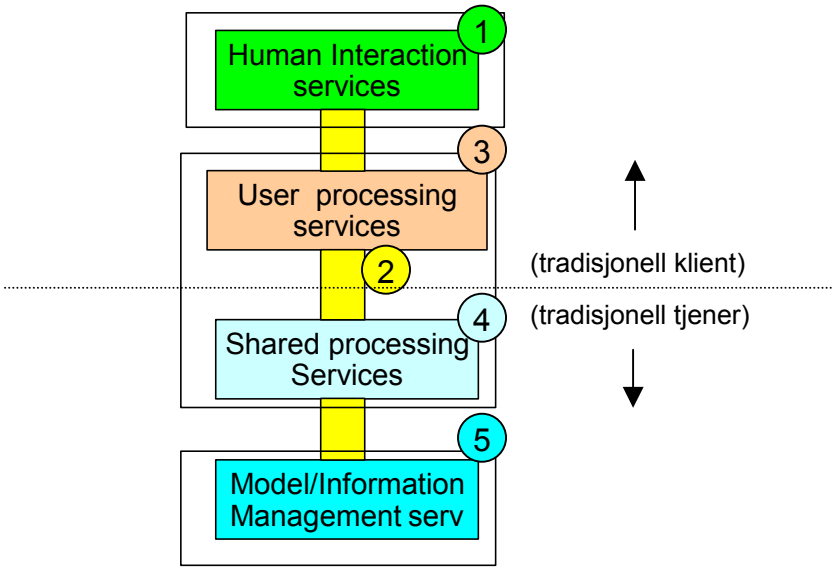
Den observante leser vil her kunne se en viss likhet mellom disse og noen av de tidligere omtalte "transparencies" vi har fra RM-ODP.

2.7 Referansemodell fra MACCIS

I MACCIS versjon 2.0 definerer Sintef et arkitektur-rammeverk rettet mot kommando- og kontrollsystemer. Her finner vi også generaliserte, abstrakte modeller som kan være nyttige når man skal analysere distribuerte informasjonssystemer.

2.7.1 4-lags modell

Som et grunnlag for å analysere mellomvare har vi valgt å trekke frem en 4-lags referansemodell som er definert i MACCIS. I beskrivelsen nedenfor kan vi identifisere den typiske mellomvaren som de gule boksene merket (2) som ivaretar kommunikasjonen mellom de ulike lagene. Spesielt vil vi i overgangen mellom det brukernære (user processing) og det sentrale (shared processing) finne kommunikasjonsmellomvaren som ivaretar dialogen mellom klient og tjener(e).



Figur 2.3: En 4-lags referansemodell for "Enterprise" systemer

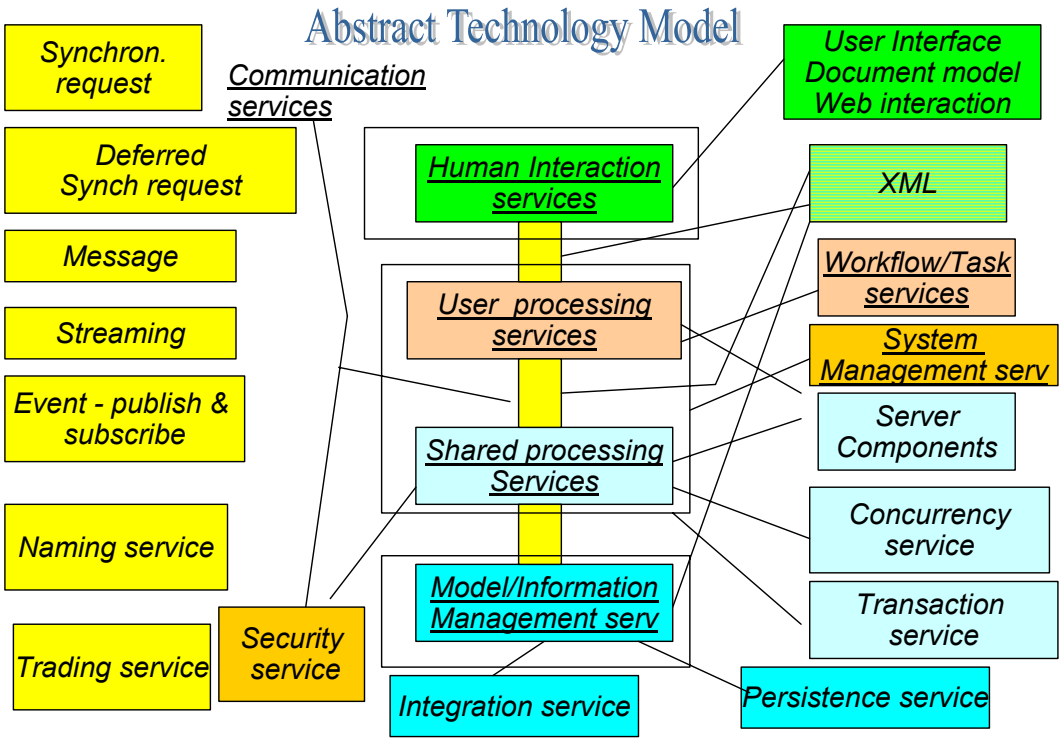
De ulike delene av figuren beskrives som følger:

1. *Human Interaction service* er ansvarlig for brukergrensesnittet og dialogen med bruker.
2. *Communication services* er ansvarlig for sammenkoblingen mellom de ulike lagene. I tillegg til nevnte kommunikasjonsmellomvare kunne man tenke seg å kalle de øvrige to områdene for henholdsvis presentasjonsmellomvare og databasemellomvare.
3. *User processing service* er den delen av prosesseringen som ivaretar den brukernære funksjonaliteten. Denne delen kan typisk inkludere sekvenshåndtering og arbeidsflyt.
4. *Shared processing service* er den delen av prosesseringen som er ansvarlig for fellestjenester som kan brukes av mange. Også kalt *Business processing.*
5. *Model/Information Management service* er ansvarlig for fysisk datalagring og administrasjon av data.

Denne logiske arkitekturen er meget generell og kan passe inn i ulike fysiske arkitekturer, alt fra det å ha alt i en sentralisert applikasjon, til en generell n-lags klient-tjener med flere ulike tjenermaskiner. På figuren er det angitt det tradisjonelle skillet mellom klient og tjener.

2.7.2 Implementasjonsnøytral teknologimodell

Den 4-lags klient-tjener modellen er brukt som utgangspunkt for å peke på tjenester som det typisk vil være behov for i distribuerte systemer, og som støttes av ulike mekanismer innenfor hver av utviklings- og kjøretidsplattformene.



Figur 2.4: Implementasjonsnøytral teknologimodell (Sintef)

Figuren viser en implementasjonsnøytral og abstrakt teknologimodell basert på 4-lags modellen med tillegg av ulike typer tjenester som finnes i typiske omgivelser. I MACCIS-rapporten er det vist hvordan denne modellen kan brukes for å identifisere hvilke mekanismer som støtter disse tjenestene innenfor J2EE og Corba Component Model.

Vi skal ikke gå nærmere inn på disse sammenligningene her, men bare peke på den vektleggingen som er relevant i forhold til vårt mellomvare-fokus. De deler interaksjonstjenestene inn i følgende områder:

- Synkrone kall (*synchronous* og *deferred synchronous* i figuren)
- Asynkron meldingsbasert kommunikasjon
- Hendelsesbasert kommunikasjon (*event – publish/subscribe*)
- Flyt-orientert kommunikasjon (*streaming*)

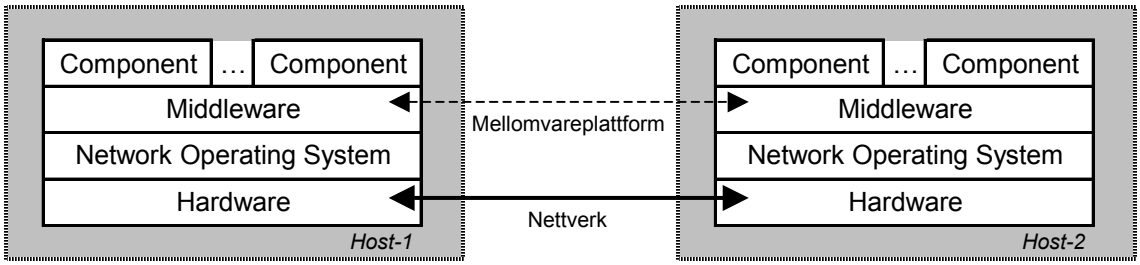
Innenfor kommunikasjon påpekes også behovet for generelle navne- og oppslagstjenester (*naming/trading*). I tillegg pekes det på behov for støtte til felles prosessering på områdene

- Concurrency
- Transaction
- Persistence
- Security

Vi legger til grunn at dette er områder som er egnet for å gjøre sammenligninger av mellomvare.

3 MELLOMVARE

Vi tar utgangspunkt i at det i det foregående er redegjort for mellomvarens overordnede rolle i det å utvikle distribuerte informasjonssystemer. Enkelt sagt skal mellomvaren utgjøre et bindeledd mellom applikasjonslogikken i systemet (de funksjonell delene) og de teknisk relaterte forhold som må hensyntas. En hensiktsmessig mellomvare skal forenkle applikasjonsutviklingen ved å tilby et enhetlig grensesnitt ned mot operativsystemer, nettverk og maskiner. Mellomvaren skal maskere eventuell heterogenitet i de underliggende nivåer.



Figur 3.1: Mellomvare i distribuerte systemer (Emmerich)

Mellomvare forutsettes å ligge over Nettverkslaget (nivå 3 av 7) i OSI-modellen. Det betyr enkelt sagt at de grunnleggende kommunikasjonsforutsetninger må være tilstede.

Dette kapittelet har følgende struktur:

- Først omtales noen grunnleggende konsepter for mellomvare. Fokuset er på kommunikasjon, det være seg synkront eller asynkront, og det forutsettes veldefinerte grensesnitt. Mellomvare på dette nivå kan betegnes som basis mellomvare eller connector-teknologier. Her nevnes bl.a. CORBA og Java RMI.
- Mellomvare som er en del av et mer omfattende komponentrammeverk eller teknologisk plattform. Her finner vi de kjente teknologiene EJB og Microsoft .Net. Den plattformnøytrale CORBA Component Model omtales også her.
- Løsninger for Enterprise Application Integration. Disse er særlig egnet for integrasjon av eksisterende systemer som ikke har blitt bygget for å kommunisere med andre systemer.
- Teknologier som muliggjør ad-hoc nettverk, der det er fokus på egenskaper for ”Service Discovery” og støtte for peer-to-peer nettverk mellom aktører

3.1 Grunnleggende teknologikonsepter

3.1.1 Synkron og asynkron kommunikasjon

De første mekanismer for å håndtere det som siden har fått betegnelsen ”mellomvare”, baserte seg på fjernprosedyrekall (RPC = Remote Procedure Call), dvs at klienten kaller en prosedyre som er lokalisert på tjeneren. Denne mekanismen er i sin natur synkron, dvs at klienten er blokkert mens den venter på svaret fra tjeneren.

Meldingsorientert mellomvare (MOM) baserer seg på at det utveksles meldinger som mellomlagres i køer. I sin natur er dette en asynkron kommunikasjon mellom partene.

Mellomvareløsninger vil ofte tilby begge typer kommunikasjon.

3.1.2 Elementer i mellomvare

Britton beskriver mellomvare som bygget opp av følgende elementer:

Category	Element	Example
Networking and interoperability	1. The communications link	TCP/IP
	2. The middleware protocol	SMTP
The programmatic interface	3. The application programmatic interface	CORBA API's
	4. A common data format	XML
Server control	5. Server process control	(resource mgmt)
System administration infrastructure	6. Naming/directory services	Active Directory
	7. Security	
	8. Administration	

Beskrivelsen er rettet mot utvikleren som skal lage sin egen mellomvare. Det advares i boken om at denne beskrivelsen griper dels inn i nettverkslaget (lag 3) og har fokus på grensesnittet ned mot det. Av "høyere nivå" tjenester er boken i tillegg til oppslagstjeneste og sikkerhet også opptatt av at det må finnes mekanismer for administrasjon, dvs. konfigurering, overvåking og daglig drift.

3.1.3 Grensesnittbeskrivelser

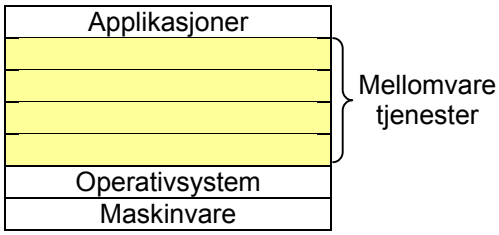
CORBA er en spesifikasjon fra OMG som har blitt en standard innenfor mellomvareområdet, og som de fleste leverandører forholder seg til. Microsoft har sine varianter innenfor .Net-konseptet, mens Sun har Java Remote Method Invocation (RMI) som sin grunnleggende kommunikasjonsløsning. Gjennom kombinasjonen av CORBA's protokoll Internet Inter ORB Protocol (IIOP) og Java RMI, kan man kommunisere på tvers av Java- og CORBA-omgivelser med RMI-IIOP.

Grensesnittbeskrivelsen står sentralt når det handler om mellomvare. CORBA Interface Definition Language er definert av OMG, og gir en språkuavhengig beskrivelse av de komponenter og tjenester som tilbys. Realiseringen av disse komponentene og tjenestene kan være utført i hvilket som helst av de mange programmeringsspråk som tilbyr CORBA støtte (Java, C, COBOL bl.a.).

3.1.4 Flere lag av mellomvaretjenester

I en artikkel av Douglas C Schmidt beskrives en lagdeling der mellomvare-lagene befinner seg mellom applikasjoner på øverste nivå og operativsystem og maskinvare i bunnen. Mellomvaretjenestene deles der inn i:

- Verts-infrastruktur. Mellomvare på laveste nivå som utnytter mekanismer i operativsystem og kommunikasjonsløsning.
- Distribusjonstjenester. Mellomvare som skjuler systemets distribuerte egenskaper og gir uavhengighet av lokasjon, språk, protokoller etc.
- Fellestjenester. "Allmennyttige" mellomvaretjenester, eksempelvis for å håndtere de tidligere omtalte transparens-områdene
- Domene-spesifikke tjenester. Et eksempel på domene kunne være militære kommando- og kontrollsystemer. Generelt er dette nivået av mellomvare ennå et umodent område.



Figur 3.2: Mellomvarelag (Schmidt)

Bevissthet omkring denne type lagdeling vil være nyttig i heterogene miljøer. Det passer også godt overens med rådet fra Hafnor om å etablere et eget abstraksjonslag over mellomvarelaget av hensyn til at man ønsker teknologiuavhengighet. Et slikt abstraksjonslag kan for våre formål tilsvare denne modellens ”domene-spesifikke” lag av mellomvaretjenester.

3.2 Komponentmodeller og tilhørende mellomvare

Her tar vi for oss de tre kjente komponentmodellene som ble nevnt i foregående kapittel, og betrakter dem som mellomvareplattformer.

3.2.1 Enterprise Java Beans (EJB)

Enterprise Java Beans (EJB) er en komponentstandard som igjen er en del av Java 2 Enterprise Edition (J2EE), spesifisert av Sun Microsystems. EJB-komponenter kjøres i et kjøretidsmiljø kalt en container på en applikasjonsserver. Gjennom containeren får komponentene tilgang til en rekke vanlige tjenester som ikke er komponent-spesifikke, blant annet tilgang til databaser, hjelp til transaksjoner, email osv.

Dette sparer utviklere for mye arbeid, siden de da kan konsentrere seg om hovedformålet med komponentene eller ”business logic”. Det finnes en rekke forskjellige applikasjonsservere, også open-source, og mange ”tunge” leverandører (IBM, Sun, Oracle, Bea) støtter J2EE-standarden. Kommunikasjon med en applikasjonsserver foregår via RMI-IIOP protokollen.

Det er tre hovedtyper komponenter i EJB-spesifikasjonen:

- Session Beans
- Entity Beans
- Message-Driven Beans

Session Beans gjelder bare for en sesjon med en bruker, og kan både ha tilstand (dialog) og være til ”engangsbruk”. Entity Beans holder data, som igjen persisteres i en underliggende database. Message-Driven beans er spesifisert for bruk sammen med JMS-teknologien, og responderer på asynkrone meldinger.

3.2.2 CORBA Component Model

CORBA (Common Object Request Broker Architecture) er en objektorientert teknologi for fjernprosedyrekall som er spesifisert av Object Management Group (OMG). Teknologien har eksistert i mer enn ti år, men er fortsatt i utstrakt bruk på grunn av at teknologien er både språk- og plattformuavhengig.

CORBA Component Model (CCM) er utviklet i ettertid (spesifisert som en del av CORBA 3.0). CCM er CORBAs svar på EJB, og minner mye om EJB. Forskjellen er at CCM kan kjøres i en rekke forskjellige miljøer. CCM er foreløpig en umoden teknologi, sammenlignet med EJB.

3.2.3 Microsoft COM+ og .Net

Fra tidligere har Microsoft hatt konsepter som Distributed interNet Application (DNA), med Component Object Model (COM) og Distributed COM (DCOM) som elementer. Alle forutsetter bruk av Windows som kjøretidsmiljø. For enkelhets skyld antas det at disse og tilsvarende løsninger fra Microsoft vil være å betrakte som en del av deres nåværende produktplattform .Net.

Microsoft .Net plattform kan sies å være en konkurrent til EJB-teknologien. Plattformen består av både en utviklingsomgivelse, flere programmeringsspråk (blant annet C#, uttales ”*C sharp*”), produkter og en kjøretidsomgivelse (Common Language Runtime) som kan sammenlignes med en EJB-container. Microsoft .Net er tett integrert med Web Services-teknologien og Windows-plattformen.

3.3 Enterprise Application Integration (EAI)

EAI-området skal dekke en virksomhets (enterprise) behov for å integrere først og fremst de eksisterende systemløsninger som allerede finnes innen virksomheten (arv, legacy). Her har ulike leverandører utviklet produkter som tar sikte på å dekke virksomhetens helhetlige behov for integrasjon. Typiske krav til en integrasjonsløsning er:

- Minst mulig omskriving av eksisterende applikasjoner
- Sentral styring og drift av integrasjonsløsningen

Et populært EAI-konsept innebærer en sentralisert server-løsning som styrer kommunikasjonen med øvrige applikasjoner i en nav-eike topologi. De mest avanserte løsningene av typen ”integration broker” inneholder et bredt spekter av transformasjonsmuligheter mellom formater og protokoller. På den måten kan intergrasjonstjeneren stå som en styringssentral i nettet og sørge for dialog mellom systemer på ulike plattformer.

Integrasjonstjeneren kan på denne måten betraktes som det sentraliserte knutepunktet i virksomhetens informasjonsinfrastruktur.

EAI-løsningene inneholder altså integrerte mellomvare-løsninger som håndterer mange former for grunnleggende mellomvare-tjenester. Majoriteten av EAI-løsninger synes å være basert på J2EE og EJB – muligens fordi det er den mest veletablerte komponentteknologien. IBM’s WebSphere-løsning – som inkluderer det meldingsbaserte produktet MQ Series – er et eksempel. BEA’s WebLogic er et annet velkjent produkt. Andre kjente EAI-leverandører er Tibco og IONA.

Dette er et område der produktene er i rivende utvikling, og konkrete vurderinger av produkter og leverandører blir raskt foreldet. En hensiktsmessig måte å få oppdatert oversikt over EAI-markedet kan være å anskaffe en nyere vurdering fra analyseselskaper som Gartner Group, Butler Group, m.fl.

3.4 Mellomvare som muliggjør ad-hoc nettverk

Til forskjell fra de tradisjonelle konseptene som bygger på klient-tjener tenkning, har vi de mellomvareløsningene som legger til grunn kommunikasjon mellom likeverdige partnere. Dette benevnes som ”peer-to-peer” (P2P) orienterte distribuerte systemer.

Jini med sine dynamiske løsninger for ”Service Discovery”, er et eksempel på en slik teknologi. Andre slike tjenesteorienterte teknologier er Jxta, ebXML og OpenWings. Web Services er også en teknologi som må kategoriseres som tjenesteorientert.

3.4.1 Jini

Jini er en ”Service Discovery” teknologi spesifisert av Sun Microsystems, og kan betraktes som et objektorientert rammeverk for å bygge skalerbare og robuste distribuerte systemer ved hjelp av programmeringsspråket Java. Ved å benytte Jini kan datamaskiner finne hverandre og benytte hverandres tjenester (også software-tjenester) over et nettverk uten at de kjenner til hverandre fra før.

Det eneste en klient må vite, er hva slags tjeneste den ønsker å benytte. Dette spesifiseres ved hjelp av et Java interface, som fungerer som en kontrakt mellom tjenesten og klienten. Jini baserer seg på oppslagstjenere (Lookup Service), og etter å funnet en tjeneste lastes mobil kode ned til klienten for å aksessere tjenesten. Dette kalles et proxy-objekt og innebærer at tjenesten selv har kontroll med hvordan (med hvilken protokoll) den blir aksessert.

Jini-rammeverket består av flere komponenter, blant annet forskjellige protokoller, klasser, interfaces og tjenester, herunder distribuerte events og transaksjoner. Jini er bygget på Suns Remote Method Invocation-teknologi (RMI) som sender objekter over nettverket.

En leasing-mekanisme brukes for å sikre at alle registreringer er ”levende”, det vil si at registrerte tjenester vil forsvinne fra oppslagstjenere dersom de ikke oppdaterer sine ”leieperioder”. Denne mekanismer er unik for Jini, og er en av grunnene til at Jini kan kalles robust.

Jini krever at det eksisterer maskiner i systemet som kjører Java. Jini har med godt resultat vært hovedfundamentet for løsningen ”CoABS Grid” fra DARPA, der formålet er kommunikasjon mellom selvstendige ”agenter”.

3.4.2 Web Services

Web Services er en samling spesifikasjoner utviklet av W3C-konsortiet for at maskiner skal kunne benytte hverandres tjenester over Internett ved hjelp av XML.

De tre mest sentrale standardene i Web Services er

- UDDI (Universal Description, Discovery and Integration)
- SOAP (Simple Object Access Protocol)
- WSDL (Web Services Description Language)

som henholdsvis spesifiserer en oppslagstjeneste, et meldingsformat og en grensesnittbeskrivelse.

Bruk av Web Services må spesifiseres på design-tidspunkt, det vil si før et program kompileres og kjøres. Dette gjør at Web Services foreløpig ikke er spesielt dynamisk. Imidlertid vil Web

Services være et nyttig redskap for å danne nye sammenslutninger av systemer innenfor et lukket område der partene kan stole på hverandres tjenester.

3.4.3 ebXML

En teknologi som er blitt spesifisert parallelt med ovennevnte Web Services er ebXML (Electronic Business using eXtensible Markup Language). Målet med ebXML er at bedrifter skal kunne benytte elektronisk business-informasjon på en interoperabel, sikker og konsistent måte. ebXML er også bygget opp rundt en oppslagstjeneste (ebXML registry), standarder for å beskrive tjenester og standarder for transport. Det er blitt hevdet at ebXML er mer modent enn Web Services i dag, spesielt for ”åpen” kommunikasjon mellom bedrifter. ebXML har i tillegg spesifisert en meldingstjeneste som sikrer garantert levering av dokumenter. Det ses på integrasjon mellom deler Web Services og ebXML, og forhåpentligvis vil dette bli komplementære teknologier.

3.4.4 JXTA

JXTA er en peer-to-peer teknologi basert på et sett med XML-baserte protokoller. JXTA kan derfor kalles for en plattformuavhengig ”Service Discovery” teknologi. Kommunikasjonen mellom endepunkter (peers) er basert på enveis virtuelle forbindelser kalt pipes. JXTA-arkitekturen er tilnærmet desentralisert, og prinsipielt er alle enheter likeverdige. Det er også mulighet for å sende binære meldinger ved hjelp av JXTA og det arbeides både med sikkerhet og en lettvektsversjon.

3.4.5 OpenWings

Openwings er et open-source prosjekt ledet av General Dynamics Decision Systems. Openwings er en komponentmodell for service-orienterte systemer, der målet er å kunne bytte ut ”Service Discovery” mekanismen samt kommunikasjonsprotokollen uten å gjøre endringer i komponenter.

De første versjonene av Openwings er kun basert på Jini, men det arbeides med en plug-in for Web Services. Openwings har som mål å kunne tilby en rekke underliggende tjenester, blant annet sikkerhet, management osv. Det er et krav at maskiner som kjører en Openwings-container kan kjøre Java, men det vil være mulig å benytte tjenester fra en rekke miljøer.

4 KLASSIFISERING AV MELLOMVARE

En klassifisering av mellomvare kan i utgangspunktet gjøres langs flere ulike dimensjoner. Eksempler på karakteristikker er:

- Vi har vært inne på det grunnleggende skillet mellom synkrone og asynkrone mekanismer, og at de fleste kommersielle produkter håndterer begge former.
- Vi har ulike grader av leverandøravhengighet, der CORBA utgjør den eneste ordentlig plattform-nøytrale løsningen både når vi snakker om mellomvare og som komponentmodell.

- Skillet mellom tjener-orienterte og tjeneste-orienterte teknologier er ikke alltid like lett å definere. Det synes som stadig flere leverandører omtaler produktene sine som tjeneste-orienterte.

4.1.1 Inndeling i kategorier

Som en overordnet inndeling og kategorisering av de mellomvarer som er behandlet i denne rapporten, har vi lagt til grunn inndelingen som er vist i følgende oppstilling. EAI-løsninger er ikke tatt med. Vi har her brukt "Service Discovery" som betegnelse på de teknologiene som muliggjør ad-hoc nettverk og støtter tjenesteorienterte peer-to-peer omgivelser.

	Komponent-modell	"Service discovery"	Basis mellomvare
Enterprise Java Beans [1)]	X		
Java RMI/RMI-IIOP			X
Microsoft COM+ og .Net [1)]	X		
Web Services		X	X [2)]
CORBA Component Model [1)]	X		
CORBA			X
Jini		X	
Jxta		X	X
EbXML		X	X [2)]
OpenWings	X	X	

1) Eksklusive sin "tilhørende" basis mellomvare (vist i egen rad)
2) Baserer seg på den samme Simple Object Access Protocol (SOAP)

Figur 4.1: Overordnet inndeling av mellomvareteknologier

Ved å skille ut basis mellomvaren (også kalt connector-teknologien) fra de tre komponent-rammeverkene, har vi fått frem en oppstilling som gir en veiledning til hvilke teknologier det vil være relevant å sammenligne.

I det følgende trekkes det opp ytterligere inndelinger og egenskaper som hver for seg er et mulig grunnlag for å vurdere mellomvare opp i mot.

4.1.2 De åtte vrangforestillingene og Colouris' utfordringer

Sun's Peter Deutsch har fått navnet sitt knyttet til følgende postulat som han har kalt "The eight Fallacies of Distributed Computing":

Essentially everyone, when they first build a distributed application, makes the following eight assumptions. All prove to be false in the long run and all cause big trouble and painful learning experiences.

1. *The network is reliable*
2. *Latency is zero*
3. *Bandwidth is infinite*
4. *The network is secure*
5. *Topology doesn't change*
6. *There is one administrator*
7. *Transport cost is zero*
8. *The network is homogeneous*

Colouris beskriver følgende utfordringer som må hensyntas ved bygging av distribuerte systemer:

1. Heterogenity
2. Openness
3. Security
4. Scalability
5. Failure handling
6. Concurrency
7. Transparency

Sammen med transparens-egenskapene definert i RM-ODP er dette gode eksempler på områder som vil være egnet til å gjøre mellomvarevurderinger opp i mot.

4.1.3 Primære mellomvaretjenester for tradisjonelle behov

På basis av Sintef's teknologimodell som er gjengitt i denne rapporten, samt en foreløpig vurdering av hvilke tjenestetyper som kan karakteriseres som de viktigste i kommunikasjonsmellomvare, har vi satt opp følgende enkle vurderingsmodell.

Kategori	Tjenestetype	Vurdering ut fra gitt behov eller anvendelse
Basis kommuni-kasjon	Synkron kommunikasjon	*1. Vurder viktigheten av hver enkelt tjenestetype* *2. For hver av de aktuelle mellomvareløsningene* *a) Vurderløsningens egnethet innen hver tjenestetype*
	Asynkron (meldingsbasert)	
	Event & pub/sub-håndtering	
Sentrale tilleggs-tjenester	Katalogtjeneste/oppslag	
	Transaksjoner/samtidighet	
	Lagring/persistens	
	Sikkerhet	

Figur 4.2: Vurderingsmodell basert på Sintef

4.1.4 En samling av egenskaper

I denne rapporten har det vært referert mange ulike opplistinger av egenskaper ved distribuerte systemer eller områder som det er viktig å ha fokus på ved utviklingen av slike. Det har vært

- RM-ODP Transparencies (enkelte kilder angir et litt annet sett med begreper, nedenfor benevnt RM-ODP1)
- MACCIS' Concerns
- MDA Pervasive services (her finnes det også noen flere enn de som er nevnt, tilleggstjenestene er nedenfor benevnt MDA1)
- Mellomvare-relaterte tjenester fra Sintef's modell
- Britton's oversikt over elementene som utgjør mellomvare
- Deutsch' åtte vrangforestillinger
- Colouris' utfordringer

Alle disse begrepene er sammenstilt i følgende oversikt.

Begrep \ Kilde	Britton	Colouris	Deutsch	MDA	MDA1	RM-ODP	RM-ODP1	Sintef	MACCIS	Totalt
access						x	x			2
administration	x		x							2
bandwidth			x							1
communication								x	x	2
communications link	x									1
concurrency		x					x	x	x	4
configuration									x	1
data format	x									1
directory/naming	x			x				x	x	4
embedded					x					1
events				x						1
failure/fault tolerant		x			x	x	x			4
heterogenity		x	x							2
interface	x									1
latency			x							1
location						x	x			2
migration						x	x			2
openness		x								1
performance							x			1
persistence				x		x		x		3
process control	x									1
protocol	x									1
real-time					x					1
reliability			x							1
relocation						x				1
replication						x	x			2
scalability		x			x		x			3
security	x	x	x	x				x		5
synchronization									x	1
topology-change			x							1
transaction				x		x		x		3
transparency		x								1
transport cost			x							1
Sum forekomster	8	7	8	5	4	8	8	6	5	59

Figur 4.3: Samling av ønskede egenskaper

Vi ser at det er et stort mangfold i begreper som betegner ønskede egenskaper. Her er vist 33 begreper fra 9 kilder. Det er neppe så enkelt som å si at de begreper som er foreslått flest ganger er de viktigste. Igjen så vil utvalget av egenskaper være avhengig av behovet og den konkrete anvendelsen som er tenkt. Hensikten med å vise denne oversikten er å illustrere mangfoldet.

Oversikten kan i denne sammenheng betraktes som en sjekkliste, samt at det er mulig å gå tilbake til kildene og undersøke nærmere bakgrunnen for hver av egenskapene.

4.2 NbF innebærer strenge krav til dynamiske egenskaper

Innenfor applikasjons-domenet som omfatter Forsvarets kommando- og kontrollsystemer og i en fremtidig NbF-setting der sammenkoblingen av alle informasjonssystemer betraktes som en eneste Infostruktur, ser vi behov for mekanismer som krever mer enn det man i utgangspunktet får i gjeldende CORBA spesifikasjoner. Det gjelder primært de dynamiske egenskapene ved mellomvaren, der vi ser behov for å kunne sette sammen komponenter etter behov på en måte som er umulig å forhånds-konfigurere. Likeledes må dynamisk bortfall av komponenter og tjenester kunne håndteres.

En punktvis opplisting av sentrale krav i forbindelse med NbF:

- Dynamiske oppslagstjenester
- Peer-to-peer basert (no single point of failure)
- Asynkron kommunikasjon
- Funksjonsdyktig også over lave båndbredder
- Sikkerhet

Disse kravene peker klart i retning av løsninger i kategorien ”Service Discovery”.

5 KONKLUSJON

Det gis ikke grunnlag for å gjøre konkrete valg av mellomvare ut fra det som er fremkommet i denne rapporten. Det var heller ikke meningen. Hensikten var å gi en overordnet innføring i hva mellomvare er, og en oversikt over de typer av teknologier som eksisterer.

Vi har tatt utgangspunkt i en overordnet inndeling i grove og dels overlappende grupper (i tillegg til EAI har vi komponentmodell-baserte, ”Service Discovery” relaterte samt basis mellomvare), og funnet frem til et sett av begreper som kan legges til grunn for et videre arbeid i retning av en mer detaljert vurdering.

Men et iboende problem er å være i stand til å identifisere de kandidatene som skal klassifiseres. Som en parallell til at man må ”unngå å sammenligne epler og pærer”, vil spekteret av mellomvare-løsninger fra de største EAI-produktene ned til de mest kommunikasjonsnære enkelt-protokollene bli som en hel fruktkurv å regne.

5.1 NbF-relaterte krav

Gitt vårt utgangspunkt i å vurdere løsninger som støtter et nettverksbasert forsvar, ser vi et behov for best mulig støtte for P2P-løsninger. I konteksten bildeoppbygging basert på et nettverk av beslutnings- og sensorkomponenter, ser vi et stort behov for dynamikk. Det må der være mulig å koble sammen enheter etter behov på en måte som det vil være tilnærmet umulig å forhånds-konfigurere.

Samtidig vil fremtiden også måtte innebære integrasjon av en mengde eksisterende systemer, noe som bør kunne løses ved bruk av EAI. Og tradisjonell systemutvikling støttes trolig best ved bruk av et anerkjent komponentbasert rammeverk. En viktig ressurs på det området bør kunne bli COTS-komponenter som kan gjenbrukes.

En rekke ønsker og behov vil trolig vise seg å være motstridende. Et eksempel: Det må kunne antas at strenge krav til dynamiske egenskaper i utgangspunktet vil innebære store utfordringer for å ivareta nødvendig ytelse og sikkerhet. Resultatet vil måtte bli en avveining der man snakker om grader av det å tilfredsstille ønskede egenskaper, og ikke et enten-eller.

En observasjon er at mellomvare for integrasjon av eksisterende løsninger må møte helt andre krav enn mellomvare for bruk i fremtidige løsninger. I en fremtid vil også morgendagens systemer være ”legacy”. Det er da viktig at de inneholder de riktige grensesnittene for integrasjon med nye løsninger.

5.2 Teknologiuavhengighet

Som tidligere nevnt legger vi til grunn at veien til teknologiuavhengighet går via modellering og arkitekturfokus. En anbefaling er derfor at fremtidig systemutvikling gjøres i forhold til veldefinerte arkitekturmodeller. Og det anbefales spesielt å ha et forhold til utviklingen innenfor Object Management Group (OMG) og videreføringen av deres Model Driven Architecture (MDA).

Utgangspunktet for denne rapporten var nettverksbasert forsvar, arkitekturfokus og utvikling av åpne, distribuerte systemer basert på objektorienterte og komponentbaserte modeller. Rapporten gir grunnlag for å trekke følgende slutninger:

- Infostrukturen som støtte for et NbF vil generelt sett være heterogen, med ulike mellomvaretyper og –konsepter i et samspill
- EAI er velegnet for integrasjon av eksisterende løsninger
- De dynamiske egenskapene ved NbF støttes best av ”Service Discovery” teknologier
- ”Peer-to-peer” synes å være en egnet tilnærmingsmodell når sensor- og beslutningskomponenter skal kobles sammen på en dynamisk måte
- Det må foretas behovsrelaterte avveininger mellom behov for dynamiske løsninger og krav til ytelse og sikkerhet. Konkret er motstriden mellom behovene for ”sentralisert styring og drift” og ”dynamisk rekonfigurering” et eksempel på et område som bør utdypes i det videre arbeid
- Valg av utviklingsmiljø (komponentmodell og leverandører) gir føringer for mellomvare
- Et eget abstraksjonslag over mellomvarelaget er et bidrag til uavhengighet
- CORBA-standarden er og vil være en felles referanse – den eneste plattform-nøytrale

5.3 Mulig videreføring

Denne rapporten gir en oversikt over og et teoretisk grunnlag for nærmere vurdering av mellomvare. En videreføring i retning av konkrete anbefalinger av spesifikk mellomvare vil kreve god teknologisk innsikt i det som skal vurderes. I tillegg vil en slik undersøkelse måtte datostemples, ettersom utviklingen på dette området går fort.

En hensiktsmessig videreføring av dette arbeidet vil være å gjennomføre eksperimentering med utvalgte typer mellomvare, fortrinnsvis innenfor en NbF-relatert setting. Et eksempel kunne være eksperimenter på områder der EAI-baserte integrasjonsløsninger og P2P-basert ”Service Discovery” kan sammenkobles og/eller vurderes opp mot hverandre i konkrete scenarier.

Innenfor dette prosjektet vil dette arbeidet i noen grad bli videreført i Demonstrator-delen av prosjektet, ved at nærmere utvalgte mellomvareløsninger skal benyttes for bildeoppbygging.

DEFINISJONER

Nedenfor forklares de akronymer som er brukt i denne rapporten.

Begrep	Forklaring
CCM	Corba Component Model
CoABS	Control of Agent-Based Systems
COM	Common Object Model (Microsoft)
CORBA	Common Object Request Broker Architecture
COTS	Commercial Off-the-shelf
CWM	Common Warehouse Model (OMG)
DARPA	Defense Advanced Research Projects Agency (USA)
DoD-AF	Department of Defence – Architecture Framework
EAI	Enterprise Application Integration
ebXML	Electronic Business using XML
EJB	Enterprise Java Beans
FIS/O	Forsvarets InformasjonsSystem – Operativ del
IDL	Interface Definition Language
IIOP	Internet Inter-Orb Protocol
J2EE	Java 2 Enterprise Edition
JMS	Java Messaging Service
JXTA	Avledet fra ordet ”juxtapose”, som betyr å sammenstille
MACCIS	Model Architecture for Command and Control Information Systems
MDA	Model Driven Architecture
MOF	Meta Object Facility (OMG)
MOM	Message-Oriented Middleware
NbF	NettverksBasert Forsvar
OMG	Object Management Group
OSI	Open Systems Interconnection
P2P	Peer-to-peer, et ”nettverk mellom likestilte”
RMI	Remote Method Invocation (Java)
RM-ODP	The Reference Model of Open Distributed Processing
SOAP	Simple Object Access Protocol
UDDI	Universal Description and Discovery Interface
UML	Unified Modelling Language
W3C	The World Wide Web Consortium
WSDL	Web Services Definition Language
XML	eXtended Markup Language
XSLT	eXtensible Stylesheet Language Transformation

LITTERATUR

Forfatter	Tittel
Atkinson, Colin et. al.	Component-based Product Line Engineering with UML (Addison-Wesley, 2002)
Britton, Chris	IT Architectures and Middleware (Addison-Wesley, 2001)
Colouris et. al.	Distributed Systems – concepts and design (3rd ed, Addison-Wesley, 2001)
Deutsch, Peter	http://java.sun.com/people/jag/Fallacies.html
Elvesæter, Brian et. al.	MACCIS – Minimal Architecture for C2IS, Part A and B (SINTEF, 2001)
Emmerich, Wolfgang	Engineering Distributed Objects (Wiley, 2000)
Hafnor, Hilde et. al.	FFI rapport 2000/04582 ”Arkitekturer for kommando og kontroll informasjonssystemer”
Hedenstad, Ole-Erik	FFI rapport 2002/03973 ”Informasjonsinfrastruktur for NbF”
OMG	http://www.omg.org/mda/
Putman, Janis R	Architecting with RM-ODP
Schmidt, Douglas C	Middleware for real-time and embedded systems (artikkel, ACM june 2002)
SINTEF	http://www.informatics.sintef.no/~jol/ODP/rmodp_overview.html

FORDELINGSLISTE

FFIE **Dato:** 31. januar 2003

RAPPORTTYPE (KRYSS AV)	RAPPORT NR.	REFERANSE	RAPPORTENS DATO
X RAPP ☐ NOTAT ☐ RR	2003/00462	FFIE/855/134	31. januar 2003
RAPPORTENS BESKYTTELSESGRAD		ANTALL TRYKTE UTSTEDT	ANTALL SIDER
UGRADERT		39	29
RAPPORTENS TITTEL		FORFATTER(E)	
MELLOMVARE		RASMUSSEN, Rolf	
FORDELING GODKJENT AV FORSKNINGSSJEF		FORDELING GODKJENT AV AVDELINGSSJEF:	
Vidar S Andersen		Johnny Bardal	

EKSTERN FORDELING

ANTALL	EKS NR	TIL
1		FO/FST
1		v/ Gunnar Arneberg
1		FO/I
1		v/ Per Trygve Gundersen
1		v/ Tor Einar Wivelstad
1		v/ Svein Morten Olaussen
1		v/ Lasse Halaas
1		FOHK
1		v/ Arve Offigstad
1		v/ Sigurd Iversen
1		FSS FSTS
1		v/ Tor Hylin
1		FLO/IKT
1		v/ Per Anders Jørgensen
1		v/ Torstein Haugland
1		v/ Espen Sundal
1		v/ Leif-Erik Jordan

INTERN FORDELING

ANTALL	EKS NR	TIL
9		FFI-Bibl
1		FFI-ledelse
1		FFIE
1		FFISYS
1		FFIBM
1		FFIN
1		Forfattereksemplar(er)
7		Restopplag til Biblioteket
		Elektronisk fordeling:
		FFI-veven
		Ian Bjørn Bednar (IBB)
		Karsten Bråthen (KaB)
		Bjørn Jervell Hansen (BHn)
		Ole-Erik Hedenstad (OEH)
		Reinert Korsnes (RKo)
		Anton B Leere (ABL)
		Ole Martin Mevassvik (OMM)
		Kjell Rose (KjR)
		Geir Sletten (GSl)
		Tore Smestad (TSm)
		Morten Urdahl (MoU)
		Tommy Gagnes (ToG)
		Rolf Rasmussen (RRa)
		Vidar S Andersen (VSA)
		Geir Enemo (GEn)
		Jan Erik Torp (JET)
		Stig Lødøen (SEL)
		Anne Lise Bjørnstad (ALB)
		Robert H Macdonald (RHM)
		Tor Gjertsen (TGj)
		Anders Eggen (AnE)