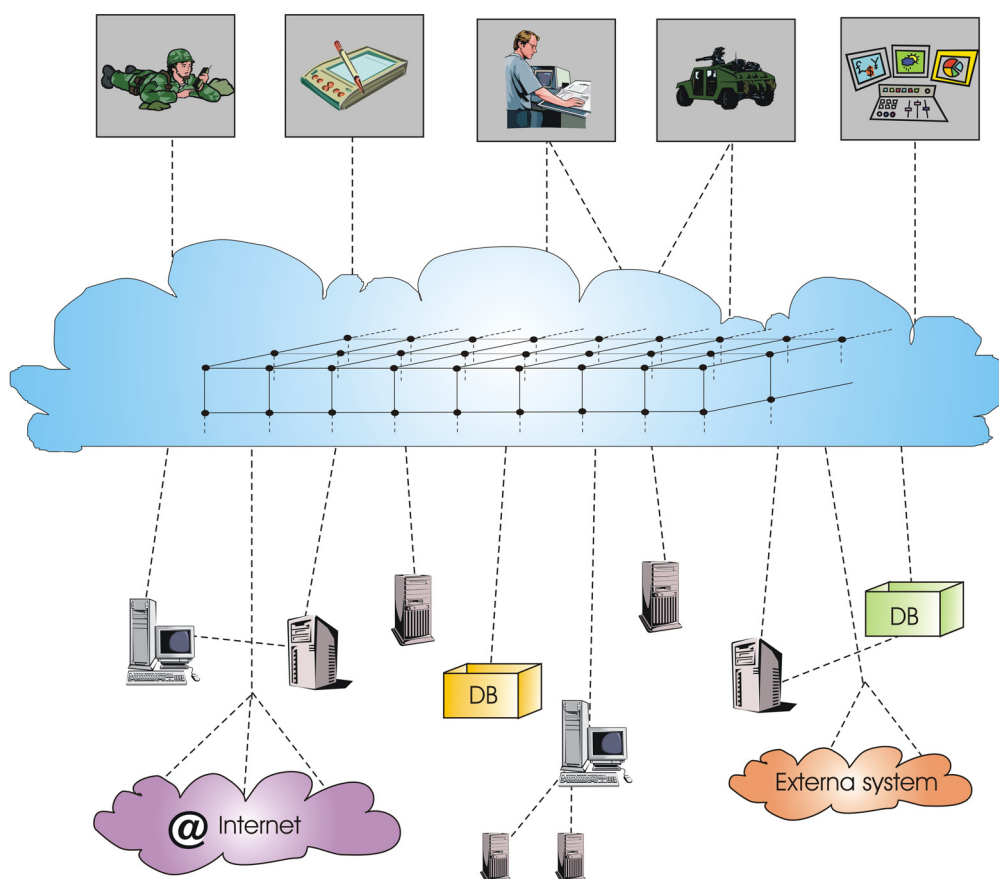


Marianela García Lozano, Jenny Ulriksson

Att koppla samman logistiksystem och användare i en heterogen försvarsmiljö - En förstudie



Marianela García Lozano, Jenny Ulriksson

Att koppla samman logistiksystem och användare i en heterogen försvarsmiljö - En förstudie

Utgivare Totalförsvarets Forskningsinstitut – FOI Systemteknik 172 90 Stockholm	Rapportnummer, ISRN FOI-R--1294--SE	Klassificering Metodrapport
	Forskningsområde 2. Operationsanalys, modellering och simulering	
	Månad, år Augusti 2004	Projektnummer E6925
	Verksamhetsgren 5. Uppdragsfinansierad verksamhet	
	Delområde 23 Logistik	
Författare/redaktör Marianela García Lozano Jenny Ulriksson	Projektledare Marianela García Lozano	
	Godkänd av Monica Dahlén	
	Uppdragsgivare/kundbeteckning FMV	
	Tekniskt och/eller vetenskapligt ansvarig PL	
Rapportens titel Att koppla samman logistiksystem och användare i en heterogen försvarsmiljö - En förstudie		
Sammanfattning (högst 200 ord) Denna rapport behandlar olika metoder och tekniker som kan användas för att hantera verksamhetsprocesser och informationsflöden samt för att koppla samman heterogena system och användare. Dessa tekniker och metoder skall utgöra ett mellanlager som binder ihop befintliga och framtida logistiksystem inom det nätverksbaserade försvaret. Ramar och krav för dessa tekniker behandlas och särskild vikt läggs på viktiga parametrar såsom säkerhet, robusthet och skalbarhet. Olika och intressanta tekniker och metoder såsom distribuerade tekniker, <i>Web Services</i> o.s.v. behandlas. Fokus i förstudien ligger på agenttekniker där tre olika agentramverk behandlas närmare och fördelar och nackdelar tas upp. Slutligen diskuteras olika förslag på arkitekturer för mellanlagret och konkreta rekommendationer till fortsatt arbete ges.		
Nyckelord Logistik, agentsystem, Web Services, distribuerade system, NBF		
Övriga bibliografiska uppgifter	Språk Svenska	
ISSN 1650-1942	Antal sidor: 75 s.	
Distribution enligt missiv	Pris: Enligt prislista	

Issuing organization FOI – Swedish Defence Research Agency Systems Technology SE-172 90 Stockholm	Report number, ISRN FOI-R--1294—SE	Report type Methodology report
	Programme Areas 2. Operational Research, Modelling and Simulation	
	Month year August 2004	Project no. E6925
	General Research Areas 5. Commissioned Research	
	Subcategories 23 Logistics	
Author/s (editor/s) Marianela García Lozano Jenny Ulriksson	Project manager Marianela García Lozano	
	Approved by Monica Dahlén	
	Sponsoring agency FMV	
	Scientifically and technically responsible PM	
Report title (In translation) Connecting logistics systems and users in a heterogeneous defence environment – A preliminary study		
Abstract (not more than 200 words) This report investigates various methods and techniques that can be used to manage processes and information flows, and to connect heterogeneous systems and users. These techniques and methods will constitute a middle layer that ties existing and future logistics systems within the network based defence. Limits and requirements for these techniques are considered and particular attention is laid upon important parameters such as security, robustness and scalability. Various and interesting methods such as distributed techniques, <i>Web Services</i>, etc. are investigated. Focus for the preliminary study is on agents. Three agent frameworks are more closely studied, and advantages and disadvantages with each one are discussed. In the end different propositions for the middle layer architecture are discussed and concrete recommendations on further work are suggested.		
Keywords Logistics, agent systems, Web Services, distributed systems, NBD		
Further bibliographic information	Language Swedish	
ISSN 1650-1942	Pages 75 p.	
Price acc. to pricelist		

Sammanfattning

Enligt uppdrag från FMV har en förstudie genomförts där tekniker och metoder som kan användas för att hantera logistikrelaterade verksamhetsprocesser och informationsflöden samt för ihopkoppling av befintliga och framtida logistiksystem har undersökts. Teknikerna skall utgöra ett mellanlager som binder ihop logistiksystem och olika typer av plattformar och användare och som en naturlig del av förstudien presenteras förslag på utformning av mellanlagret. Det för användarna transparenta mellanlagret skall möjliggöra sömlös hantering och fusion av information i nätverket. Mellanlagret är tänkt att hantera informationsflöden från många olika försvarssystem för logistik, samt i flertalet olika situationer och för varierande användare.

Ramar och Krav

Med förutsättningen att lagret ska tjänstgöra i den heterogena och dynamiska miljö som dagens och framtida logistikrelaterade försvarssystem utgör och kommer att utgöra utarbetades först ett antal krav och ramar som kunde användas för utvärdering av tekniker, metoder och arkitekturförslag. Kraven och ramarna för mellanlagret utarbetades och identifierades genom framförallt tre tillvägagångssätt; genom att skapa scenarion, genom att undersöka existerande logistiksystem inom försvarsmakten, samt genom att identifiera de krav som ett mellanlager av denna karaktär generellt behöver möta. En hel del av kraven är gemensamma för hela NBF. Det var även särskilt viktigt att ta hänsyn till och undersöka ett antal parametrar såsom säkerhet, robusthet, tillgänglighet samt skalbarhet.

De aspekter på säkerhet med störst bäring på idén med att införa ett mellanlager på det sätt som är fokus för studien var de förknippade med datorkommunikation och information. Robusthet togs upp som ett mått på systemkomponenters förmåga att bibehålla sin funktionalitet trots att systemet blir utsatt för någon typ av fel. Oavsett dynamik och fel i systemet förväntas ett robust system bibehålla sina tjänster. Skalbarhet är förmågan hos en datorapplikation, datorsystem eller nätverk att fungera normalt, när det förändras i storlek eller volym för att möta användares behov. Man kan även se skalbarhet som ett mått på att utnyttja den ”nyskalade situationen” till fullo.

Metoder och tekniker

I ett försvar som det svenska som går mot att bli nätverksbaserat (NBF) blir betydelsen av nätverkstekniker och –lösningar allt större. Dessutom tenderar mjukvara att bli alltmer komplicerad och resurskrävande och nya vägar söks för att möta problemen. Distribuerade datorsystem och relaterade tekniker var därför i sammanhanget ett viktigt område att titta på.

Distribuerad teknik är det mest lämpliga alternativet för det nätverksbaserade logistiksystem som var fokus för denna förstudie. P2P och *Web Services* (WS) är intressanta och lovande koncept som kan användas, speciellt WS som är standardiserat och följer tjänstestrukturen. *Grids* är också ett lovande koncept men då det ännu är omoget är det för närvarande något som går att vänta med. Antagligen kommer det framöver att kunna integreras och bli en naturlig del av ett framtida logistiksystem. Agenter är en teknik som har använts länge och det är mycket intressant för syftet.

Agenter

Inom logistikområdet ställs särskilt höga krav på den teknik som stödjer informationshanteringen. Detta på grund av de stora mängder information som behöver hanteras (insamlas, sökas, fusioneras och presenteras). Därför behövs en teknik som kan hantera massiva informationsflöden, samtidigt som den är skalbar och inte degraderas i takt med att det nätverksbaserade försvaret introduceras. Agenter är en teknik som kan användas för att lösa dessa problem inom logistik.

En agent i datorsammanhang är mjukvara som agerar och reagerar autonomt till fördel för en klient. Den interagerar aktivt med den omkringliggande datormiljön som den arbetar i, eventuellt i samarbete med andra agenter, för att utföra uppdrag eller uppnå mål för ett visst syfte. Agent-system består ofta inte av en enda agenttyp, utan av flera agenter som har olika funktionalitet och egenskaper och som utnyttjar olika "strategier" för att uppnå mål. Eftersom agenter ofta består av mobil mjukvara som kan förflytta sig autonomt över nätverk, finns ett flertal olika aspekter av säkerhet som måste beaktas till skillnad från med andra distribuerade system. För att åtgärda problemen finns ett antal väl etablerade metoder.

Först gjordes en genomgång av ett flertal agentramverk och ur dessa valdes sedan tre intressanta kandidater ut för fortsatt arbete och utvärdering. DIET Agents, JADE och Cougaar var de tre ramverk som verkade ha störst potential för uppgiften. En analytisk jämförelse mellan de tre genomfördes och som grund för jämförelsen låg dels ett antal parametrar som avhandlades i respektive systemgenomgång samt dels de krav som tidigare identifierats.

Arkitekturförslag

För att slutligen utforma konkreta förslag till en arkitektur för mellanlagret och analysera den funktionalitet och de krav som arkitekturen måste svara mot, sattes ett enkelt användarfall upp, som baserades på det första scenariot som utarbetades, d.v.s. internationell operation. Mellanlagret är tänkt att följa tjänstekonceptet, d.v.s. de system och tillgångar som finns inom nätverket och som är tillgängliga för logistikplanering och andra logistiksyften är utformade som tjänster. Detta gör att det på ett effektivt och modulärt sätt går att lägga till och dra ifrån nya tjänster till systemet utan att infrastrukturen ändras.

Baserat på dagens trender och tekniker som ansågs lämpliga och som svarade mot kraven valdes fem möjliga lösningar för arkitekturen för mellanlagret ut. Den första möjliga lösningen var att göra en helt egen implementation av mellanlagret. Nästa förslag var att helt basera det på WS. Med hjälp av WS kan logistiksystem på ett flexibelt och återanvändbart sätt göras åtkomliga via webben. Det tredje alternativet var att implementera hela mellanlagret med hjälp av agenter. Det var en intressant lösning med tanke på den komponentbaserade och flexibla infrastrukturtekniken erbjuder. Det fjärde förslaget var att kombinera agenter och WS då dessa tekniker kompletterar varandra mycket bra. Det sista förslaget baseras på Mobila *Web Services*, som är ett koncept som till viss del fortfarande ligger på forskningsstadiet. Det går i stort ut på att dra tjänstekonceptet till sin spets.

Författarnas rekommendationer till arkitektur baseras på en kombination av agenter och *Web Services*. Det agentramverk som framförallt rekommenderas är Cougaar, då det är ett stort och komplext agentramverk med mycket potential, som är utvecklat för militär logistikplanering i USA. Information och kommunikation i mellanlagret kan t.ex. formateras m.h.a. XML. XML används bl.a. för informationsmodellen som utgör gränssnitt mot logistiksystemen. Den ontologi som utnyttjas för modellen kan på detta sätt bytas ut i framtida utveckling utan att infrastrukturen påverkas. Dessutom kan fler modeller läggas till och vidareutvecklas utan att delsystemen behöver modifieras.

Fortsättning

En naturlig fortsättning på förstudien är att praktiskt implementera en prototyp av den arkitektur som har föreslagits. Prototypen kan demonstrera dels den eftersökta funktionaliteten och dels en komplett processkedja. Resultatet utvärderas lämpligen mot de ramar och krav som ställs upp.

Innehållsförteckning

SAMMANFATTNING	5
INNEHÅLLSFÖRTECKNING	7
AKRONYMER	9
LÄSANVISNINGAR	11
1 INTRODUCTION	13
1.1 UPPDRAGSFORMULERING	13
1.2 BAKGRUND	13
1.3 TILLVÄGAGÅNGSSÄTT	15
2 RAMAR OCH KRAV	17
2.1 SCENARIOS	17
2.1.1 Scenario 1 – Internationell operation	17
2.1.2 Scenario 2 – Krissituation	18
2.1.3 Scenario 3 – Fredstid	18
2.2 KRAV FÖR MELLANLAGRET	18
2.3 VIKTIGA PARAMETRAR	21
2.3.1 Säkerhet	21
2.3.2 Robusthet	23
2.3.3 Skalbarhet	24
2.3.4 Data, information och kunskap	25
3 TEKNIKER OCH METODER	27
3.1 DISTRIBUTUERADE DATORSYSTEM	27
3.2 PEER-TO-PEER	27
3.3 AGENTSYSTEM	28
3.4 WEB SERVICES	29
3.4.1 XML	29
3.4.2 Konceptet	29
3.4.3 Arkitekturen	30
3.5 GRIDS	31
3.6 INFORMATIONSMODELLERING	31
3.7 SLUTSATSER	32
4 AGENTSYSTEM	33
4.1 AGENTER I DISTRIBUTUERADE SYSTEM	33
4.2 MOBILA AGENTER	33
4.3 AGENTTYPER	34
4.3.1 Agenter och säkerhet	34
4.3.2 FIPA	35
4.4 AGENTER I LOGISTIK	36
4.4.1 Extra höga krav på tekniken	36
4.4.2 Mobila agenter för logistik	36
4.4.3 UltraLog	36
4.4.4 Praktiskt exempel – Seaway	37
4.5 GENOMGÅNG AV POTENTIELLA AGENTRAMVERK	38
4.5.1 DIET Agents	38
4.5.2 JADE	41
4.5.3 Cougaar	43
4.5.4 Jämförelse mellan systemen – Diskussion och slutsatser	48
5 DISKUSSION OCH FÖRSLAG TILL ARKITEKTUR	51
5.1 ANVÄNDARFALL	51
5.2 ÖVERGRIPANDE – MODULÄR TJÄNSTEARKITEKTUR MED GENERISK INFORMATIONSMODELL	52
5.2.1 Logik – Vad, Var och Hur?	53
5.3 MÖJLIGA LÖSNINGAR	54

5.3.1	<i>Egenutveckling</i>	54
5.3.2	<i>Web Services</i>	54
5.3.3	<i>Agenter</i>	55
5.3.4	<i>Kombination av Web Services och Agenter</i>	55
5.3.5	<i>Mobila Web Services</i>	55
5.3.6	<i>Sammanfattning och utvärdering</i>	56
5.4	ARKITEKTURFÖRSLAG	56
5.4.1	<i>Agenters funktionalitet i mellanlagret</i>	58
5.4.2	<i>Webbtjänster i mellanlagret</i>	58
5.4.3	<i>Integrering av Web Services och agenter</i>	58
5.4.4	<i>Krav för systemet</i>	59
5.4.5	<i>Säkerhetspolicy</i>	60
6	FRAMTIDA ARBETE	61
6.1	UTVECKLING AV EN PROTOTYP FÖR MELLANLAGRET	61
6.1.1	<i>Exempel på utökningar</i>	62
6.2	EGENUTVECKLING AV AGENTRAMVERK	63
	REFERENSER	65
	APPENDIX A –LOGISTIK	69
	APPENDIX B –AGENTSYSYSTEM	71

Akronymer

ACL	Agent Communication Language
AgDUS	Arbetsgrupp Drift och Underhållssystem
AI	Artificiell Intelligens
ALP	Advanced Logistics Project
ALPINE	Advanced Logistics Program Integration and Engineering
API	Application Programming Interface
ARC	Application Reusable Components
Cougaar	Cognitive Agent Architecture
DAML	DARPA Agent Markup Language
DARPA	Defense Advanced Research Projects Agency
DIDAS	DriftDataSystem
DIET	Decentralized Information Ecosystem Technologies
DoD	Department of Defence
FM	Försvarsmakten
FIPA	Foundation for Intelligent Physical Agents
GUI	Graphical User Interface
HTML	Hyper Text Markup Language
JADE	Java Agent Development Framework
JESS	Java System Expert Shell
JVM	Java Virtuellt Maskin
LIFT	Lednings och Informationssystem för Förnödenhetsförsörjning samt Teknisk tjänst
NATO	North Atlantic Treaty Organization
NBF	Nätverksbaserat Försvar
OWL	Web Ontology Language
P2P	Peer-to-Peer
RDF	Resource Description Framework
RETSINA	Reusable Environment for Task Structured Intelligent Network Agents
RTE	Runtime Environment
XML	Extensible Markup Language
UDDI	Universal Description, Discovery and Integration
VSHMOD	Verksamhetsmodellering
W3C	World Wide Web Consortium

WS	Web Services
WSDL	Web Services Description Language

Läsanvisningar

Kapitel 1 är en introduktion till förstudien som är fokus för denna rapport. Uppdragsformulering, bakgrund och tillvägagångssätt presenteras här.

Kapitel 2 tar upp hur ramar och krav för hur ett mellanlager, som är fokus för studien, ser ut. Några scenarion utformades för att få en uppfattning om bredden hos (framtida) logistiksystem samt för att sätta krav på mellanlagret. Dessutom beskrivs viktiga parametrar såsom säkerhet, skalbarhet o.s.v. mer utförligt.

I det tredje kapitlet går ett antal tekniker och metoder igenom. De har valts ut då författarna tror att dessa är av intresse, lämpliga eller nödvändiga för det som vill åstadkommas. Dessa inkluderar bl.a. distribuerade system, *Web Services*, Agenter och XML. För läsaren som är insatt i området distribuerade system och relaterade tekniker, rekommenderas att endast läsa översiktligt och sedan gå direkt till Kapitel 4.

Den teknik som var central för förstudien, Agentsystem, fick ett eget kapitel, Kapitel 4. Här går agenter igenom mer på djupet, samt potentialen för agenter i Logistik. Olika logistikrelaterade projekt som har använt agenter tas också upp. Några särskilt intressanta agentsystem som vi har identifierat behandlas här mer utförligt.

I Kapitel 5 utformas och diskuteras olika arkitekturförslag för mellanlagret. De är baserade på ett användarfall, de ramar och krav som fastställs i Kapitel 2 samt de tekniker och metoder som behandlas i Kapitel 3 och 4. Slutsatser dras sedan från dessa olika förslag och en rekommendation på arkitektur presenteras i det näst sista avsnittet. I detta kapitel sammanfattas slutsatser och rekommendationer från hela det arbete som har utförts för denna förstudie.

I Kapitel 6 ges förslag och rekommendationer till fortsatt arbete.

Appendix A – Logistik – är en kort orientering i området logistik samt några av dagens logistiksystem inom försvarsmakten. Detta appendix är bra att börja med om läsaren saknar insyn i vad militär logistik är.

I Appendix B – Agentsystem – står kortfattat om ett flertal av de agentsystem som författarna har tittat på inför urvalet av de tre mest intressanta som vi sedan fördjupade oss i. Detta appendix kan vara bra för läsaren som önskar orientera sig i vilka slags agentsystem som finns på marknaden, samt få motivering till varför respektive agentsystem ansågs passa syftet eller ej.

1 Introduktion

1.1 Uppdragsformulering

Under de senaste åren har två stora projekt på FMV, VSHMOD-UH-2010 och AgDUS, kartlagt logistikrelaterade processer och system inom försvarsmakten. De har dels identifierat de processer och det informationsbehov som är förknippat med verksamheten inom olika logistikområden och dels de befintliga logistiksystem som finns inom försvarsmakten, FM, för drift- och underhållsverksamheten. Ett annat projekt på FMV har initierats med målet att samordna sådana informationssystem och presentera informationen på ett användaranpassat sätt. Dessa arbeten och andra närbesläktade verksamheter och trender i omvärlden, har påvisat ett stort behov av att identifiera och undersöka lämpliga tekniker och metoder för ihopkoppling av befintliga och framtida logistiksystem samt även med olika typer av användare.

En stor potential ses i att sammanföra och integrera logistiksystem inom försvaret i ett enda stort nätverk. Den gemensamma lägesbilden förbättras bland annat genom att fler informationskällor kan nyttjas samtidigt och tillsammans. Beroende på i vilken nivå (operativ, taktisk, stridsteknisk, teknisk), samt till vilket syfte logistiksystemen är tänkta att fungera och erbjuda stöd för, finns det många olika aspekter att ta hänsyn till vid en sådan integrering. Till exempel är det skillnad på förutsättningarna och kraven som ställs på logistiksystem som används för materielunderhåll och felrapportering, samt system som är anpassade för planering i dynamiska situationer där användare, system och resurser kan tillkomma och försvinna på oförutsägbara sätt. Det ställs därför i många fall särskilt höga krav på de tekniker som ska hantera informationsflödet i och mellan systemen samt användare. Det behövs tekniker som exempelvis kan hantera massiva informationsflöden, plötsligt ändrade villkor och komplexa situationer, samtidigt som de ska vara skalbara och inte degraderas i takt med att t.ex. det nätverksbaserade försvaret introduceras.

Uppdraget från LOG Utv på FMV är att undersöka tekniker och metoder som kan användas för att hantera verksamhetsprocesser och informationsflöden samt ihopkoppling av befintliga och framtida logistiksystem. Exempel på sådana tekniker och metoder är agentsystem, Peer-to-Peer, och *Web Services*. Vid en sådan utredning finns det ett antal parametrar såsom säkerhet, robusthet, tillgänglighet samt skalbarhet som är särskilt viktiga att ta hänsyn till på grund av den heterogena naturen hos logistiksystemen. Teknikerna skall utgöra ett mellanlager som binder ihop logistiksystem och olika typer av plattformar och användare. Det för användarna transparenta mellanlagret skall möjliggöra sömlös hantering och fusion av information i nätverket.

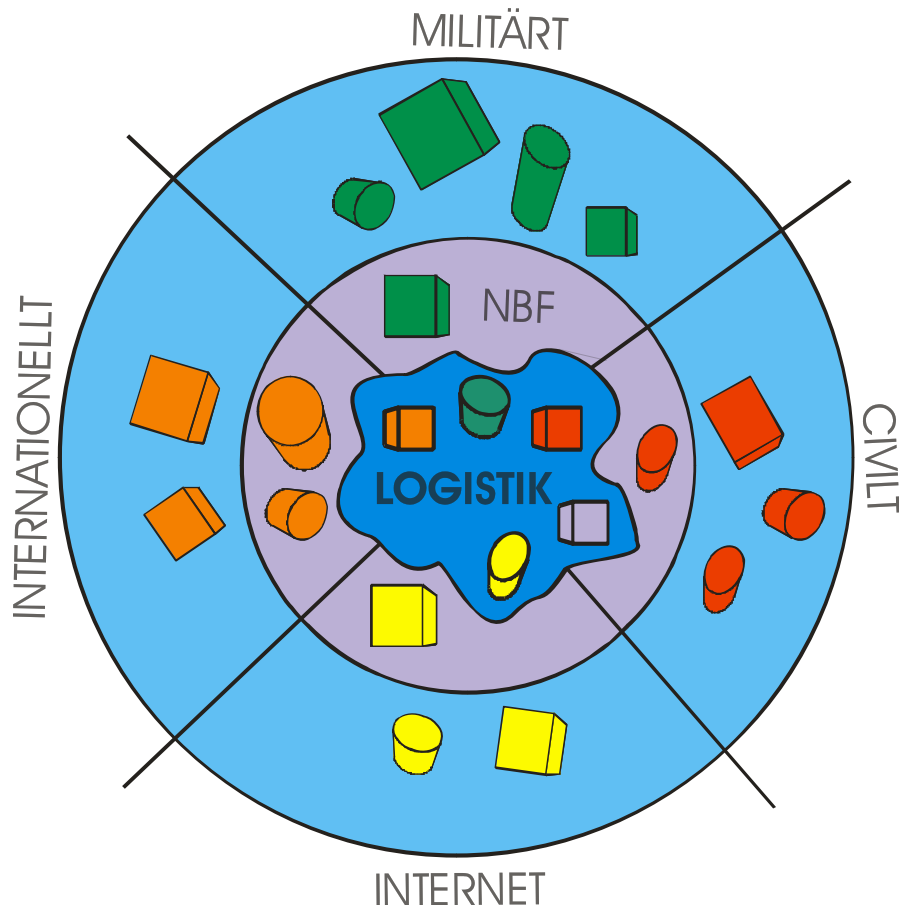
1.2 Bakgrund

Som en följd av Sveriges förändrade säkerhetspolitiska läge påbörjades 1999 en förändring av försvarsmakten, från ett invasionsförsvar till ett mer dynamiskt och flexibelt insatsförsvar. Detta innebär ett förändrat arbetssätt då olika militära domäner och funktioner t.ex. mark, luft och sjö, beslutsfattare, informationssystem och vapensystem, ska kunna länkas samman på ett mer flexibelt sätt än tidigare. Målet är att de skall länkas ihop i en nätverksorganisation och bilda det flexibla insatsförsvaret, NBF. För mer information om NBF, se [NBF].

Ett framtida logistiksystem¹ inom ramen för NBF har många krav på sig för att kunna utgöra det stöd som behövs och efterfrågas. FMV har under de senaste åren påbörjat arbetet med övergången från dagens domänspecifika, ofta monolitiska och statiska logistiksystem, till morgondagens flexibla och dynamiska. Visionen är att, genom ett för användaren transparent lager,

¹ För den läsare som är obekant med logistikdomänen, vad logistik är eller hur dagens logistiksystem ser ut, rekommenderas att läsa Appendix A –Logistik, först.

information och tjänster ska kunna nås och utnyttjas från ett antal heterogena och distribuerade källor. Istället för att användaren behöver söka, känna till och använda ett specifikt system kan en klient istället tillhandahålla ett utbud av tjänster och funktionalitet. Dessa tjänster kan finnas inom FM:s nätverk eller utanför i form av kommersiella system eller till och med på Internet, se Figur 1-1 nedan.



Figur 1-1: Logistik inom FM ska för NBF kunna tillhandahålla tjänster från flera olika domäner

För att kunna uppnå visionen med det nätverksbaserade försvaret och även de framtida nätverksbaserade logistiksystemen finns det många utmaningar att hantera. Det finns t.ex. utmaningar i strukturering av information, att lyckas enas kring gemensamma begrepp, hur ökade informationsflöden från många olika heterogena källor skall kunna hanteras samt hur säkerhetsaspekter, skalbarhet, robusthet och tillförlitlighet hos systemen skall kunna säkerställas. Gamla och nya tekniker och metoder, eller kanske nya kombinationer av dessa, behövs för att hantera dessa utmaningar.

I ett par projekt inom logistikdomänen på FMV har man i ett första steg kartlagt processer, informationsbehov och befintliga system inom framförallt drift och understöd. Projekten VSHMOD och AgDUS har lagt en del av grunden för att utforma morgondagens logistiksystem.

VSHMOD-UH-2010 –Verksamhetsmodellering I fyra faser har projektet inriktat sig på att modellera de processer och informationsbehov som är förknippade med materielunderhåll, förnödenhetsförsörjning, drift och telekommunikationsnätverk samt tekniskt systemstöd för FM. Varje fas har resulterat i att bl.a. ett processdiagram med organisatoriskt- och systemberoende processer och deras ingående aktiviteter har framställts. Även det informationsbehov som finns för varje process har identifierats. Målet har varit att ta fram beständiga processer och verksamheter för underhållstjänster med fokus på NBF och målbild 2010. Från det resultat som

erhölls efter varje fas sammanställdes slutligen en gemensam och generisk informationsmodell med tillhörande begreppsdefinitioner. För mer information om projektet se [VSHMOD]. Fortsättningsprojektet LOGLED NBF modellerar de processer och informationsbehov som är förknippade med logistikledningen av underhållstjänsterna.

AgDUS –Arbetsgrupp Drift och underhållssystem är ett projekt som har inriktat sig på att kartlägga de system som stöder drift- och underhållsverksamheten inom FM och se hur väl de stämmer överens med morgondagens krav. Dessa krav kommer delvis från det arbete som har genomförts inom VSHMOD-projektet. Målet har varit att med hjälp av teknisk analys sammanställa respektive systems användningsmöjligheter, undersöka vilka som kan tas bort och vilka funktioner som saknas. Syftet är att få fram en helhetsbild av alla system och se var överlappningar och brister finns. De största systemen som har kartlagts inom projektet är DELTA, LIFT, DIDAS och FREJ. För mer information om projektet se [AgDUS]. För kortfattad information om de nämnda logistiksystemen se Appendix A - Logistik. Ett nytt projekt, PRIO, som startas på Högkvarteret kommer att genomföra en liknande kartläggning som AgDUS men inrikta sig på personal, materiel och ekonomi.

Genom arbetet inom dessa projekt och andra relaterade sådana har man insett att det saknas ett mellanlager som kan användas för att koppla ihop befintliga och framtida logistiksystem med varandra, andra sorters system samt olika användare med olika krav på stöd och information. Exempel på krav på ett sådant mellanlager är att systemen på ett flexibelt sätt kan interagera med varandra och med användare, samt på lämpligt sätt stödja informationshantering. Informationen i ett sådant system skall även gärna vara användaranpassad.

Författarnas bakgrund är forskning inom modellering och simulering, M&S, ett område som är besläktat med frågeställningen som har utgjort kärnan i uppdraget. Inom M&S-världen började heterogena system att kopplas samman redan under 80-talet för att exekvera stora distribuerade simuleringar. Interoperabilitet och återanvändbarhet har hela tiden varit två viktiga forskningsområden i detta arbete, och på senare tid har det inneburit att nätverksbaserade arkitekturer, infrastrukturer och informationsmodellering har fått allt större fokus.

1.3 Tillvägagångssätt

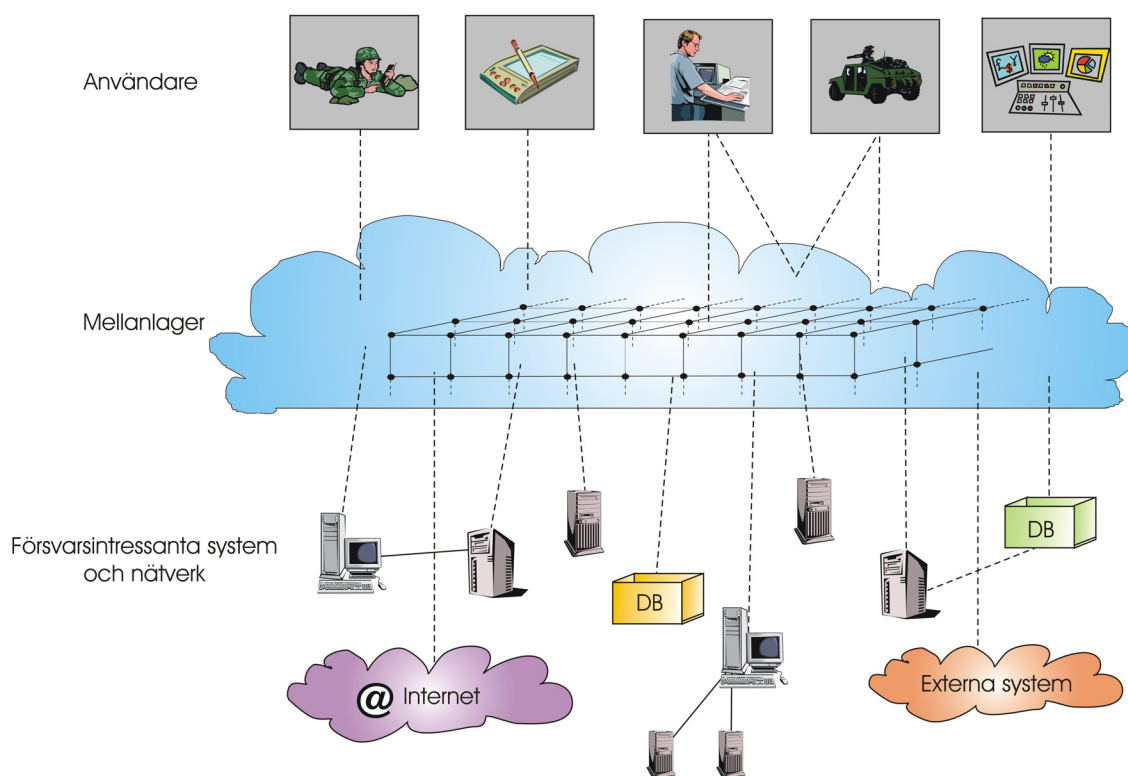
För att tillgodose behovet av att knyta samman heterogena logistiksystem med olika typer av användare genomförs en förstudie i hur ett mellanlager för detta skulle kunna utformas. Mellanlagret ska möjliggöra sömlös och transparent sammankoppling mellan användarprofiler och system som kan användas för försvarssyften, såsom logistiksystem, civila system och Internet etc, se Figur 1-2. Huvudmålet med detta projekt är därför att utforma ett förslag till en arkitektur för ett sådant mellanlager, samt skapa underlag och rekommendationer för hur ett sådant skulle kunna realiseras.

Vi började med att undersöka området militär logistik, sätta oss in i bakgrunden till uppgiften, samt orientera oss i hur dagens existerande logistiksystem inom FM ser ut. Vi gick sedan vidare till kraven för hur ett mellanlager såsom beskrivet ovan skulle se ut. Vi skapade några enkla scenarion och använde bl.a. dessa som grund för att identifiera de krav och förutsättningar som ställs på mellanlagret. Olika parametrar som var av speciell vikt undersöktes närmre.

Olika tekniker och metoder som var av intresse för det aktuella problemet identifierades och undersöktes. I fokus för studien låg agentsystem och en eftersökning i existerande tekniker för detta genomfördes. Dessutom undersöktes särskilt hur agenter och logistik skulle kunna sammanföras och vilka för- resp. nackdelar detta skulle medföra.

Den tredje fasen i arbetet var att undersöka hur en arkitektur för mellanlagret skulle kunna se ut. Ett användarfall, baserat på ett av de ursprungliga scenarion som gjordes, användes för att analysera hur mellanlagret skulle svara mot situationen utifrån ett antal olika teknik-

kombinationer. Baserat på detta drog vi slutsatser, tog fram rekommendationer och förslag till fortsatt arbete.



Figur 1-2: Sömlös och transparent koppling mellan användare och system i det framtida insatsförsvaret. I figuren visas ett antal användarprofiler samt olika typer av system. Systemen kan vara databaser (markerade DB) eller andra system, applikationer eller nätverk

2 Ramar och krav

Det mellanlager som avses i denna rapport är tänkt att hantera informationsflöden från många olika försvarssystem för logistik, samt i flertalet olika situationer och för varierande användare. För att sätta ramarna för ett sådant lager, och identifiera kravställningar och förutsättningar för systemet, var därför ett naturligt led i utvecklingen att ställa upp ett antal relaterade scenarion. Detta kapitel presenterar därför tre scenarion, de krav, ramar och förutsättningar vi har identifierat samt parametrar som är av särskild vikt för detta arbete.

2.1 Scenarios

Följande scenarion är avsedda att spegla några vanliga situationer inom försvaret där logistiksystemen används och där funktionaliteten hos mellanlagret ställs på sin spets. Detta för att sätta så rimliga och verklighetstroga krav på mellanlagret som möjligt. De har utvecklats separat men med de scenarion i åtanke som framlades i [LOGLED], från vilken paralleller har dragits och formuleringar har lånats. Vidare arbete som skulle behövas för en framtida implementering är att undersöka vilka processer¹ som hanteras inom varje scenario.

2.1.1 Scenario 1 – Internationell operation

Hastigt uppkommen situation

Plötslig informell förfrågan från regeringen om huruvida FM kan verka internationellt i specifikt syfte. En beredning genomförs om huruvida insatsen är logistiskt möjlig och hur. Ett förband skapas efter givna förutsättningar och ska snabbt förflyttas till annat land med behövliga förnödenheter och materiel. Rekommendationer och logistisk planering genomförs i hemlandet men eventuellt även med *input* från personal som redan finns på plats. En del av förnödenheter och materiel kan skickas omedelbart, men annan kan behöva införskaffas senare p.g.a. kostnader, tidsaspekter, brister etc.

Frågor som behöver hanteras är t.ex.: Hur ska förnödenheterna införskaffas? Finns materiel och förnödenheter tillgängliga eller kan de införskaffas på plats? Eventuell tillverkningstid? Vad är lämpligaste transport; båt, landfordon eller flyg? Måste förnödenheter och materiel finnas på plats när förbanden kommer dit eller kan de skickas ner allt eftersom? Vissa förnödenheter kanske kan ersättas med andra likvärdiga medan andra inte går att ersätta, o.s.v.

Logistikledningen besvarar frågorna och skapar underlag för förnödenhetsförsörjning, underhållsupplägg, införskaffning av resurser, omfördelningar, modifieringar etc. Arbetet görs mha. sökning och inhämtning av information från olika källor där parametrar som beaktas är: lagerstatus, införskaffningstid, pris etc. Informationen fusioneras och en optimerad plan för införskaffning, underhåll och transporter skapas. Eftersom arbetet inkluderar många olika parter är det viktigt att även dessa kan delta i arbetet.

Pågående internationell insats

Efter att en operation har pågått ett tag blir nytillkomna parametrar intressanta för förnödenhetsförsörjning och materielunderhåll osv. Möjligheter kanske finns att samarbeta med andra nationer i t.ex. FN-uppdrag för förnödenhetsförsörjning. Viss materiel kanske dock fortfarande är nationsspecifik och måste införskaffas från hemlandet. Utrustning, materiel, m.m. kanske har börjat gå sönder och reparationer måste utföras.

Logistikstöd från Sverige understödjer med förnödenhetsförsörjning, ransonering, planering etc. samt andra lösningar till uppkomna problemsituationer. Problem och planering beräknas med

¹ Med processer syftas här till de processer som identifierades inom ramen för VSHMOD-projekten.

stöd av personal på plats, samt eventuellt i samarbete med andra deltagande nationer etc. Olika lösningar för att åtgärda problemen kan vara att hitta ersättningsdelar och förnödenheter på plats eller få till stånd ett utbyte med andra nationer. Andra nationer kanske i sin tur behöver hjälp med materiel och förnödenheter.

Avslut internationell operation

När en operation avslutas, avvecklas förband inklusive materiel, infrastruktur, *camp* mm. Materiel hanteras på plats (förstörs etc.), överlämnas till annan nation, säljs eller återfraktas till hemlandet. Statusbedömning och inventering görs på plats och ekonomisk bedömning om återtransport utförs. I hemlandet mottas och hanteras återfraktad materiel. Förrådsställning, återförande i system, samt fakturering hanteras. Efteranalys görs parallellt för att analysera åtgång, bortfall etc.

2.1.2 Scenario 2 – Krissituation

En långtradare med en last av högexplosivt bränsle välter strax utanför Stockholm. Långtradaren är av utländsk nationalitet och dålig information finns att tillgå kring bränsletypen. Samtidigt har ett massivt *hacker*-intrång i försvarsmaktens system slagit ut ett delsystem till ett av de stora understödssystemen för logistik. Man beslutar att använda och samordna flera resurser såsom FM och även civila myndigheter (brandkår, polis etc.). Operationen blir omfattande och det krävs direkta räddningsinsatser för omedelbar evakuering av människor i närheten m.m. Logistikunderstöd är nödvändigt för transporter, förnödenheter, sjukvårdsmateriel etc. För att återställa och ersätta den förlorade och skadade informationen i det kraschade delsystemet (för understöd logistik) behöver omfattande informationsinsamling genomföras. Detta för att bestämma gasspridningsförloppet, optimala transportvägar, eventuell påverkan på området, samt för andra logistiska beräkningar och logistikunderlag. Information och stöd behövs från ett stort antal system och myndigheter. På grund av informationsbortfallet som föregås av systemintrånget måste information för planering och beräkningar nu även sökas och inhämtas på annan plats än inom försvarsnätverket. Kommersiella och civila nätverk (Internet, olika leverantörer etc.) utnyttjas externt för detta. Även militära sjukvårds- och transportresurser utnyttjas i operationen.

2.1.3 Scenario 3 – Fredstid

Under fredstid behövs förnödenheter och materiel ”*Just in case*”, dvs. i fall att behov plötsligt skulle uppstå. Då behöver materiel och förnödenheter snabbt kunna tillgå. Samtidigt innebär alltför stora lager en kostnad som inte är försvarbar. Om då exempelvis en flottilj har en nedskärning av en viss transporttyp, pga. ekonomiska eller andra omständigheter, behöver ett antal saker beräknas. Transporttypen kanske kan ersättas av annan befintlig transport. För detta behöver den ”nya” typens prestanda vägas mot den gamla och underlag måste skapas för huruvida denna kan ersätta den eller inte under de förhållanden som råder (utnyttjandetid per år t.ex.). Logistikprocesser utnyttjas för att kunna uppnå en optimal balans mellan lagerstatus, beredskapsnivå, kostnad, samt prestanda.

2.2 Krav för mellanlagret

Detta avsnitt utgör en generell kravspecifikation för mellanlagret. Specifikationen ger implicita krav på de tekniker och metoder som krävs för att mellanlagret ska kunna bli verklighet. Med förutsättningen att lagret ska tjänstgöra i den heterogena och dynamiska miljö som dagens och framtida försvarssystem utgör och kommer att utgöra (med särskild inriktning mot logistik i detta syfte) har vi utarbetat ett antal krav och ramar som kan användas för utvärdering av tekniker, metoder och arkitekturförslag.

Kraven för mellanlagret har vi utarbetat genom framförallt tre tillvägagångssätt; det är dels de krav som har framkommit från arbetet med scenariona, dels är det genom att undersöka de krav som existerande logistiksystem inom t.ex. försvarsmakten ställer, och slutligen har de krav som vi anser att ett mellanlager av denna karaktär generellt behöver möta identifierats. En hel del krav är gemensamma för hela NBF.

Krav från scenarion

Från scenariomodelleringen i Avsnitt 2.1 och efteranalysen kunde många olika typer av slutsatser dras och krav identifieras. Vi har försökt begränsa oss till de som har direkt påverkan på mellanlagret och de funktioner som måste kunna stödjas.

Sammanfattningsvis kan sägas att det i varje scenario är många olika försvarsgrenar som är inblandade men det är fortfarande till stor del samma typer av funktioner som måste kunna utföras. I många fall existerar en viss tidsbrist och det är då viktigt att systemen och kommunikationen dem emellan inte är långsam. Det är även viktigt med säkerhet och robusthet. Det finns ett visst behov av att uppkoppling ständigt finns. Informationshantering och fusionering är vitalt, speciellt att information som efterfrågas alltid finns tillgänglig.

I många av scenariona kunde uttydas att det är flera olika logistikfunktioner och nivåer som är inblandade såsom logistikledning, -planering, -uppföljning o.s.v. Det är många olika typer av processer och olika sorters information som är inblandad och som kommer från många spridda system. Här kan det även vara av intresse med informationsfusion för att optimera flödena av data. Det kan vara kritiskt om information är fel. Många gånger är det dynamiska situationer där det som var sant för en tid sedan kanske inte är sant nu. Mobila och tunna klienter behöver kunna stödjas för t.ex. scenario 1.2. I scenario 1 kunde speciellt ses att internationell samverkan måste kunna ske och i scenario 3 att koppling till externa system eller externa aktörer, både nationella och internationella, behöver kunna göras. Säkerheten för systemen och det data som finns i dem är viktig. Uppföljning av data måste kunna göras och tillståndsförändring hos materiel och förnödenheter (från t.ex. utomlands hem) så att de kan komma tillbaka in i systemet. Från scenario 1 och 3 kunde ses att analys och efterbearbetning måste kunna göras och fusionering av uppgifter är nödvändig då information finns på flera platser. I scenario 2 ställs de minst kritiska kraven, men dock är det antagligen här som den största delen ny information kommer in och lagras i systemen. Resurshantering är därför viktigt här.

Logistiksystem

Från att ha fått bilda oss en uppfattning av hur dagens logistiksystem ser ut kunde en del slutsatser dras som påverkar utformningen av mellanlagret. Huvudinriktningen på dagens logistiksystem kan i princip beskrivas med:

- Transport, Drift och Underhåll
- Ekonomi
- Grund- och förvaltningsdata
- Utvärdering
- Anskaffning/inköp
- Modellering och simulering
- Personal
- Lagerstyrning och -hantering
- Konfiguration

Klassningen på informationen som dessa system hanterar kan vara av typen hemlig eller ej hemlig. I framtiden kan det även tänkas tillkomma fler klassningsnivåer då Sverige alltmer anpassar sig till de internationella nivåerna. De olika systemen stödjer vapenslagen luft, mark och sjö, antingen bara en enda eller kombinationer. De flesta systemen verkar vara egenutvecklade och då i COBOL, Java, C eller C++. Antalet användare för varje system varierar från knappt 20 till flera tusen. För det mesta består systemen av databaser såsom Access och Progress. Operativsystemen är av varierande slag t.ex. AIX och Windows. Systemen kan vara distribuerade men många är monolitiska. För det mesta är de distribuerade systemen av typen *client-server*. Det finns olika sätt att komma åt data i de olika systemen dels via nätverk och dels manuellt genom portabla media, typ disketter och CD. Data i systemen är formaterad på olika sätt.

Andra aspekter – sammanställning och komplettering

Erfarenheter från scenarion, logistiksystem, infrastrukturer och annat ledde till egna generella krav som kan ställas på mellanlagret. Vi tittade på det generellt och då i samma sammanhang och situationer som det framtida nätverksbaserade insatsförsvaret kommer att ställas inför. Det finns ett antal viktiga aspekter som mellanlagret och framtida logistikverksamhet kommer att tvingas hantera, där många är gemensamma med de för NBF.

Då dagens, och antagligen även framtida, system och data är distribuerade och delvis decentraliserade (centralisering kan ske delvis eller inte alls, och utesluter inte replikeringar för att uppnå robusthet). Noder och klienter kan vara av varierande karaktär. Mellanlagret ska hantera att den underliggande nätverksarkitekturen inte är homogen, utan kan bestå av olika delnätverk som interagerar internt och externt, samt är av olika karaktär (hanterar olika frågor och information). Mellanlagret ska kunna hantera och anpassa sig dynamiskt efter nodens nuvarande status. Till exempel ska informationsflödet kunna optimeras utefter tillgänglig bandbredd.

Då det är många olika system behöver mellanlagret vara plattformsoberoende. Samverkan och kommunikation ska kunna ske med och mellan, inte bara olika operativsystem, men även olika ramverk och applikationer. Då det ofta är dynamiska situationer måste mellanlagret stödja att noder och klienter tillkommer och försvinner. Det måste därför vara flexibelt och robust så att det inte är beroende av specifika noders bortfall. Det är även bra om dynamisk uppkoppling stöds, det vill säga en förfrågan ska kunna bearbetas av mellanlagret samtidigt som klienten är *offline*. Skalbarhet är även viktigt då mellanlagret ska vara utbyggbart. Ny funktionalitet skall kunna läggas till utan att det påverkar grundfunktionaliteten och systemets prestanda och funktionalitet ska inte påverkas märkbart av att antalet noder och klienter ökar. Säkerhet är naturligtvis också viktigt och det ska finnas på olika nivåer. Informationsbearbetning och -fusionering ska ske på lämplig plats. Det vill säga, det som skall processas (informationen) kan ha specifika krav (prestanda, plattform, säkerhet etc.). Dessutom kan nuvarande situation, till exempel belastningen på en nod, påverka hur lämplig en plats är.

Sammanfattningsvis:

- Dynamisk miljö
- aktörer som kommer och går
- situationer och förutsättningar som förändras dynamiskt
- Mycket information som behöver hanteras (inhämtas, selekteras, fusioneras etc.)
- Många informationskällor
- Många olika typer av källor
- Samarbete mellan olika försvarsgrenar och externa aktörer

2.3 Viktiga parametrar

Det har flera gånger framgått att ett antal parametrar såsom, säkerhet, robusthet och skalbarhet är särskilt viktiga för att uppnå ett mellanlager som uppfyller sin funktion. I detta avsnitt tar vi upp vad som menas med dessa parametrar och vad det innebär för ett mellanlager av den typ som behandlas i denna rapport. Hanteringen av information är också en viktig del av mellanlagret och det utvecklas också i detta avsnitt. Alla dessa parametrar måste på ett eller ett annat sätt hanteras i framtida försvarsnätverk såsom det nya insatsförsvaret avser.

2.3.1 Säkerhet

När datorsäkerhet diskuteras kan det ske på flera nivåer och inom flera kategorier, det kan t.ex. handla om datorkommunikationssäkerhet, säkerhetskontroll och auktorisering av personal och företag, fysisk säkerhet, etc. Sedan kan i sin tur varje kategori delas upp i ett antal undernivåer. Datorrelaterad säkerhet kan t.ex. ligga på olika platser såsom på nätverksnivå eller på applikationsnivå. Det kan även finnas olika mål såsom hårdvarusäkerhet, mjukvarusäkerhet, informationssäkerhet, o.s.v. Här tar vi bara kort upp de aspekter på säkerhet som vi tror har störst bäring på idén med att införa ett mellanlager på det sätt som är fokus för studien. Detta innebär att vi framförallt fokuserar på de aspekter som är förknippade med datorkommunikationssäkerhet¹ och delvis med datorsäkerhet². De andra säkerhetskategorierna är naturligtvis också nödvändiga för att uppnå en acceptabel säkerhetsnivå, men vi tar inte upp dem närmre här. För den som önskar läsa mer om datorkommunikationssäkerhet rekommenderas [CCS].

Det som framförallt kommer att hanteras i mellanlagret är information. Följande mål brukar tas upp när det gäller informationssäkerhet:

- *Confidentiality*: att vara säker på att information inte visas eller röjs för oauktoriserade personer
- *Integrity*: att säkerställa att data är konsistent och framförallt att oauktoriserat skapande, förändring eller borttagande av data ej sker
- *Availability*: att användare som har tillstånd ej felaktigt nekas tillgång till information och resurser
- *Legitimate use*: att resurser inte används på ett icke auktoriserat sätt eller av icke auktoriserade användare

För att kunna uppfylla dessa mål behövs flera saker, att det finns en säkerhetspolicy, att man är medveten om vilka potentiella hot och risker som finns, att de klassiska säkerhetstjänsterna finns och att intrångsdetektion och säkerhetsgranskning finns.

Säkerhetspolicy

En säkerhetspolicy är ett ganska brett begrepp men i princip utgörs den av en mängd regler som appliceras på alla säkerhetsförknippade aktiviteter i en säkerhetsdomän. I vårt fall handlar det mest om regler för auktorisering och för *access control* d.v.s. kontroll av åtkomst.

Auktorisering är en viktig del av säkerhetspolicyen. Det handlar i princip om att bestämma vem som får göra vad och hur. Ett exempel är att information som klassificeras på något sätt såsom

¹ Med datorkommunikationssäkerhet menas skyddandet av information medan det förmedlas från ett system till ett annat.

² Med datorsäkerhet menas skyddandet av information inuti ett datorsystem, d.v.s. det berör underkategorier såsom operationssystemssäkerhet och databassäkerhet.

fri, klassificerad hemlig eller hemlig, endast får visas för användare som har tillstånd att se information på en viss nivå eller högre.

Regler för *access control* avser det automatiska upprätthållandet av auktorisering i datorsystem och nätverk. Detta innebär flera typer av regler, t.ex. identitetsbaserade, roll-baserade och nivå-baserade. Identitetsbaserade regler tillåter eller nekar åtkomst för identifierade personer eller grupper. Roll-baserade regler är relaterade till de identitetsbaserade reglerna, men här är det roller som tilldelas individer och grupper, och auktoriserar baserat på dessa roller. En viss individ kan alltså ha rätt att i olika situationer anta vissa roller och då ha rätt att komma åt information som är anpassad till rollen. I NBF kommer informationsåtkomst att vara rollbaserad [GTI]. De nivå-baserade reglerna är till för att reglera åtkomst för olika klassificeringsnivåer på information och olika auktoriseringsnivåer för personer och grupper. Alla dessa regler går även att applicera på datorsystem.

Potentiella hot och risker

De potentiella hot och risker som finns och som direkt relaterar till de informations säkerhetsmål som togs upp tidigare är; informationsläkage, integritetskränkning, nekande av tjänst s.k. *denial of service* och oauktoriserad användning. De sätt som dessa hot och risker kan inträffa på är framförallt ”penetrering” eller ”plantering”. Exempel är:

- Maskeradning: då en person eller ett system låtsas vara någon annan. Detta är ett av de vanligaste sätten att penetrera säkerhetsskydd. Det är bl.a. så här *hackers* gör
- Kringgå kontroller: då systemfel eller svagheter används för att kringgå säkerhetsskydden
- Auktoriseringsöverträdelse: då en person eller system som har tillstånd att använda system eller resurser på ett sätt använder det på ett annat ej auktoriserat sätt. Detta brukar vara s.k. insider-hot
- Trojansk häst: då mjukvara som innehåller osynliga eller till synes harmlösa komponenter eller funktionalitet äventyrar säkerheten i systemet när de exekveras
- Fallucka: då funktionalitet som byggs in i system eller mjukvara gör att när viss data tillhandahålls så går det att kringgå eller överträda säkerhetspolicys

Det finns även ett antal underkategorier till dessa hot som t.ex. tjuvlyssning, uppsnappning eller förändring av information, analys av nätverkstrafik, nekande av skickandet av information, virus, m.m. som även måste kännas till. För att skydda sig mot alla dessa hot och risker gäller det att dels tillhandahålla skydd för de fysiska systemen, administrativa rutiner, personalkontroller, o.s.v. samt dels ett säkerställa antal säkerhetstjänster.

Säkerhetstjänster

De 5 typer av säkerhetstjänster som behövs för informationssäkerhet brukar delas in i autenticitet, dataintegritet, *access control*, *confidentiality* och *non-repudiation*. Det behöver även finnas sätt att detektera intrång och granska säkerheten.

Autenticitetstjänsterna säkerställer identiteten hos en person eller system när denne eller hävdar en viss identitet. Ett vanligt sätt att uppnå detta är t.ex. att ha lösenord. Ett annat användningsområde är t.ex. när upphovskällan till viss data behöver identifieras. Autenticitetstjänsterna är några av de viktigaste då de andra till viss del är beroende av dessa. Dessa tjänster kan användas för att bekämpa t.ex. maskeradning.

Dataintegritetstjänsterna är till för *integrity*-målen. På samma sätt som för *confidentiality*-tjänsterna är det viktigt att veta vilken granularitet man skall applicera tjänsterna på d.v.s. vilka dataenheter som behöver skyddas. Alla dataintegritetstjänster skyddar mot skapandet eller modifierandet av

data. Borttagande och duplicering av data är däremot svårare att detektera och inte alla klarar av det.

Access control-tjänsterna är till för att motverka hot som berör alla säkerhetsmålen, genom att de skyddar mot icke auktoriserad användning, tillgång, avslöjande eller manipulation av t.ex. resurser såsom datorer, kommunikation eller data.

Confidentiality-tjänsterna skyddar, som tidigare nämnts, mot att information avslöjas eller röjs för icke auktoriserade. Detta kan även inkludera att hindra observation av existensen respektive avsaknaden av data¹, datas storlek, dynamiska variationer hos datas karakteristika såsom innehåll, storlek, frekvens, etc.

Non-repudiation-tjänsterna är till för att skydda mot två saker, efter att meddelanden skickats, att det senare är någon som förnekar att meddelanden verkligen har skickats samt att avsändaren verkligen stämmer. Dessa tjänster förhindrar i sig inte att någon enhet försöker fuska men det de gör är att de tillhandahåller obestridliga bevis som kan användas för att lösa en konflikt. Det svåra är att system inte är perfekta, meddelanden som skickas kan gå förlorade på vägen, de kan förstöras innan mottagaren har hunnit bearbeta dem, de kan försenas o.s.v.

Alla dessa tjänster är en viktig och nödvändig del av ett mellanlager som ska verka i en NBF-miljö, de behövs även för hela NBF. Idag finns det många olika protokoll som tillhandahåller dessa tjänster på olika nivåer och för olika syften. För alla nya tekniker och metoder som introduceras och används måste en hot- och riskanalys göras och undersöka om det finns existerande stöd för tjänsterna inbyggt eller om det finns som tillägg. Det är även viktigt att undersöka om eventuella nya hot kan uppstå och då om detta kräver att nya protokoll måste tas fram.

2.3.2 Robusthet

Robusthet är ett mått på systemkomponenters förmåga att bibehålla sin funktionalitet trots att systemet blir utsatt för någon typ av fel. Robusthet kan även refereras till som ett systems hälsa, styrka och livslängd. Ett robust system förväntas bibehålla sina tjänster oavsett dynamik och fel i systemet [Anderson et al.]. Delkomponenter ska kunna haverera eller även avge felaktig information, utan att resten av systemet betar sig felaktigt eller faktiskt inte fungerar alls.

Händelser som kan påverka robustheten hos systemet utgörs t.ex. av:

- Enskilda komponenter som havererar
- Hot mot fysiska förbindelser (nätverksförbindelser mm.)
- Nätutrustning och annan fysisk utrustning som havererar
- Yttre attacker
- Överbelastning
- Dataintrång

När termen robusthet används som egenskap på ett system kan det även innebära att systemet i sig stödjer ett antal funktioner, såsom automatisk återställning av fel, korta svarstider², automatisk *back-up*, låg felfrekvens mm.

¹ Observera att vi här gör skillnad mellan information och data. För en vidare förklaring av skillnaden se Avsnitt 2.3.4.

² Svarstid är tiden det tar från att en applikation skickar ett meddelande till en annan applikation eller system, till att den får tillbaka svaret (svarsmeddelandet).

Metoder för robusthet

För att förbereda systemet eller nätverket för robusthet finns det en mängd metoder, varav den vanligast omnämnda är redundans. Redundans innebär att det hela tiden finns uppdaterade replikeringar av applikationen eller databasen i nätverket, lämpligast inte på samma nod som originalapplikationen residerar. Att utnyttja protokoll med en hög säkerhetsnivå är ett annat sätt att understödja robusthet. Dessutom är det viktigt att hårdvara skyddas fysiskt, eftersom även denna kan haverera. En mycket viktig faktor i sammanhanget är att säkerheten i systemet generellt hålls hög.

Robust infrastruktur för svenska försvarsnätverk

Robusthet är en mycket viktig fråga i design av nätverk och system. Denna faktor är mycket viktig i ett försvarsnätverk eller ett nätverk för försvarsintressant information, där en kritisk situation kräver att systemen fungerar och information alltid är tillgänglig. I kritiska händelser såsom i den hastigt uppkomna situationen i Scenario 2.1.1 eller i krissituationen i Scenario 2.1.2 är det mycket ödesdigert ifall åtkomst till information plötsligt inte finns eller går för långsamt, eller där delar av systemet plötsligt inte fungerar. Delsystem i försvarets nätverk måste kunna tillkomma, frångå och haverera utan att hela systemet påverkas nämnvärt.

Därför krävs det en mycket robust infrastruktur för ett system och nätverk som försvarsresurser förlitar sig till. I framtida insatsförsvaret kommer systemet dessutom att vara uppkopplat mot ett flertal olika typer av nätverk, såsom säkra nätverk, trådlösa nätverk och Internet etc., vilket höjer komplexiteten. Dessutom behöver den infrastruktur och det informationssystem som utvecklas vara tillräckligt robust för att kunna bibehålla sin funktionalitet och relevans även när tekniker och metoder i systemet förändras över tiden.

2.3.3 Skalbarhet

Skalbarhet är förmågan hos en datorapplikation eller ett datorsystem/nätverk att fungera normalt, när det förändras i storlek eller volym för att möta användares behov. Fungera normalt innebär här att bibehålla prestationsförmåga utan att förlora i kvalitet. Förändringen kan motsvara exempelvis nya resurser (processorer eller hårdvara) som läggs till, eller nya klienter som ansluts till systemet. Exempelvis sägs en databas vara skalbar om den bibehåller samma prestanda oavsett antalet transaktioner och förfrågningar till databasen. Generellt används skalbarhet som mått på systemets förmåga att öka eller minska i prestanda och kostnad som svar mot förändringar. Som parametrar används till exempel, som i fallet med databaser, antalet användare viktat mot hur snabbt en förfrågan besvaras.

Man kan även se skalbarhet som ett mått på att utnyttja den ”nyskalade situationen” till fullo. Exempelvis kanske en applikation flyttas till en mer komplex miljö, med tillgång till kraftfullare processorer, som ger inte bara större krav utan även medför nya möjligheter. Om applikationen är skalbar menas här inte enbart att applikationen bibehåller prestanda, utan även att den kan utnyttja den nya miljön fullt ut, och t.ex. hantera fler användare än förut.

När man talar om skalbarhet kan man tala om den som horisontell eller vertikal. Att skala vertikalt, d.v.s. oftast skala upp, betyder att addera resurser till en nod i systemet, såsom att addera minne eller snabbare hårddisk till en dator. Att skala horisontellt, eller s.k. skala ut, innebär att addera fler noder till systemet, såsom att addera nya datorer till ett kluster.

För försvarets datornätverk och framtida NBF innebär skalbarhet bl.a. krav på att system och resurser kan tillkomma utan att systemet i helhet påverkas och är en viktig parameter för systemets funktionalitet. Vid design av ett system såsom ett mellanlager för informationshantering i logistik i ett sådant heterogent och komplext nätverk, rekommenderas därför att i första hand fokusera på skalbarhet snarare än kapacitet. Detta för att det är betydligt lättare att

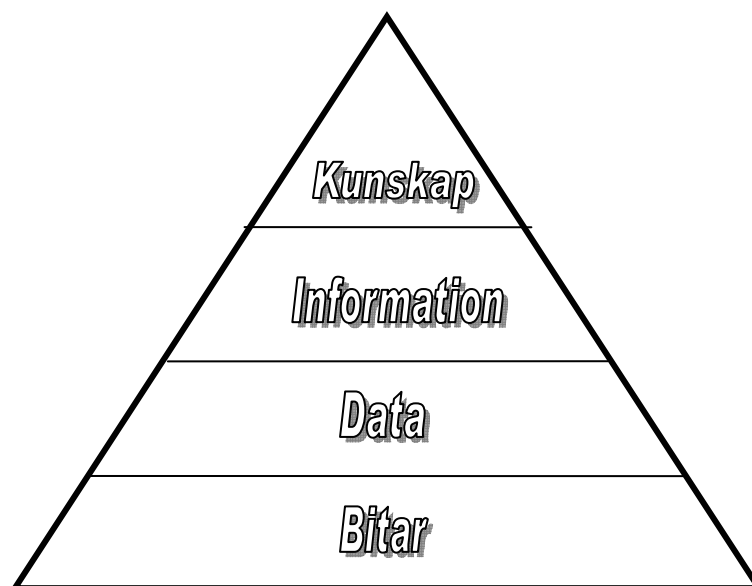
addera noder och resurser till systemet i ett senare skede för att förbättra prestanda, än det är att i efterhand försöka justera systemet och parametrar för att uppnå bättre systemprestanda efter vad varje nod klarar av.

2.3.4 Data, information och kunskap

Det som kommer att hanteras i mellanlagret är data och information. Det är bra att notera skillnaden mellan data, information och kunskap. Till vardags brukar ingen större skillnad göras mellan de olika begreppen men då det handlar om kommunikation via datorer blir det viktigare. Olika författare har olika gränsdragningar för skillnaden, men i stort sätt går de ut på samma sak. För exempel på en sammanställning se [DIK].

Information har en semantik d.v.s. en mening, medan data är uppbyggt av strängar bestående av bitar som i sig inte har någon mening. Data används för att kommunicera ett kodat stycke information och det är först i en viss kontext som det får en mening. I sin tur ger inte information upphov till någon kunskap förrän vi som mottagare har tolkat och förstått informationen.

För att exemplifiera förhållandet mellan bitar, data, information och kunskap kan vi titta på talet 42. När vi gör en förfrågan från en databas kommer talet skickas som den binära strängen 0010 1010. Programvara kommer sedan att tolka strängen till det decimala talet 42. Detta data blir information först när vi får reda på att "42" är svaret på "livet, universum och allting¹". Detta kommer inte att ge oss någon ny kunskap förrän vi har fått reda på *hur* talet 42 är svaret till frågan. För en enkel bild som visar förhållandet mellan de olika nivåerna, från bitar till kunskap, se Figur 2-1.



Figur 2-1: Pyramid som ger en bild av förhållandet mellan de olika begreppen (kunskap, information, data, bitar) och hur de bygger på varandra

¹ Detta exempel är taget från boken "Lifarens Guide till galaxen" skriven av Douglas Adams

Då logistik är ett informationstungt område (på samma sätt som NBF) behöver kraven från informationshanteringen på mellanlagret identifieras. Det finns ett antal frågor som skulle behöva besvaras, exempel är:

- Hur behandlas redundant information?
- Hur behandlas ofullständig information?
- Hur behandlas fusion av information från två källor med olika datatyper?
- Klassificering av data och information?
- Säkerhet och data samt information?
- Hur skall gemensamma begrepp säkerställas?

Vi kommer i denna rapport inte att gå in närmre på detta område då det i sig är stort och skulle behöva en egen studie. Det är inte bara hanteringen av information i mellanlagret som skulle behöva studeras utan även processer för hur inhämtning, modellering, verifiering, validering och accreditering ska göras som skulle behöva studeras. Vi kommer kort att ta upp några tekniker och metoder för informationsmodellering som vi tror är intressanta, se Kapitel 3. I sammanhanget är det därför intressant med några begrepp som brukar förekomma; metabeskrivning, formatering och ontologi.

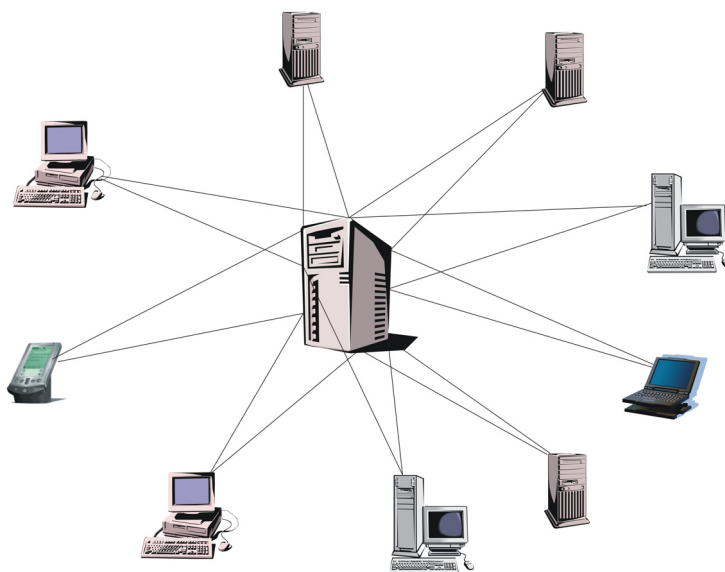
- Metabeskrivningar: ett sätt att sätta data i ett sammanhang. Genom att abstrahera data går det lättare att strukturera upp det
- Formatering: data kan modelleras och formateras på olika sätt. Det viktiga är att det finns en överenskommelse mellan olika parter om hur det ska formateras
- Ontologi: i det här sammanhanget är det en gemensam, avgränsad och strukturerad beskrivning av objekten och deras relationer i en viss domän där syftet är att göra den maskinbearbetningsbar. I ett informationssystem används olika ontologier på olika nivåer. Ett examensarbete som tar upp många av de aspekter som är förknippade med ontologier är [RIIB]

3 Tekniker och Metoder

I ett försvar som det svenska som går mot att bli nätverksbaserat (NBF) blir betydelsen av nätverkstekniker och -lösningar allt större. Dessutom tenderar mjukvara att bli alltmer komplicerad och resurskrävande och nya vägar söks för att möta problemen. Distribuerade datorsystem och relaterade tekniker är i sammanhanget ett viktigt område och en nödvändighet för att möta de nya kraven som försvaret och framtida logistiksystem ställs inför. Det här kapitlet presenterar därför en bakgrund till distribuerade system och relaterade tekniker. Dessutom diskuteras kort ett antal potentiella nätverkstekniker som lämpar sig för syftet och alltså logistiksystem där flera olika och heterogena källor behövs och används.

3.1 Distribuerade datorsystem

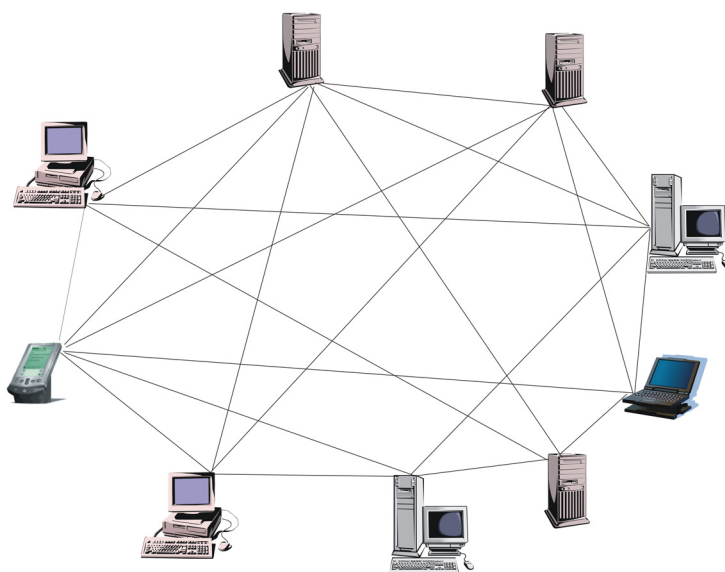
Med dagens ökade behov av datorresurser och ökade krav på datorkapacitet, har området *Distribuerade Datorsystem* vunnit ett stort inflytande på det vardagliga datoranvändandet. Då nätverk distribueras kan resurser och datorkapacitet användas på ett mer effektivt sätt än i traditionella monolitiska system. På detta sätt kan tunga applikationer delas upp i komponenter och exekveras samordnat på flera resurser, istället för att exekveras på en enda beräkningsresurs som då behöver vara betydligt större. Distribuerade system innebär dock inte naturligt att kommunikationen och styrningen inom systemet distribueras. Tvärtom är det vanligt att all kommunikation och styrning leds genom en central enhet, ofta en server, som härmed tvingas hantera stora mängder kommunikationsflöden mellan noderna i nätverket. I Figur 3-1 nedan ses ett distribuerat men centraliserat system.



Figur 3-1: Distribuerat nätverk med centraliserad kommunikation

3.2 Peer-to-peer

Peer-to-peer (P2P) är en term som ofta används när det refereras till en populärt och vida använd nätverksteknik och koncept som innebär att noder i ett nätverk (s.k. *peers*) kommunicerar direkt mellan varandra, se Figur 3-1 och jämför med Figur 3-2. Det helt distribuerade förfarandet tillåter att noder enklare kan dela och använda varandras resurser, där resurser kan vara allt ifrån dokument och kod, till processoranvändning. Den distribuerade styrningen kan, beroende på vad man vill göra, medföra att informationstrafik i nätverken reduceras betydligt i jämförelse med centraliserad styrning, genom att nätverkstrafiken mellan noderna distribueras [Gong].



Figur 3-2: Distribuerat nätverk med distribuerad kommunikation, ofta under benämningen Peer-to-Peer

Många produkter för P2P har utvecklats genom åren. Dock var de program som först väckte kommersiellt intresse de som tillät personer att illegalt dela musik och filmer, såsom Napster och Gnutella. Detta kan vara en av anledningarna till att det dröjde en tid innan organisationer och företag anammade P2P-konceptet. Nu har P2P fått ett betydligt större inflytande i affärsnätverk, och många organisationer, affärssystem, samt andra system baseras på P2P-metodologi. Exempel på existerande tekniker är JXTA från Sun och Jini från Microsoft.

3.3 Agentsystem

En agent är i datorsammanhang mjukvara som agerar och reagerar autonomt till fördel för en klient [Tanenbaum et al.]. Den interagerar aktivt med den omkringliggande datormiljön som den arbetar i, eventuellt i samarbete med andra agenter, för att utföra ett uppdrag eller uppnå ett mål för ett visst syfte. En agent kan också ha flera olika uppgifter och den kan sköta en uppgift efter ett givet mönster, eller beteende, som ofta benämns "strategier". Som exempel kan agenter användas för att användaranpassat söka och filtrera information åt en klient, möjligen även i samarbete med andra agenter. Dessutom kan agenten dynamiskt anpassa sättet på vilket informationen presenteras efter klientens rådande miljö och förutsättningar [Weiss].

Mobila agenter är agenter som kan förflytta sig i nätverk mellan olika maskiner och system, i heterogena miljöer. Denna egenskap tillåter programvara att migrera till annan plats och därefter exekveras på distans på till exempel en server, utan att under tiden vara till belastning för klienten [Wilson et al.]. Mobila agenter används ofta i distribuerade system för distribution av information och funktionalitet. På så vis kan informationsflödet i nätverket minimeras gentemot det traditionella sättet som förfrågningar hanteras på, det vill säga en fråga skickas till en tjänst eller en mängd tjänster, som besvarar frågan genom att skicka all möjlig information över nätverket till klienten. I ett nätverk där källorna och resurserna som används är många och heterogena, såsom i logistiksystem och försvarsnätverk, är därför agenter en mycket lovande teknik.

Området agenter och agentsystem, samt en mer ingående genomgång av ett antal intressanta agentramverk, presenteras mer detaljerat i Kapitel 4.

3.4 Web Services

3.4.1 XML

Extensible Markup Language (XML) är namnet på ett idag mycket populärt och använt märkspråk. Det baseras på SGML (Standard Generalized Mark-Up Language) som är föregångaren till det enkla märkspråk som gjorde information på webben till vad det är idag: Hyper Text Markup Language (HTML). XML tillhandahåller ett flexibelt sätt för att strukturera och kapsla in information, samtidigt som det på ett fördelaktigt sätt separerar data från presentation. Språket är textbaserat och således läsbart av människan, samtidigt som det är plattformsoberoende [Williamson].

XML tillhandahåller ramar för att användare ska kunna utveckla sina egna specifika märkspråk. Detta görs genom att skapa mallar s.k. XML Schemas. I XML Schema definieras ett antal regler som ett XML-dokument ska följa, t.ex. struktur och semantik för innehållet. Mallen kan sedan användas för att validera innehållet i dokumentet och att det beskrivs på en form som är enhetlig och tolkas på ett korrekt och gemensamt sätt. På detta sätt kan XML användas inte bara för att lagra och strukturera data för webben, utan även som en plattformneutral och interoperabel metod för organisationer och företag att dela information mellan sig och olika system i heterogena miljöer.

3.4.2 Konceptet

Web Services (WS), eller på svenska webbtjänster, är såsom namnet antyder tjänster som tillhandahålls via webben. Allteftersom gränser och samarbeten mellan företag och organisationer har breddats, har behovet av en gemensam teknik och metod för att dela tjänster och applikationer mellan dem uppdagats [Nagappan et al.]. Ur detta behov växte WS-konceptet fram, som precis som föregångare som RMI och Corba är baserade på konceptet bakom distribuerade system. För att möjliggöra för interoperabilitet mellan företag och organisationer baseras WS på öppna Internet-tekniker och –standarder. En stor del i utvecklingen har organisationen *World Wide Web Consortium* (W3C) haft.

“Web Services are loosely coupled software components delivered over Internet standard technologies.” [Nagappan et al.]

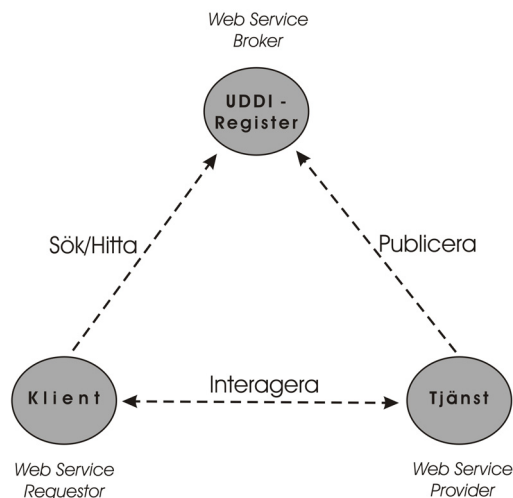
Webbtjänster baseras på en tjänsteorienterad arkitektur, där alla funktioner och beräkningar (d.v.s. mjukvarukomponenter) som en nod kan erbjuda publiceras som en tjänst och blir tillgänglig över webben och nätverk [Nagappan et al.]. En förfrågan kan sedan skickas till tjänsten varefter denna processar den och returnerar svar till klientapplikationen. Webb-tjänsten kan i sin tur utnyttja ett antal andra tjänster för att besvara förfrågan. På detta sätt kan företag och andra dela på, inte bara data, utan även tjänster och funktioner. WS-tekniken har vunnit stort intresse i bl.a. nätverksvärlden och används ofta i affärsapplikationer, webbhandel och många andra typer av distribuerade applikationer. Ett populärt kommersiellt användningsområde för webbtjänster är i handdatorer för vägbeskrivningar. En användare utnyttjar då en webbtjänst för att exempelvis få reda på var närmaste restaurant ligger och få visat i handdatorn närmaste vägen dit. Användaren skickar en förfrågan till en webbtjänst med sin adress och en fråga om var närmaste restaurant finns. Webbtjänsten kan i sin tur utnyttja ett antal andra tjänster för att besvara förfrågan, en kedja av förfrågningar [Weiss], innan användaren får tillbaka svar och vägbeskrivning direkt i sin handdator.

3.4.3 Arkitekturen

WS-konceptet tillhandahåller ett antal tekniker och standarder för att publicera tjänster över Internet och på ett standardiserat sätt. I huvudsak är det tre tekniker som möjliggör webbtjänster [Nagappan et al.]:

- SOAP – Står för Simple Object Access Protocol och är ett grundläggande meddelandeprotokoll för WS. Eftersom SOAP baseras på XML är även protokollet utvidgningsbart och tillhandahåller standardiserat och strukturerat sätt att skicka meddelanden. Protokollet stödjer meddelandehantering baserat på flera olika nätverksprotokoll
- WSDL – Står för *Web Services Description Language* och är skrivet i XML (är ett XML-dokument). WSDL tillhandahåller ett standardiserat sätt för att beskriva WS och används för att lokalisera tjänster. Dessa residerar i en WS-container
- UDDI – står för *Universal Description, Discovery and Integration*. UDDI är en standardiserad mekanism för att publicera och upptäcka publicerade WS. En implementering av UDDI-specifikationen benämns UDDI-register. I ett UDDI-register lagras information om publicerade webbtjänster. Klienter skickar förfrågningar till UDDI-register för att söka efter tjänster. All kommunikation sker över SOAP

I Figur 3-3 nedan ses en klient som utnyttjar och lokaliserar tjänster (s.k. *Web Service Requestor*), en nod som tillhandahåller en WS (s.k. *Web Service Provider*), samt ett UDDI-register som är värd åt de publicerade tjänsterna (s.k. *Web Service Broker*).



Figur 3-3: Enkel figur över grundläggande mekanismer i Web Services för att registrera, söka och lokalisera en webbtjänst

Tjänsten i figuren har registrerats som ett WSDL-dokument hos ett UDDI-register. Klienten frågar ett UDDI-register efter en tjänst, som beskrivs och kan lokaliseras via WSDL-dokumentet. Därefter kan klienten utnyttja tjänsten (initiera den) direkt och över kommunikationsprotokollet SOAP.

3.5 Grids

Ett viktigt nytt område som har växt fram ur Distributed Computing-domänen är Grid Computing. Termen Grid myntades i mitten av 90-talet och som en motsvarighet till den engelska benämningen för kraftnätverk, *Power Grids*. Kraftnätverk består av en mängd noder som tillsammans ser till att vårt land förses med elektricitet. Användaren kopplar in sig mot nätet och utan att se noderna får han/hon el utan att behöva tänka på nätverket bakom. På samma sätt kan man se på Grids, där användaren som kopplar upp sig direkt får tillgång till en stor datorresurs, en virtuell superdator, som i sig egentligen består av en mängd mindre noder (resurser) och tillhandahåller tjänster för att dessa ska kunna utnyttjas på ett transparent sätt (såsom elektricitet) [Baker et al.]. På så vis kan en användare nyttja Grid-systemet till att utföra stora beräkningar, utan att själv behöva tänka på hur problemet ska delas upp för syftet. För Grids krävs därför (till skillnad från P2P) extra tjänster som hanterar Grid-beräkningarna, det vill säga som delar upp och delar ut problemet i olika delar för beräkning på de olika resurserna m.m.

Grids är sålunda en avancerad metod för delning av distribuerade resurser; datorresurser, filer etc. [Foster et al.]. Många andra distribuerade nätverkstekniker (såsom P2P) erbjuder möjligheter för att dela på resurser, men tillhandahåller inte de tjänster som behövs för att koordinerat (och transparent) utnyttjar resurserna. Som exempel kan nämnas Internet-tekniker som ofta adresserar kommunikations- och informationsutbyte mellan datorer, men som inte tillgodoser integrerade lösningar för att koordinerat utnyttja resurser på multipla platser för beräkning.

Open Grid Services Architecture

En standard för Grids är OGSA, Open Grid Services Architecture, som är en *middleware*-modell och arkitektur. OGSA bygger på och kombinerar koncept och tekniker från *Web Services* och Grid-domänen, där Grid-konceptet implementerar en tjänsteorienterad arkitektur [Foster et al.]. OGSA tillhandahåller på detta sätt en uniform och standardiserad tjänstesemantik, samt mekanismer för att skapa, namnge och upptäcka Grid-instanser, multipla protokollbindningar, lokaliseringsfunktionalitet mm. Då OGSA baseras på WS medför detta att även OGSA ger möjligheter att standardiserat skapa sofistikerade distribuerade system, inklusive all hantering det innebär. För att sätta samman tjänster används till exempel såsom i WS WSDL, och medför bl.a. att utvecklaren kan sätta samman komplexa nya Grid-tjänster av andra tjänster.

Globus Toolkit är en referensimplementering av nyckelprotokoll för Grids som stödjer en bred massa av e-science-projekt [Foster et al.]. Det är *Open Source* och den enda fullskaliga implementationen av OGSA.

3.6 Informationsmodellering

Informationsmodellering är något som kommer att behöva göras för att många heterogena system skall kunna samverka. Det finns en del tekniker som har utvecklats inom ramen för *the Semantic Web* och som har standardiserats med hjälp av organisationen W3C och DARPA. Vi tar endast upp dessa tekniker översiktligt här. För den som önskar läsa mer om detta område kan t.ex. referenserna [IML, RIIB, RDF, OWL, DAML] vara av intresse.

RDF och OWL

RDF står för *Resource Description Framework* och OWL står för *Web Ontology Language*. OWL är på samma sätt som RDF en standard för att beskriva och hantera resurser och data på webben. Dessa båda standarder tillsammans med XML är viktiga pusselbitar i informationsmodellering. OWL och RDF är inte beroende av XML men de används gärna ihop. XML tillhandahåller regler och syntax för strukturerade dokument. RDF ger ett ramverk för att hantera och beskriva data. OWL tar hand om behovet att utveckla ämnes- eller domänspecifik vokabulär, d.v.s. ontologier.

DAML

DAML är ett DARPA¹-initierat program med målsättningen att utveckla språk och verktyg för att underlätta *the Semantic Web*. Samma person som står bakom programmen UltraLog och Cougaar, Dr. Mark Greaves, se Avsnitt 4.4.3 och 4.5.3, är även ansvarig för detta projekt. DAML som är en förkortning av the DARPA Agent Markup Language, är ett projekt som började utvecklas 2000. Målsättningen är att möjliggöra automatisk lokalisering, selektering, sammansättning, exekvering och monitorering av webbaserade tjänster. I projektet DAML tillhandahålls i princip en infrastruktur för detta och bl.a. har en ontologi för beskrivning av tjänster, d.v.s. webbaserade resurser som tillhandahåller någon form av interaktion, gjorts (DAML+OIL). Med dessa maskintolkbara beskrivningar skall sedan agenter kunna agera på uppdrag av användare och system som vill utföra en specifik uppgift. Numera har man dock övergått till OWL inom detta projekt men flera andra verktyg och intressanta idéer har tagits fram som är värda att undersökas närmare.

3.7 Slutsatser

Distribuerad teknik är det mest lämpliga alternativet för ett framtida försvarssystem såsom det nätverksbaserade logistiksystem som är fokus för denna förstudie. P2P är ett intressant och lovande koncept, men då det ännu inte finns någon standard blir det för närvarande en metod som kan väntas med. Antagligen kommer det framöver att kunna integreras och bli en naturlig del av ett framtida logistiksystem samma gäller för GRIDS-konceptet.

Ett annat intressant koncept är *Web Services*, det följer tjänstestrukturen som är avsedd för det framtida försvaret och det är en standard. Det är accepterat av industrin vilket alltid är en fördel då integration och kommunikation mellan olika system utvecklade av olika aktörer behöver göras.

Agenter är intressant och lämpligt för syftet, dessutom är det en teknik som har använts länge. Etablerade logistiksystem utnyttjar detta såsom UltraLog med flera. Även om vi vidare i rapporten endast fördjupar oss i agenttekniken, så utesluter inte detta att även de andra tekniker och metoder utnyttjas, möjligen i kombination med agenter.

Informationsmodellering är viktigt och något som behöver göras. Det finns tekniker för det som är standardiserade och accepterade av industrin och försvaret. Däremot är metodiken inte lika väl utvecklad och den skulle behöva mer arbete. XML är en accepterad standard och det är just nu det mest naturliga att använda för informationshantering tillsammans med *WebServices*.

¹ DARPA –Defense Advanced Research Projects Agency. Det är den amerikanska motsvarigheten till FOI. För mer information se <http://www.darpa.mil/>

4 Agentsystem

I Avsnitt 3.3 förklarades kort vad en agent innebär när man talar om begreppet i datorsammanhang. Eftersom agenter och agentsystem delvis har varit fokus i projektarbetet som potentiell teknik för mellanlagret, är därför detta kapitel en fördjupning i ämnet. Först diskuteras agenter generellt och i mer detalj än i 3.3. Därefter diskuteras användning av agenter inom logistik. Slutligen presenteras tre lovande agentramverk för sammanhanget, som studeras och jämförs.

4.1 Agenter i distribuerade system

Agenter används ofta till att integrera olika komplexa mjukvarusystem. En av huvuduppgifterna i detta arbete är att gömma komplexitet för användaren. Beteendet och logiken kapslas här in i agenten för att autonomt kunna besvara förfrågningar och även automatiskt agera utifrån ändrade situationer för klientens räkning. Agenterna kan även inneha lärande beteende, det vill säga agenten lär sig allteftersom den agerar och får uppgifter. Detta innebär att agenter själva kan resonera kring omvärlden och lättare kan anpassa sig till klienten och nya förutsättningar [Weiss].

En agent kan generellt sägas vara strukturerad som en samling av element:

- *Tjänster*: funktioner som tillhandahålls för andra agenter
- *Mål*: självinitierade uppgifter, det vill säga uppgifter för att utföra mål för sig själv och/eller sin ursprungliga klient
- *Resurser*: information om externa resurser, tjänster, bibliotek, ontologier, grupper etc.
- *Interna objekt*: interna datastrukturer som delas av alla processer som agenten startar för att lösa tjänsteförfrågningar eller för att uppnå mål
- *Kontroll*: specifikation över hur tjänsteförfrågningar hanteras av agenten

4.2 Mobila agenter

Mobila agenter är agenter som kan förflytta sig i nätverk mellan olika maskiner och system, i heterogena miljöer. Denna egenskap tillåter programvara att migrera till annan plats och därefter exekveras på distans på till exempel en server, utan att under tiden vara till belastning för klienten. Detta möjliggör även för dynamisk lastbalansering och parallella beräkningar mm. [Wilson et al.].

Mobila agenter används ofta i distribuerade system för distribution av information och funktionalitet. Agenten kan på detta sätt själv bära med sig information och funktionalitet i nätverket och överlämna det på sin specifika plats. Detta medför även att en agent kan svara på en förfrågan av en klient och utföra beräkningar på distans, utan att klienten behöver blandas in under tiden. Istället kan agenten söka själv i nätverket och endast återlämna det efterfrågade resultatet till klienten. På så vis kan informationsflödet i nätverket minimeras gentemot det traditionella sättet som förfrågningar hanteras på, det vill säga en fråga skickas till en tjänst eller en mängd tjänster, som besvarar frågan genom att skicka all möjlig information över nätverket till klienten. Det har visat sig att mobila agenter är mycket lämpade för detta syfte. I ett nätverk där källorna och resurserna som används är många och heterogena, såsom i logistiksystem och försvarsnätverk, är därför agenter en lovande teknik för sammanhanget.

4.3 Agenttyper

Agentsystem består ofta inte av en enda agenttyp, utan av flera agenter som har olika funktionalitet och egenskaper och som utnyttjar olika ”strategier” för att uppnå mål. Agenterna kan även grupperas, antingen efter vilka tjänster eller egenskaper de har att erbjuda, eller för någon annan orsak (egenskaper, rättigheter och så vidare). För att exemplifiera användningen av agenter nämns nedan ett antal agenttyper och med den engelska benämningen de brukar ha:

- *Profiler Agents* – Kan vara stationära agenter som profilerar, alternativt filtrerar, information så att den passar antingen den typen av användare som den server, eller den person som den server (alltså helt personprofilerad)
- *Maintenance Agents* – Kontrollerar att ett system fungerar och om det inte gör det så åtgärdar den det om den kan, eller varnar användaren
- *Filter Agents* – En filtreringsagent filtrerar information med avseende på önskemål från användaren, och/eller med grund i agentens eget beslutsfattande. En filtreringsagent kan finnas på olika platser i ett informationssystem, men om den används som en mobil agent och skickas till källan för informationen, så kan informationsflödet minimeras ”på plats” och reducera trafiken
- *Search Agents* – Söker efter information genom att resa runt i ett nätverk och/eller genom kontakt med andra agenter. Agenten tar antingen med sig hela informationen tillbaka, eller bara en adress. Sökningen sker efter önskemål från användande och/eller efter agentens eget beslutsfattande
- *Distribution Agents* – Distribuerar resurser eller information till de platser det är lämpligt eller får plats. Nära besläktade med *Resource Allocation Agents* som används för att identifiera och allokera lämplig plats för en resurs att antingen lagras eller exekveras
- *Watch Agents* – Ofta mobil agent som klonas och skickas iväg till det ställe som den ska passa. Agenten ligger och väntar tills något som är intressant sker. Först då får användaren reda på det. Användaren behöver alltså inte själv vara uppkopplad för att agenten ska kunna verka, utan agenten väntar tills användaren är uppkopplad igen med att berätta nyheten. Används ofta i mäklartjänster etc. på Internet
- *Software Agents* – Kontrollerar att en användare har lämplig mjukvara för att kunna ta emot och använda en resurs. Om agenten ser att användaren inte har de förutsättningarna, så ser den till att användaren får det. En sådan agent kan antingen vara stationerad hos en användare, eller skickas med en resurs till en användare

Många agentsystem består av en kombination av agenttyper, där vissa av agenterna är mobila och vissa inte. På så vis kan det även implementeras olika nivåer av säkerhet i agenterna och de kan anpassas efter olika plattformar o.s.v.

4.3.1 Agenter och säkerhet

När agenter omnämns, är säkerhet ett ofta omtalat ämne i sammanhanget. Med säkerhet avses här inte systemsäkerhet, det vill säga robusthet i systemet och att det inte finns någon *single-point-of-failure*, utan det som diskuteras är säkerhet vad gäller information och intrång i hårdvara etc. Detta är inte alls konstigt med tanke på naturen hos agenter och agentsystem. En agent är i praktiken ett program som får tillgång till en dator och kanske till och med får rättigheter att ändra och skriva på mottagardatorn, samt dessutom agerar autonomt i sitt arbete. Med andra ord agerar agenten i likhet med många virus och det är inte svårt att se problemen som kan uppkomma med denna egenskap [Janssen et al. 1].

Eftersom agenter ofta består av mobil mjukvara som kan förflytta sig autonomt över nätverk, finns här ett flertal olika aspekter av säkerhet som måste beaktas till skillnad från andra distribuerade system. För att åtgärda problemen finns ett antal väl etablerade metoder och nivån av säkerhet varierar mellan olika system, samt är beroende av vilka uppgifter en agent utför och vilken information den har tillgång till. Agentsäkerhet kan huvudsakligen delas upp i tre hot [Janssen et al. 2]:

- *Hot #1: "Fientliga" agenter.* Intrång hos mottagare från s.k. "*malicious agents*", alltså agenter som är felaktiga eller på annat sätt kan skada den mottagande värdmiljön. Det vill säga klienten (mottagaren, det mottagande systemet, eller den mottagande klienten etc.) måste skyddas mot intrång från "elaka" agenter
- *Hot #2: "Fientlig" värd.* Värden som äger (skickade iväg agenten) eller tar emot agenten maskeras som en vanlig värd men är felaktig och kan påverka agenten och agentkoden och/eller skicka iväg agenten på oönskade uppdrag. Den kan dessutom ta emot information via agenterna som egentligen inte är menad för den, eller ge agenten felaktig information som den sprider i nätverket. För detta måste både andra noder (värdar) i nätverket och agenter skydda sig
- *Hot #3: "Man-In-The-Middle".* Då agenterna transporteras kan de påverkas under vägen till och från platser där de löser de uppgifter de ska utföra. Detta gör agenterna sårbara under transport, då den information agenterna bär på och koden som utgör agenten skulle kunna modifieras av fientlig programvara. I detta fall är det alltså agenten själv som måste skyddas från yttre påverkan

Eftersom agenter är autonoma (till den grad som programmeraren/utvecklaren väljer) uppstår ett antal relaterade frågor som är bra att besvara *innan* ett agentbaserat system utvecklas. Speciellt viktigt är det om systemet är tänkt att existera i exempelvis ett försvarsnätverk som har höga krav på säkerhet:

- Hur mycket autonomitet och befogenhet får en agent ha i ett svenskt försvarsnätverk?
- Vem hålls ansvarig för en agents agerande och händelser?
- Hur garanteras en klients integritet i ett system av "själv tänkande" agenter?
- M.fl.

4.3.2 FIPA

The Foundation for Intelligent Physical Agents (FIPA) är en icke-vinstdrivande organisation i Schweiz [FIPA]. Huvudsyftet med organisationen är att öka användandet av agentbaserade applikationer, tjänster och annan mjukvara. Detta uppnås genom att tillhandahålla internationellt överenskomna specifikationer som underlättar interoperabilitet mellan agentsystem. Det vill säga de agenter som följer FIPA-standarderna är kompatibla med varandra, och de som följer FIPAs specifikationer för agentspråk och kommunikationsprotokoll kan kommunicera med och förstå varandra. Specifikationerna tas fram i samarbete mellan universitet och industri och är öppna för allmänheten. Förutom specifikationerna tillhandahåller FIPA även generiska tekniker som kan användas som bas i utvecklingen av komplexa agentsystem som har en hög grad av interoperabilitet gentemot andra system.

Arbete pågår fortlöpande med att komplettera och förbättra FIPA-specifikationerna. Detta i takt med att allt fler i agentvärlden blir medlemmar i FIPA och/eller implementerar standarden i sina agentramverk och -system.

4.4 Agenter i logistik

4.4.1 Extra höga krav på tekniken

Inom logistikområdet ställs särskilt höga krav på den teknik som stödjer informationshanteringen. Detta på grund av de stora mängder information som behöver hanteras (insamlas, sökas, fusioneras och presenteras). Här behövs en teknik som kan hantera massiva informationsflöden, samtidigt som den är skalbar och inte degraderas i takt med att det nätverksbaserade försvaret introduceras. Dessutom är datorsystemen inom FM heterogena, med olika typer av system och varierande antal och typer av användare.

Logistiksystemet i FM måste vara flexibelt nog att snabbt kunna svara mot plötsligt ändrade villkor och komplexa situationer, såsom beskrivs i Scenario 1 – *Internationell Operation* där en snabb situation inträffar som innefattar kommunikation mellan många olika parter och till och med över landsgränserna. Dessutom måste systemen vara robusta i en mycket dynamisk datormiljö, vilket exemplifieras i Scenario 2 – *Krissituation* där en situation uppkommer som kräver snabba insatser från olika domäner och där de inblandade förmodligen även befinner sig på resande fot (vilket ofta innebär dynamisk nätverksuppkoppling). I dessa två scenarion, med snabba förändringar och flera inblandade system, demonstreras även vikten av att systemet självt ger stöd för användaren och minimerar komplexiteten för användaren. Här är det extra viktigt med vald nätverks- och kommunikationsteknik, samt vilken logik som finns inbäddad i logistiksystemen.

4.4.2 Mobila agenter för logistik

Mobila agenter kan vara en teknik som används för dessa problem inom logistik. Agenter kan dessutom hjälpa till att *effektivisera* och *reducera* informationsflödet i nätverket. Vilket t.ex. kan göras genom att selektera information på plats från flera källor, istället för att skicka information från olika källor över nätverket och hantera detta först på klientsidan. Agenter kan kommunicera med varandra, och härigenom reducera och effektivisera informationsflödet i nätverket. Dessutom kan kunskap och beteende byggas in i agenterna och genom att de även kan kommunicera och samverka kan de utbyta kunskap och samordna beteende i syfte att samverka till att lösa ett problem. På detta sätt kan agenterna ge bättre stöd till klienten och själva fatta beslut, vilket gömmer komplexiteten från användaren. Agenter har dynamiskt och flexibelt beteende, vilket betyder att de kan ha möjlighet att anpassa sig efter rådande och dynamiska omständigheter. Agenter kan här stödja och anpassa sig efter rådande omständigheter.

För att sammankoppla äldre och nya logistiksystem med heterogena användare krävs ett flexibelt mellanlager. För detta kan mobil agentteknik vara en bra lösning, men det behöver inte nödvändigtvis utesluta andra tekniker eller integrering av någon ytterligare teknik. Till exempel skulle en infrastruktur baserad på annan teknik kunna användas i kombination med ett agentsystem. Ett intressant uppslag vore att kombinera så kallade webbtjänster (*Web Services*) med agenter. Detta och ett förslag på en sådan kombination diskuteras vidare i Kapitel 5.

4.4.3 UltraLog

UltraLog är ett DARPA-sponsrat forskningsprojekt med fokus på att skapa robusta, storskaliga, överlevnadskraftiga och distribuerade agentsystem där systemen ska vara kapabla att operera effektivt i mycket kaotiska miljöer såsom t.ex. militära taktiska situationer och under kristid. Målet för projektet är att skapa ett lättillgängligt och lättförståeligt distribuerat agentsystem som på en stor skala möjliggör, för logistiksyften, en säker infrastruktur som kan fungera under svåra omständigheter. Utvecklingen och forskningen baseras framförallt på agentramverket Cougaar, *Cognitive Agent Architecture* och den huvudsakliga utmaningen för projektet är att utöka agent-

arkitekturen med ett lagerbaserat system av tekniker som hanterar robusthet, säkerhet, stabilitet och skalbarhet [UltraLog obj].

Ursprungligen utvecklades Cougaar av projektet *Advanced Logistics Project*, ALP, som var ett 5-årigt DARPA-program med målsättningen att utveckla innovativa sätt att lösa logistikrelaterade problem. Ett av resultaten från det projektet var en ny och kraftfull agentarkitektur som baserades på det mänskliga kognitiva sättet att planera. Det utvecklades för DARPA:s räkning av ALPINE, *Advanced Logistics Program Integration and Engineering*, ett konsortium helt bestående av "BBN Technologies". Agentarkitekturen Cougaar är numera ett *Open Source*-projekt och fritt tillgängligt från hemsidan [Cougaar].

UltraLog-projektets tre huvudmål är att:

1. Utöka och förbättra Cougaar-funktionaliteten till att motsvara UltraLogs krav för robusthet, skalbarhet och säkerhet mm. för kaotiska miljöer och förhållanden
2. Generera mjukvaruprodukter för mjukvarudesign, -utveckling, -analys och -integrering inom UltraLog-gruppen
3. Utveckla en agentbaserad arkitektur och nödvändigt stöd för att tillhandahålla bakåtkompatibilitet samt att metodiken och tekniken ska spridas och användas av fler

Projektet arbetar också på att forska fram och utveckla en prototyp som kan användas för olika demonstrationer. Den slutgiltiga prototypen och demonstrationen ska bestå av över 1000 robusta och intelligenta agenter som dynamiskt balanserar logistikuppgifter och systemförsvar. Samtidigt ska de operera inom en föränderlig miljö som konstant utsätts för olika typer av attacker som t.ex. informationskrigsföring. Slutdemonstrationen skall hålla på under en period som motsvarar 180 dagar av militära operationer och kraven från UltraLog är att kunna operera:

- med upp till 45% informationsförlust
- i en miljö som är 90% så kaotisk som den mest kaotiska verkliga miljön
- med inte mer än 20% degradering av dugligheten
- och inte mer än 30% degradering av prestandan

4.4.4 Praktiskt exempel – Seaway

Seaway är ett beslutsstödssystem inom amerikanska försvaret som är till för marin logistisk planering och koordinering. Systemet är helt implementerat med hjälp av agenter som kommunicerar och inte bara med varandra utan även med direkt inflytande av expertis. Seaway utvecklades för att komma till rätta med behov inom logistikrelaterad verksamhet för US DoD. Systemet är ännu experimentellt och alltså inte i fullt bruk. Anledningen till att de utnyttjar agentteknik är för att kunna bygga in beteende och samordning i systemet och samtidigt få det skalbart. En annan anledning är att systemet ska vara anpassningsbart efter den dynamiska variation av användare och omständigheter som råder i försvaret och i kritiska situationer. Det vill säga systemet är tänkt att användas som adaptivt ledningssystem inom marinen och i system som beslutsstöd för att planera resurser som behövs och kommer att behövs ombord. Det ska även användas i samband med geografisk planering och föra status över positionering etc. För mer information se [Seaway].

4.5 Genomgång av potentiella agentramverk

Alltsedan agenter som koncept initierades har det varit ett forskningsämne som har behandlats i många forum och inom många områden. Från att ha varit ett område som blev mycket populärt i och med sin närhet till Artificiell Intelligens (AI), har det blivit ännu en teknik som med framgång används i distribuerade system. För detta syfte har många ramverk och system utvecklats för design och implementering av agenter och agentsystem. Med ramverk menas här plattformar och produkter med hjälp av vilka man bygger agentsystem, det vill säga system som består helt eller delvis av agenter. Några av ramverken är stora och används idag i kommersiella produkter och på många marknader för många syften. Ett sådant agentramverk är till exempel Aglets från IBM som är en plattform för mobila agenter som är avsedda främst för att färdas över Internet och samla in och hantera information från flera källor. Ett annat kommersiellt stort agentramverk är RETSINA, som står för *Reusable Environment for Task Structured Intelligent Network Agents*. Enligt [RETSINA] har RETSINA applicerats inom ett flertal olika domäner såsom portföljhantering (finansmarknaden), personliga (användaranpassade) webbtjänster, bokhandel och auktioner på nätet, logistikplanering inom militära domänen, samt för mobil och trådlös kommunikation. Dock är RETSINA ett kommersiellt ramverk, som således inte bara kostar, utan även är begränsat för egen vidareutveckling. Aglets, som från början såg ut att vara en lovande teknik, är sedan flera år inte längre under utveckling och inga nya versioner av plattformen har utkommit de senaste åren.

En studie (genomgång) av ett antal agentramverk har genomförts, där de jämfördes och tre intressanta kandidater identifierades inför fortsatt arbete och utvärdering för syftet. En viktig parameter vid urvalet var att ramverken skulle vara *Open Source*, det vill säga de är utvecklade på ett icke-vinstdrivande sätt och härmed är all källkod och dokumentation öppen, gratis och fri att modifiera. Detta är bra för vidareutveckling och egenanpassning av ramverken. Dessutom skulle de vara implementerade i Java. Detta sågs som ett naturligt steg då systemen är avsedda för nätverk och dessutom för att vara plattformsoberoende och anpassade för en så heterogen miljö som möjligt. Java är det idag existerande programmeringsspråk som till störst del uppfyller dessa kriterier.

För vidare arbete valdes ramverken DIET Agents, JADE och Cougaar. Det var de tre ramverk som verkade ha störst potential för uppgiften. De parametrar som behandlas under utvärderingen av dessa var generella och ansågs särskilt viktiga för syftet. Det är viktigt att påpeka att även om ett agentramverk exempelvis är robust så är det inte automatiskt det för alla tillämpningar. Agentramverken tillhandahåller möjligheter och ramar, men det är fortfarande upp till utvecklaren att designa och implementera applikationerna noggrant för att uppnå önskad nivå. De tre utvalda ramverken presenteras nedan. En kortfattad beskrivning av de övriga agentsystemen som behandlades i studien presenteras i Appendix B.

4.5.1 DIET Agents

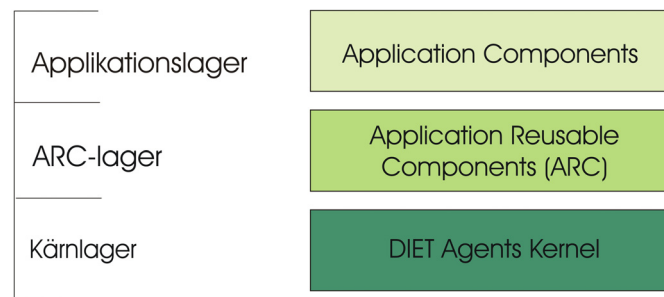
DIET står för Decentralized Information Ecosystem Technologies. Projektet var ett europeiskt samarbetsprojekt under EUs 5:e ramverk [DIET Agents]. Ett av målen med projektet var att studera och implementera ett ramverk för att hantera och processa information i en robust, adaptiv och skalbar miljö. Forskningen behandlade bl.a. interaktion mellan agenter i multiagent-system och alternativa former för detta, samt forskning kring implementering av en informationsstruktur (plattform) som svarade mot de ställda kraven. För att demonstrera och validera användbarheten av plattformen skulle ett antal uppgifter kunna utföras: informationsinhämtning, -filtrering, -fusionering och -publicering.

Forskningen och implementeringen resulterade i agentplattformen DIET som användes som plattform för experimentell forskning. När projektet avslutades i Juli 2003 släpptes ramverket och källkoden öppen för allmänheten och vidare utveckling.

DIET-ramverket är en multiagent-plattform implementerad i Java. Efter att DIET-projektet avslutades släpptes plattformen som *open source*. Enligt [DIET Agents] är ramverket särskilt bra för att snabbt utveckla P2P-applikationer eller adaptiva distribuerade applikationer. De parametrar som premierades i utvecklingen av ramverket var skalbarhet, robusthet och adaptivitet, vilka därför tillhör de största fördelarna med DIET.

Arkitektur

Arkitekturen för DIET är uppdelad i tre lager, se Figur 4-1. Det översta lagret, Applikationslagret, utgörs av applikationsspecifika komponenter. Under det ligger mellanlagret i DIET-arkitekturen, ARC-lagret (*Application Reusable Components*). ARC-lagret tillhandahåller en infrastruktur för återanvändbara komponenter där agentbeteenden integreras som *plug&play*-komponenter¹, det finns generisk schemaläggning av agentuppdrag som möjliggör dynamisk planering av uppdrag, samt agentstrategier som hanterar fel och genomförda uppdrag etc.



Figur 4-1: Enkel arkitekturbild för DIET, lagerbaserad i tre delar.

Längst ner i arkitekturen finns kärnan i DIET-ramverket, *DIET Agents Kernel*. Kärnan tillhandahåller den (minimala) mjukvara som krävs för att implementera *multiagent*-funktionalitet. Detta inkluderar bland annat:

- Ett konceptuellt agentramverk med komponenter såsom omvärldar för agenter, agentmiljöer, agenter, meddelanden (kommunikationsmekanismer), etc.
- Minimal funktionalitet för agenter som tillåter att de kan skapas, förstöras, kommunicera, migrera, etc.
- Mekanismer för parallell exekvering av agenter
- Innehållsstrukturer för meddelanden => gemensam bas för kommunikation och förståelse

¹ Med *plug&play*-komponenter avses här komponenter som kompletta följer med produkten och som går att exekvera utan modifikation eller egenutveckling.

Kommunikation

Komplexa kommunikationsprotokoll var inte något som premierades i implementeringen och designen av DIET. Därför stöds inte så många olika typer av kommunikationsprotokoll och inte heller olika meddelandemekanismer såsom en gemensam infrastruktur eller ett gemensamt språk för meddelanden i DIET. Däremot tillhandahåller DIET mycket enkla och icke-minneskrävande¹ funktioner för kommunikation mellan agenter, vilket gör kommunikationen mellan DIET-agenter mycket snabb.

Säkerhet

Säkerhetsfrågan har i DIET, precis som i många andra distribuerade system och agentsystem, inte behandlats ordentligt. Inga särskilda säkerhetsmekanismer finns inbyggda i systemet. Detta är en klar brist i ramverket och det är osäkert hur svårt det är att lägga till sådan funktionalitet.

Robusthet

DIET är en robust agentplattform, mer än många andra, på så vis att den själv tillhandahåller mekanismer för att upptäcka överbelastning och fel i systemet. Felen lokaliseras och systemet ser till att åtgärda detta och ev. flytta en exekvering till annan plats för att fortsätta exekveringen där. En stor nackdel här är att DIET inte stödjer migrering av kod, utan istället bara överflyttar status för en agent till en nod och där startar upp en helt ny instans av agenten med medflyttat status. Detta hämmar mobilitet och flexibilitet hos systemet, genom att till exempel tvinga mottagaren att tillhandahålla funktioner för att kunna starta upp exakt den agent som ska förflyttas, med den status som överförs.

Skalbarhet

Plattformen är skalbar både på lokal och på global nivå. Lokal skalbarhet uppnås genom att DIET-agenter är designade för att kunna vara mycket *lightweight*, det vill säga de upptar lite beräkningskraft för att exekveras. Detta gör det möjligt att exekvera ett stort antal agenter på samma dator. Själva DIET-plattformen är enligt [DIET Agents] globalt skalbar då den är implementerad på så vis att den inte innebär några begränsningar på storleken av distribuerade DIET-applikationer. Detta har uppnåtts genom att anta en helt decentraliserad arkitektur för konceptet.

Beteendestöd

Beteendestöd var inte premierat i utvecklingen av DIET och det finns heller inget naturligt stöd för detta. Det hindrar naturligtvis inte utvecklaren från att själv implementera beteende i agenterna och med hjälp av andra existerande beteendemodeller, men hur komplicerat det är att implementera och integrera några sådana i DIET är hittills okänt för författarna till denna rapport.

Agenttyper

DIET stödjer inte direkt olika typer av agenter, därmed inte alls sagt att agenterna inte kan vara olika. Agenterna som utvecklas i DIET är inte naturligt mobila och stödjer alltså inte denna typ av agenter. De agenter som utvecklas med hjälp av DIET bör dessutom vara lättviktsdesignade (vilket är positivt), men är inte avsedda att ha så mycket beteende. Detta gör att DIET inte lämpar sig för syftet som systemet utvärderas för i denna rapport.

¹ ”Icke-minneskrävande” är [DIET Agents] eget uttryck. Hur mycket detta stämmer i verkligheten är osäkert.

Senaste Release och Systemkrav

Den senaste utgåvan är Diet Agents version 0.93 som kom den 7 Jan 2004. Systemkraven är okända.

4.5.2 JADE

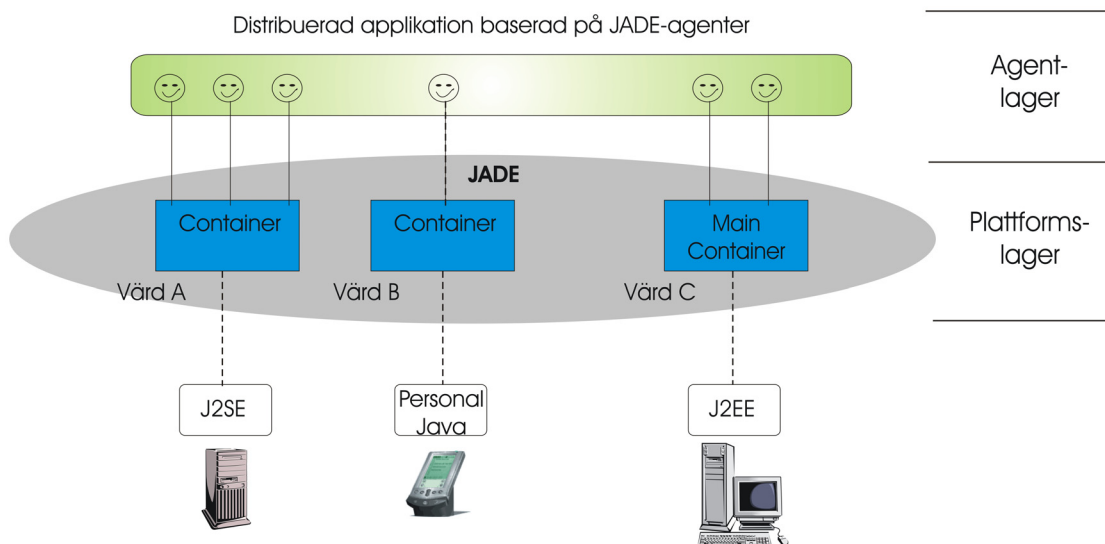
JADE är förkortningen för *Java Agent DEvelopment framework* och är ett mjukvaruramverk som används för att utveckla multiagent-system [JADE]. Ramverket initierades ursprungligen som ett *Open Source* projekt, vilket alltså har utvecklats öppet med öppen källkod mellan ett antal organisationer och andra frivilliga deltagare. Syftet med JADE är att underlätta utvecklingen av standardiserade multiagent-system genom ett mellanlager (en plattform) som följer FIPA-specifikationerna, samt tillhandahåller ett antal grafiska verktyg som stödjer debugging och driftsättning av agentsystem i olika faser av processen. Med andra ord stödjer JADE enligt dem själva utvecklingen av Peer-to-Peer-system, där noderna utgörs av intelligenta agenter och således kan reagera och agera autonomt på händelser i systemet. Agentplattformen som JADE består av kan spridas över ett antal noder distribuerat och noderna behöver inte innehålla samma operativsystem. JADE kan ses som ett agentmellanlager som implementerar en så kallade *Agent Platform* och ett utvecklingsramverk. Mellanlagret tar hand om alla tjänster som inte är applikationsspecifika, det vill säga meddelandehantering, sökning och publicering av tjänster, agentens livscykel och så vidare.

JADE är helt implementerat i Java och agentutvecklingen baseras på att det exekveras endast en Java-applikation (en JVM) på varje värddator. Agentplattformen tillhandahåller många funktioner, bl.a. ett GUI för att på distans kunna styra hantering och kontroll av status för agenterna m.m. Detta tillåter till exempel en användare att på distans starta eller stoppa om agenter. Med hjälp av tillägsbibliotek och samarbeten stödjer JADE många olika typer av klienter (Java-klienter), allt från webbklienter till vanliga PC och mobila klienter.

Arkitektur

JADE baseras i grunden på en JADE-specifik körtidsmiljö (*Runtime Environment*, RTE). Varje JADE-nod exekverar en instans av RTEn och varje instans kallas *container*. I en *container* kan JADE-agenter vara aktiva (upp till kring 1000 st.) och får härmed tillgång det stöd som JADE-miljön tillhandahåller. En samling aktiva containrar benämns plattform. För varje plattform behöver det finnas en huvud-*container*, s.k. *Main Container*, som alla nya containrar registrerar sig hos så fort de blir aktiva. Detta hindrar inte att flera JADE-plattformar existerar inom samma nätverk. Huvud-containern fungerar som en vanlig container, men tillhandahåller extra funktionalitet i form av agenthantering och en sök/register-funktion för tjänster som agenter i plattformen kan tillhandahålla.

JADE-konceptet kan på detta sätt sägas bestå av två lager, ett plattforms-lager och ett agent-lager. I Figur 4-2 ses en JADE-plattform som består av tre containrar. Containrarna kan i sig residera på olika plattformar beroende på vilket programmeringsgränssnitt (API) och vilken Java virtuella maskin (JVM) som väljs, från vanliga PC till handdatorer och mobiltelefoner. Agentlagret utgörs här av en distribuerad applikation som är baserad på JADE-agenter. Plattforms-lagret består av containrarna som är värd åt agenterna och som bidrar med specifik JADE-funktionalitet etc. Containrarna exekverar i sin tur på varsin värdmaskin (dator etc.) och i varsin JVM.



Figur 4-2: Arkitektur för JADE: två lager, ett agentlager och ett plattformslager

Kommunikation

All agentkommunikation i JADE sker genom meddelanden, där det språk som används är FIPA ACL¹ för att representera meddelandena. JADE tillhandahåller här en struktur för att hantera meddelanden effektivt, genom till exempel inköer på plats hos agenterna (alltså agentindividuell meddelandemottagning och inte på systemnivå). Meddelanden i kön kan därefter hämtas in i agenten med flera olika metoder. Till exempel finns synkrona och asynkrona metoder, sortering eller osorterade meddelanden och så vidare, vilka kräver olika mycket kontroll och *overhead* från systemet och agenten. På detta sätt kan agenten hämta meddelanden med den metod som passar agenten bäst för tillfället och i vilken ordning den önskar. Hela FIPAs kommunikationsmodell har implementerats för syftet. Den transportmekanism som används för meddelanden är adaptiv och anpassar sig till rådande situation då den transparent för användaren väljer det bästa tillgängliga protokollet. För tillfället används till exempel Java RMI och http, men fler protokoll kan läggas till genom att använda inbyggda hjälpfunktioner.

Meddelanden skickas till en destination, som i de vanligaste fallen identifieras med ett namn (agenter identifieras via namn). Men om sändaren inte vet mottagarens namn kan andra parametrar användas, till exempel kan meddelandet skickas till "alla som är intresserade av flygplansposition" eller liknande.

För kommunikation och meddelandehåll används till exempel Javas inbyggda strukturer och XML och RDF².

Säkerhet

Säkerhetsfunktionerna i JADE är implementerade som en s.k. *add-on*, det vill säga en tilläggbar komponent. Detta för att göra säkerhetsnivån valbar. Säkerhetsfunktioner bidrar oftast med en *overhead*, som kanske inte alla applikationer är beredda att betala. Som exempel på JADEs säkerhet tillhandahåller ramverket mekanismer för att verifiera och autentisera rättigheter som tilldelas agenter. Det här gör att agenterna kan kontrollera var meddelanden kommer ifrån och också bli nekade att skicka meddelanden till vissa agenter. Säkerheten i JADE hittills existerar endast på applikationsnivå, och alltså inte på nätverksnivå (såsom kommunikation, inträng i kod etc.).

¹ ACL står för *Agent Communication Language* och är ett FIPA-standardiserat språk för agenter.

² RDF – *Resource Description Framework*, är ett ramverk för att beskriva Internet-resurser med hjälp av t.ex. XML. RDF-dokumentet utgör således metadata för resursen.

Robusthet

JADE tillhandahåller funktioner för att migrera en agent till en nod och fortsätta exekvera agenten där istället, från den tidpunkt och med den status agenten hade när den stoppades. Denna funktion är dock inte helt automatisk men kan utnyttjas för att implementera robusthet i systemet, samt för att användas till lastbalansering. Det som måste tilläggas är funktioner i systemet eller i agenterna som upptäcker störningar eller obalans i last för att agenten ska kunna flyttas automatiskt då fel upptäcks.

Skalbarhet

Förutom att systemet är distribuerat (vilket medför möjligheter till skalbarhet) så har systemet ett antal inbyggda mekanismer för att öka denna parameter. Till exempel kan flera agentuppgifter hanteras parallellt inom samma Java-tråd. Detta gör att om antalet användare i systemet ökar, kan agenterna själva ta emot ett större antal uppgifter som löses snabbare, men det kräver naturligtvis att det finns beräkningsresurser som stödjer detta.

Beteendestöd

JADEs körtidsmiljö tillhandahåller några enklare beteenden för de mest vanliga agentuppgifterna såsom *FIPA interaction protocols*. Men det finns även grunder för andra beteendemönster som att de ska vakna under vissa förhållanden, färdiga strukturer (s.k. skelett) för uppgifter, och också möjligheter för att strukturera komplexa uppgifter som aggregeringar av enklare uppgifter etc. JADE tillhandahåller också så kallat *JessBehaviour*, som tillåter full integrering med JESS, där JADE är skalet i agenten och som garanterar att den följer FIPA, medan JESS¹ är motorn i agenten som genomför allt nödvändigt resonemang.

Agenttyper

Det finns ett antal färdiga agenter med GUI:n som går att använda till utvecklingen av agentsystem i JADE. *The Dummy Agent* är simpel men används för att inspektera och kontrollera meddelandeutbyte mellan agenter. *The Sniffer Agent* tillåter att meddelanden inom JADE spåras, till exempel kan kommunikation spåras för en enda agent eller en hel agentgrupp. *The Introspector Agent* tillåter att övervaka och kontrollera livscykeln hos en exekverande agent och härmed även dess meddelanden.

Senaste Release och Systemkrav

Senaste releasen är JADE 3.1 som kom den 17:e December 2003. Systemkraven är minst Java 1.4 men annars är det OS-oberoende.

4.5.3 Cougaar

Cougaar, *Cognitive Agent Architecture*, är en domänoberoende arkitektur för stora distribuerade agentsystem som designades och utvecklades för att tillfredställa de krav som ställs för att kunna hantera militär logistikplanering. Det har enligt skaparna även använts i andra domäner. Det utvecklades ursprungligen av DARPA-projektet ALP och därefter har efterföljaren, UltraLog (se Avsnitt 4.4.3), fortsatt att utveckla Cougaar med avseende på en högre nivå av skalbarhet, säkerhet och robusthet.

¹ JESS –Java System Expert Shell, <http://herzberg.ca.sandia.gov/jess/>

Typer av problem som Cougaar är lämpat för är t.ex.:

- Hierarkisk nedbrytning av komplexa uppgifter, så som t.ex. militär logistikplanering
- Integration av separata distribuerade applikationer och datakällor
- Generering av planer (ex. militär logistikplanering) i dynamiska situationer
- Applikationer som är för komplexa för att modelleras monolitiskt och som t.ex. utmärks av komponenter med föränderligt och dynamiskt beteende eller som är parallella till sin natur

Arkitektur

Förenklat är Cougaar en storskalig ”arbetsmotor” som är byggd på en komponentbaserad agentarkitektur. Agenterna kommunicerar med varandra via ett inbyggt meddelandeprotokoll och de samarbetar med varandra för att lösa gemensamma problem samt lagrar resultatet distribuerat emellan sig. De är uppbyggda av funktionella moduler som dynamiskt och kontinuerligt arbetar och omarbetar lösningar till problem allteftersom parametrar, restriktioner och annat förändras.

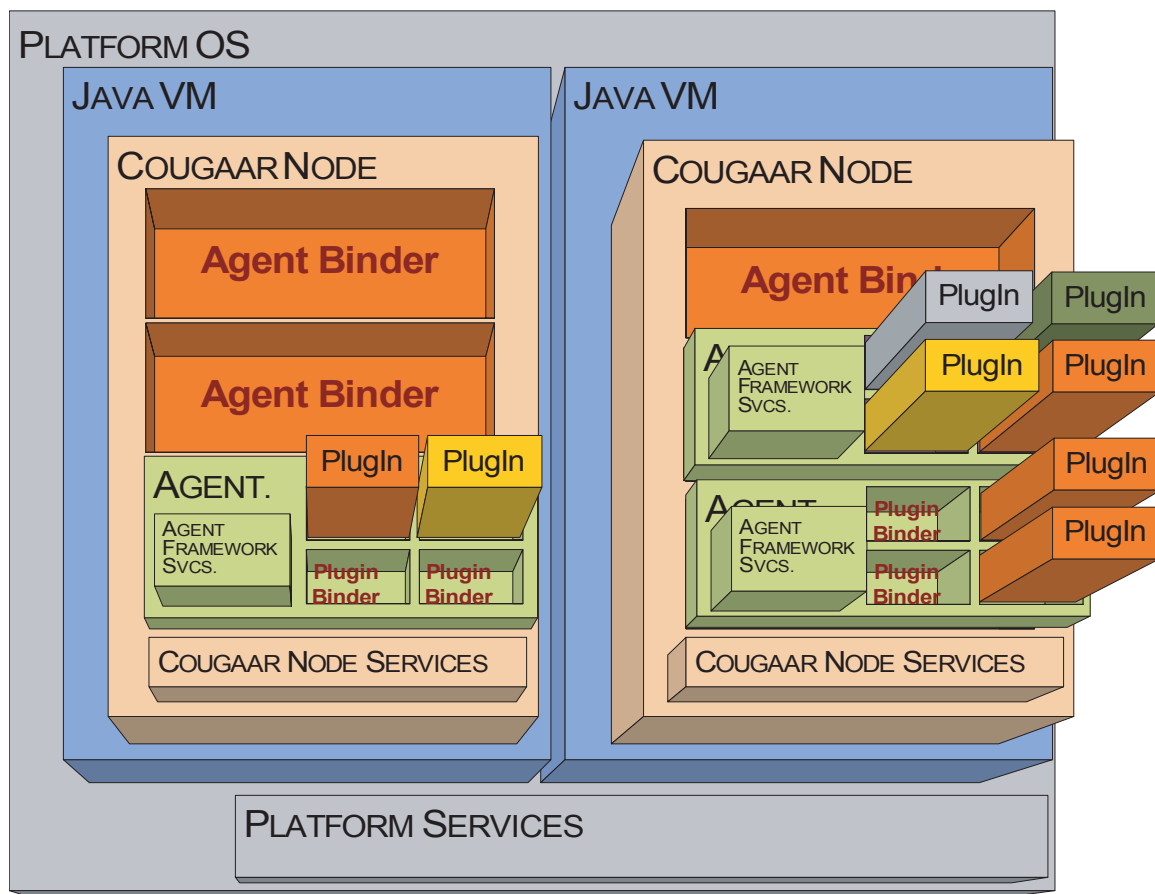
En Cougaar-agent är en agent som har fått ett visst beteende inprogrammerat för att modellera en viss organisation, algoritm eller process. En Cougaar-*Society* är en samling agenter som kollektivt interagerar i syfte att lösa ett visst problem eller vissa typer av problem. Problemen kan t.ex. vara relaterade till planering där målen och resurserna dynamiskt planeras om allteftersom förutsättningarna förändras. En Cougaar-*Community* är ett abstrakt begrepp på en grupp agenter som har någon slags gemensam funktionalitet eller organisatorisk tillhörighet. En Cougaar-*society* kan alltså utgöras av en eller flera logiska *communities*, där några agenter tillhör en eller flera *communities* och andra agenter inte är relaterade till någon alls.

En *society* delar en DNS¹-liknande *namespace* som tillåter alla agenter att lösa referenser till varandra och den kan vara monolitisk eller distribuerad. Alla agenter i en *society* har samma Cougaar kodkärna i grunden men de kan även ha tilläggsmoduler som bestämmer deras beteende samt ger dem vissa rättigheter och förmågor.

En Cougaar-*Node* är en enda instans av en JVM, *Java Virtual Machine*, som i sin tur kan innehålla och underhålla en mängd agenter. För det mesta är förhållandet mellan en *Node* och en hårdvaru-plattform, d.v.s. en dator, ett till ett. Grupperingen av agenterna i en nod är inte nödvändigtvis domänrelaterad utan snarare resursrelaterad. Ofta befinner sig agenter på samma nod för att de är intresserade av en gemensam resurs som t.ex. en databas eller något annat. Alla agenter på en och samma nod delar på processorkraft, minnesutrymme och bandbredd.

En agent består i princip av två större delar, en partitionerad och distribuerad *Blackboard*, eller ”anslagstavla”, samt *Plug-ins* d.v.s. komponenter som styr beteende, logik och funktionalitet. Komponenterna fungerar genom att de publicerar information på agentens ”anslagstavla” och prenumererar på objekt som finns där. Innehållet på ”anslagstavlan” delas upp i domäner med logiskt sammanhängande objekt. Varje agent är ansvarig för att hålla reda på sina meddelanden som den fått från andra agenter samt att schemalägga exekveringen av olika *Plug-ins*. Se Figur 4-3 för en schematisk bild på hur en Cougaar-plattform kan se ut samt Figur 4-4 för en bild på en agents ingående delar.

¹ DNS –Domain Name System. Ett sätt att hålla reda på vilka domäner vissa datorer hör till. För mer information om DNS läs t.ex. <http://susnet.se/homWebDNS.html>.



Figur 4-3: En schematisk bild av hur Cougar-plattformen kan se ut [Cougar-Bildref]

Den grundläggande tanken hos Cougar-arkitekturen är att den skall vara komponentbaserad i så stor utsträckning som möjligt. Komplexa problem bryts ner i mindre och mer hanterbara delar. Dessa delar sätts sedan samman i större konstellationer och det resulterande beteendet skall vara summan av de ingående delarna. Aggregering och disaggregering är alltså ett genomgående tema i Cougar-arkitekturen

Kommunikation

Kommunikationen i Cougar är asynkron¹. För de flesta tillämpningarna behöver inte applikationsutvecklaren bry sig om hur meddelanden mellan agenterna fungerar. Kommunikationen sköts för det mesta av de inbyggda logikkomponenterna.

Meddelanden till agenter på andra noder passerar ett meddelandelager som skickar meddelandet genom nätverket till den mottagande noden som sedan i sin tur skickar meddelandet vidare till rätt agent. Kommunikation mellan agenter i en och samma nod går, med hjälp av nodens inbyggda funktionalitet, direkt igenom den lokala JVM:en.

För att hantera kommunikation mellan agenter på olika noder och över flera typer av nätverksprotokoll har Cougar stöd för distribuerade namngivningstjänster, så kallade *naming services*. Dessa tjänster kan även användas av applikationsutvecklare för att dynamiskt upptäcka agenterna under körning.

¹ Asynkron kommunikation är sådan som inte sker samtidigt, den är oberoende av tid och rum. Den kan t.ex. jämföras med att skicka ett e-mail eller ett SMS –när mottagaren får tid att läsa meddelandet gör han det, inte tidigare.

De namngivningstjänster som Cougaar-agenter har stöd för är "*Name Generation*", "*White Pages*", "*Yellow pages*", "*Local discovery*" och "*Peer-to-Peer search*". Alla agenter i Cougaar måste ha ett unikt namn då det används för att skicka meddelanden till agenten.

- *Name Generation* konstruerar ett globalt unikt agentnamn för varje ny agent som skapas. Det kan t.ex. göras med hjälp av en enkel slupnamnsgenerator
- *White Pages* är en distribuerad tabell som mappar agentnamn till nätverksadresser. Den är modellerad efter DNS-tankarna och stödjer numera även separatorstecknet '.' som används i Internets domännamn. Vilket gör att namnrymden får en mer hierarkisk topologi, men den blir mer hanterbar för stora (över 200 agenter) distribuerade agentapplikationer. Den primära funktionen hos *White Pages* är att stödja Cougaars meddelandetransport och andra nätverksberoende funktioner
- *Yellow Pages* är en tjänst som liknar den vanliga telefonkatalogens. Med hjälp av den går det att slå upp agenter som tillhandahåller en viss funktionalitet eller tjänst
- *Local Discovery* gör det möjligt för en agent att upptäcka andra LAN-lokala agenter utan att känna till deras namn eller IP-adresser (denna funktionalitet är ännu ej helt klar)
- *Peer-to-Peer Search* tillåter en agent att upptäcka resurser, så som t.ex. filer, som finns hos andra nära agenter

Kommunikationslagret är som det mesta annat i Cougaar-arkitekturen en utbytbar komponent. Om utvecklaren vill utöka funktionaliteten eller byta ut mot något annat protokoll går det att göra.

Säkerhet

Inom UltraLog-projektet har man satsat mycket på att utöka och säkerhetsställa säkerhetsfunktionerna. En pusselbit i detta strävande är att i så stor utsträckning som möjligt dölja och kapsla in information. I Cougaar-arkitekturen är det tänkt att en given komponent ska ha tillgång till all den information den behöver för att kunna utföra sin uppgift men inte mer än så. Alla komponenter är specificerade så att de ska ha ett så rent gränssnitt mot andra komponenter som möjligt, de ska alltså vara så modulära som möjligt. Det underlättar bl.a. utbyggbarhet och aggregering av komponenter men även bättre informationssäkerhet fås då information döljs i komponenterna.

Agenter kan inte se information och objekt som finns internt i "anslagstavlan" hos andra agenter. För att få den informationen och få reda på vilken funktionalitet en annan agent har, måste de använda standardiserade gränssnitt. *Plug-ins* kan inte se tillstånden hos andra *Plug-ins*. *Communities* har rena gränssnitt mot varandra och kan inte se varandras interna konfiguration av agenter.

En Cougaar-applikation skall kunna motstå en mängd elektroniska attacker, kunna behålla integriteten hos informationen och undvika att röja kommunikation. All kommunikation som går mellan agenter internt i en nod är säker för tjuvlyssning, intrång, ändring eller sabotage. Vidare är Cougaar-infrastrukturen, *Plug-ins* och konfigureringsinformationen designad så att det ska gå att bekräfta om något slags intrång har skett.

Robusthet

Cougaar-applikationer är designade för att vara långlivade, d.v.s. exekvera kontinuerligt 24 timmar om dagen, 7 dagar i veckan under ett helt år. Därför är en annan av UltraLogs målsättningar att utöka med funktionalitetskomponenter så att en Cougaar-applikation kan överleva förlusten av individuella komponenter eller hårdvara (exekveringsplattformar) med endast minimal funktionalitetsförlust. Detta inkluderar bl.a. att automatiskt kunna återställa förlorade agenter vid t.ex. strömavbrott, hårdvarufel eller oavsiktlig omstart av datorer. Framförallt åstadkoms detta

genom att tillstånd hos publicerade objekt samt prenumeranterna på dessa objekt, tillståndet på kommunikationen mellan agenter sparas på icke förgängligt media t.ex. i filer samt i lokala eller avlägsna databaser. När en agent då startas om kan den för det mesta fortsätta från det senast sparade tillståndet. Andra agenter i en *society* klarar av att agenter tillfälligt är frånvarande p.g.a. t.ex. nätverkverksbortfall eller fel på agent. När agenten sedan vill återansluta sig till samma *society* går detta bra. Konfigurationen hos agenter, *community* och *society* kan förändras utan att det har negativ inverkan på andra komponenter, vilket ger både dynamisk rekonfiguration och evolution av funktionalitet.

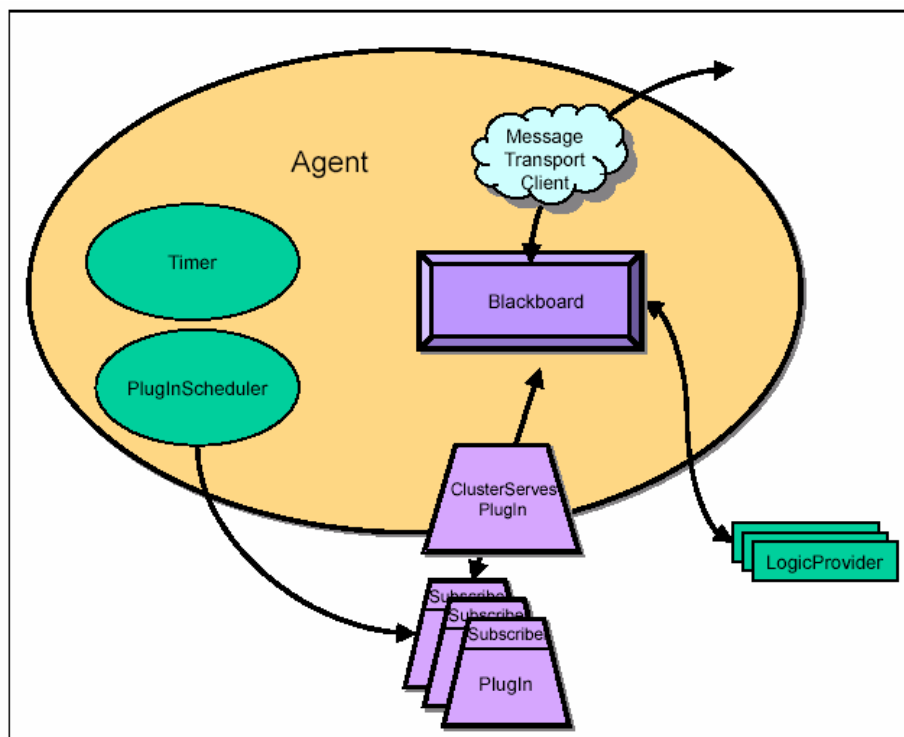
Skalbarhet

Den komponentbaserade arkitekturen gör att Cougaar inte bör ha några direkta skalbarhetsproblem. Det ska vara möjligt att utveckla Cougaar-applikationer som är så skalbara som applikationslogiken tillåter. Det är designat för att klara stora applikationer (ca 1000 agenter som kör kontinuerligt under ett helt år) och *Plug-in*-arkitekturen gör att det är lättare att bygga och underhålla stora mjukvarusystem än vanliga traditionella arkitekturer.

Under utvecklingen av Cougaar har man lärt sig flera läxor om skalbarhet, bl.a. visade sig kommunikation och namngivningstjänster vara svårast att få skalbara. Genom att bl.a. införa P2P-kommunikation mellan agenter har det problemet lösts och på detta sätt har exponentiell tillväxt i interaktioner och beroenden mellan agenter undvikits.

Beteendestöd

Beteendet hos en Cougaar-agent bestäms av dels av de *Plug-in* som agenten har och de logikdelar som finns kopplade till "anslagstavlan". Det finns bra dokumentation och skelett för att utveckla egna *Plug-ins*. I Cougaar-arkitekturen finns det även färdigt ett antal nödvändiga tjänster och funktionalitet såsom agentidentifieringsmekanismer, hantering av objekt och meddelanden, namngivningstjänster, loggningstjänster o.s.v. Se Figur 4-4 för en bild på en Cougaar-agent och dess ingående delar.



Figur 4-4: En Cougaar-agent och dess ingående delar [Cougar-Bildref 2]

Agenttyper

Cougaar-agenter är i sig helt generiska. Det finns inga fördefinierade agenttyper. För det mesta kommer alla agenter i en och samma *society* ha samma kodbas och vara instanser av samma klass men det är upp till utvecklaren att bestämma hur agenterna ska delas upp.

Systemkrav och Senaste Release

Cougaar är OS-Oberoende. Den senaste versionen är Cougaar 11.0.0 och den kom den 24 februari 2004.

Övrigt

Mer detaljerad information om hur Cougaar-arkitekturen fungerar kan läsas i [CAD] och det finns även dokumentation för den som vill göra en egen implementation [CDD]. På den officiella hemsidan [Cougaar] går det att ladda ner en *tutorial* för den som vill lära sig mer. Det finns även ett antal *open source*-grupper som har projekt för att vidareutveckla olika delar i Cougaar. För den som är intresserad är det fritt fram att gå med i någon av dessa utvecklargrupper. Ett antal artiklar har skrivits om Cougaar och de finns också att läsa på hemsidan. Under de senaste åren har det även anordnats ett par *Workshops* i USA för *Open Source Cougaar communityn*. Annan Cougaar-relaterad information kan även fås via UltraLogs hemsida [UltraLog].

4.5.4 Jämförelse mellan systemen – Diskussion och slutsatser

En analytisk jämförelse mellan de tre potentiella agentsystemen genomfördes. Som grund för jämförelsen låg dels de parametrar som avhandlades i respektive systemgenomgång, samt dels de krav som identifierades i Avsnitt 2.2 och de förutsättningar som scenarion i Avsnitt 2.1 ansågs ställa. Slutsatser för ramverken presenteras kort i prioritetsordning, där det agentramverk som anses ha bäst potential för syftet diskuteras sist. En sammanställning av slutsatser och resultat finns i Tabell 4-1. Gemensamt för alla tre ramverken är de grundförutsättningar som ställdes vid urvalet; de är *open source* och implementerade i Java.

DIET

DIET är ett mycket skalbart system för stora distribuerade applikationer, vilket är av stor betydelse för framtida försvarssystem. Däremot stödjer det inte någon allmänt vedertagen standard för kommunikation, det stödjer inte naturligt mobila agenter och inte heller avancerade beteendemodeller eller kommunikationsprotokoll. Detta gör sammantaget att DIET-plattformen inte lämpar sig för användning i ett mellanlager av den typ som avses i denna rapport, då dessa egenskaper behövs i ett distribuerat logistiksystem.

JADE

JADE är ett moget agentramverk som utvecklas ständigt. Främsta fördelarna inkluderar mekanismer för säkerhet och robusthet, samt medföljande modeller för beteende och beskrivning av komplexa uppgifter. Dessutom följer JADE FIPA-standard. Inga påtagliga nackdelar har ännu stötts på, men robustheten (felhanteringen och felupptäckter etc.) i systemet skulle kunna förbättras och automatiseras. Allt detta gör ramverket väl lämpligt för att utvärderas vidare för användning i ett mellanlager för logistik.

Cougaar

Cougaar är ett stort och komplext agentramverk med mycket potential. Det är utvecklat för den militära logistikplaneringsdomänen och kommer att användas i olika projekt i bl.a. USA. Framförallt ett av projekten, UltraLog, kommer att testa Cougaars gränser i sin stora demonstration som är planerad till slutet av 2004. Många av fördelarna med Cougaar är att det har utvecklats för att klara stora applikationer, vara säkert, skalbart och robust. Det är generiskt

och ett stort *open source community* som stödjer den fortsatta utvecklingen finns. Det är ett agentramverk som verkar mycket lämpligt för en fortsatt utvärdering för användning i ett mellanlager för logistik.

	DIET	JADE	COUGAAR
Mobilitet	- Stödjer ej migrering av kod	+ Helt mobil	-/+ Stödjer delvis migrering av kod
Kommunikation	- Stödjer inte så många protokoll - Ej gemensamt språk - Ej gemensam infrastruktur	+ Adaptiv transportmekanism + FIPAs hela kommunikationsmodell + XML => standardiserad	+ Stödjer flertalet protokoll + ”Enkelt” utbyggbart
Säkerhet	- Ej behandlad	+ Mekanismer för autentisering, verifiering etc. + Implementerade som en <i>add-on</i>	+ Väl utvecklad + Information döljs i komponenter
Robusthet	- Inbyggd feldetektion + Automatisk flytt och återställning	- Ej behandlad -/+ Stödjer full migrering inkl. status => robusthet möjlig	+ Hög nivå av robusthet
Skalbarhet	+ Stor + Mycket lightweight	+ Stor + Inbyggda mekanismer för effektiv trådning	+ Stor + Särskilt kommunikation och namngivningstjänster har utarbetats för att vara mer skalbara
Beteendestöd	- Ej inbyggd infrastruktur eller stöd	+ Tillhandahåller enkla färdiga strukturer + Följer FIPA	+ Tillhandahåller stöd + Beteende implementeras komponentbaserat (som <i>plug-in</i>)
Kommentar	+ Trevligt, lätthanterig agentplattform. - Dock lite omoget, kan bli intressant senare.	+ Mycket bra förutsättningar för syftet. + Följer FIPA-standard!	+ Mycket bra förutsättningar för syftet. - Ej rent mobilt + Implementerad för militära logistiksyften + Används i existerande logistikprojekt och – tillämpningar

Tabell 4-1: Jämförelse mellan tre potentiella agentsystem

5 Diskussion och förslag till arkitektur

I detta kapitel presenteras ett förslag till en arkitektur för mellanlagret. Förslaget har utformats med logistikdomänen som fokus, men det kan även lämpa sig generellt för informationshantering och integration av system som ska finnas tillgängliga inom det framtida svenska försvarsnätverket. Logistikområdet sätter mycket höga krav på informationshantering och -tillgång, samt de möjligheter och den flexibilitet som mellanlagret är avsett att erbjuda. Därför är det just i detta område som behovet av ett sådant mellanlager har blivit mest kritiskt. Det kan därför också vara lämpligt att i ett första steg använda detta område som referensdomän för utformningen, då detta sätter extra höga krav och ramar för arkitekturen.

5.1 Användarfall

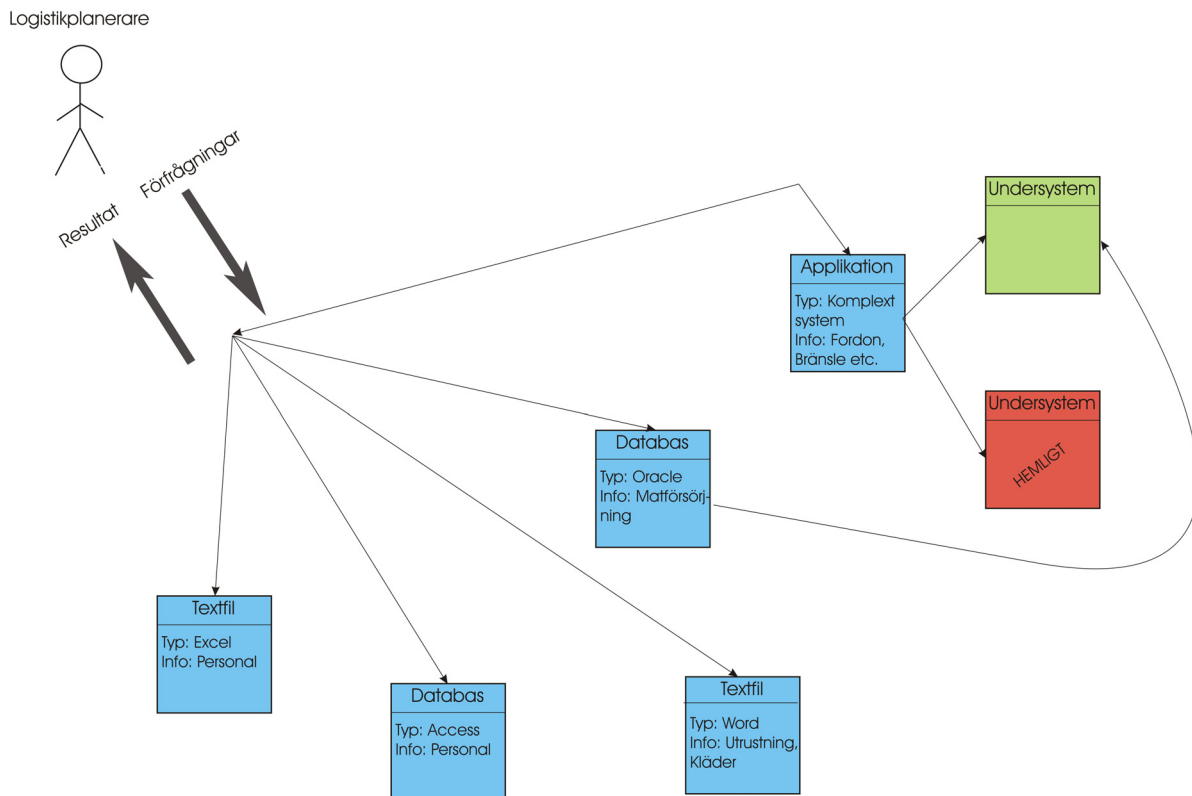
För att analysera den funktionalitet och de krav som arkitekturen måste svara mot, sattes ett enkelt användarfall upp. Detta baseras på Scenario 1 -*Internationell Operation*, som beskrivs i Kapitel 2. Användningsfallet avser en planerare inom Logistikledning, som i hemlandet ska ta reda på om en internationell insats är möjlig, samt i sådana fall planera för den. Känt är storleken på den insatsstyrka som efterfrågas (antal personal samt specifika användarprofiler), men inte varifrån den behövda personalen kan inhämtas, vilken personal som är tillgänglig, samt vilken och hur mycket utrustning som krävs etc.

Logistikplaneraren behöver samla in, jämföra och fusionera information från ett stort antal heterogena resurser. Efter beräkning behöver planeraren även allokera resurserna. Exempel på några av de resurser som utnyttjas visas i Figur 5-1, såsom databaser med personalinformation, system som hanterar och beräknar ammunitionsförbrukning etc. Samma typ av information (exempelvis Personal) kan dessutom existera i olika typer av källor (såsom databaser, Excel-ark etc.) och finnas på olika format.

Informationen måste samlas in och kunna presenteras på ett bra sätt för planeraren. Presentationen ska kunna varieras utifrån olika *användarprofiler*, d.v.s. beroende på hårdvara och mjukvara (t.ex. PC, handdator, webbläsare, Windows, Unix o.s.v.) och personprofil (ämnesområdesexpert, systemutvecklare, militär, civil o.s.v.). Kommunikationen bakom det som presenteras, d.v.s. logiken samt kopplingen ut i nätverket och till de olika resurserna, ska kunna vara olika mycket transparent beroende på personprofil. Det är även tänkbart att en viss typ av användare själv vill ange och välja en specifik resurs. Tvärtom kanske en annan användare själv inte innehar den kompetens som krävs, eller tid som behövs, för att göra ett sådant val eller ta ett visst beslut, utan istället ställer en förfrågan till systemet som autonomt bidrar med beslutsstöd till klienten.

Den typ av information som behövs och de parametrar som beräknas är t.ex.:

- Personal – De eftersökta personalprofilerna jämförs med tillgänglig personal och den tid personalen med rätt kompetens finns tillgänglig
- Utrustning – Utrustning för insatsen planeras utifrån de dynamiska omständigheterna, längden på uppdraget kanske ännu ej är känt
- Mat – Matresurser planeras gentemot vad som kan införskaffas på plats
- Fordon inklusive utrustning och bränsle – De fordon som behöver införskaffas inplaneras. Om de inte finns tillgängliga ersätts de av motsvarande fordon för syftet



Figur 5-1: En användare behöver extrahera information som underlag för logistisk planering från ett antal heterogena källor såsom databaser, olika system etc.

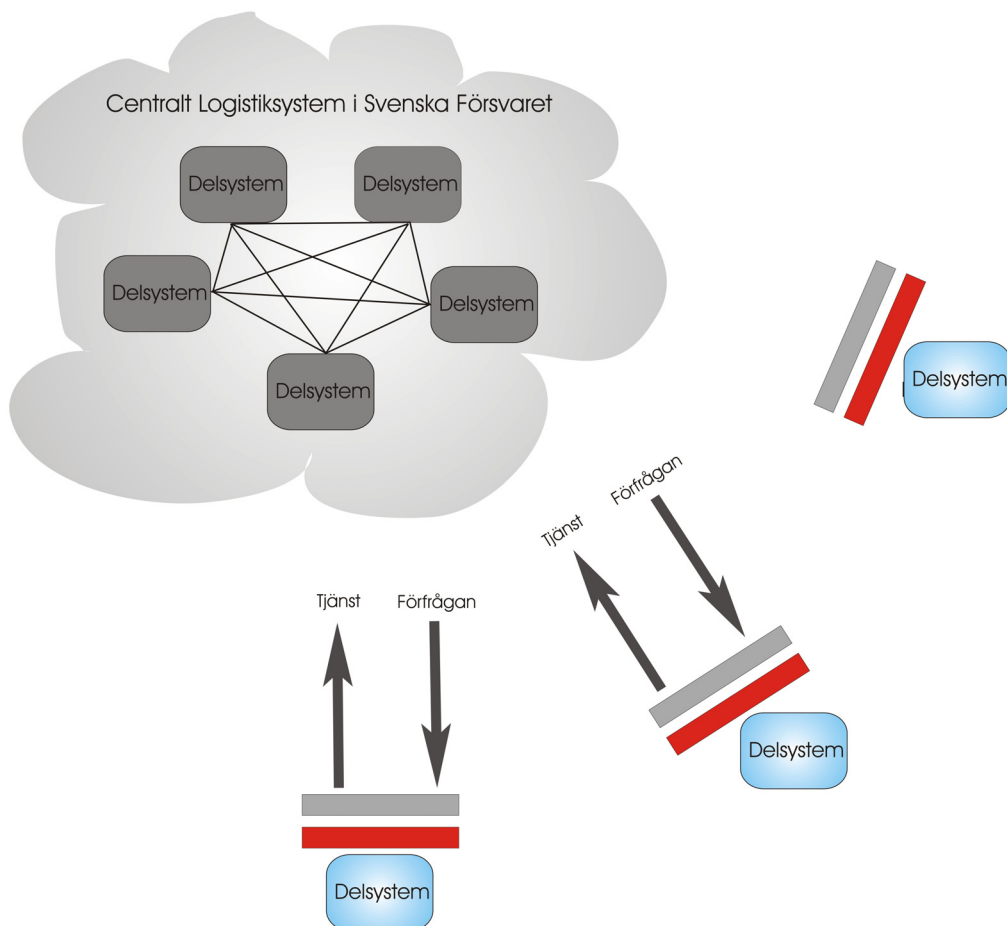
5.2 Övergripande – Modulär tjänstarkitektur med generisk informationsmodell

Mellanlagret är tänkt att följa tjänstekonceptet som även har presenterats i arbetet med NBF. Det vill säga, de system och tillgångar som finns inom nätverket och som är tillgängliga för logistikplanering och andra logistiksyften är utformade som tjänster. Detta gör att det på ett effektivt och modulärt sätt går att lägga till och dra ifrån nya tjänster till systemet utan att infrastrukturen ändras. Dessutom gör det att tjänsterna kan tillgås även utanför logistikdomänen och sålunda ökar möjlig återanvändning av systemen avsevärt.

På detta sätt kan framtidens logistiksystem ses som ett centralt logistiksystem, som i sig är distribuerat. Det centrala systemet tillhandahåller försvarsgemensamma logistikresurser och den infrastruktur som krävs för att integrera och erbjuda logistiksystem och –resurser. I ett första steg består systemet av existerande logistiksystem inom försvaret, med eventuell koppling ut i externa nätverk såsom Internet. Nya och framtida delsystem och domäner kan läggas till som nya tjänster som på ett flexibelt integreras in i systemet. Samtliga nya komponenter (delnätverk, resurser etc.) integreras som en tjänst med hjälp av en *wrapper* som infrastrukturen bidrar med, se Figur 5-2.

Wrappern består av två delar; dels ett gränssnitt in mot det system som ska integreras (rött i figur) och dels ett lager som svarar mot tjänstestrukturen och ansvarar för att kommunikation (grått i figur, kommunikation innebär här data och förfrågningar etc.) ut från och in i systemet hanteras på ett enligt infrastrukturen godkänt och uniformt sätt. Den informationsmodell som utnyttjas för detta syfte måste utarbetas i fortsatt arbete. Det är omöjligt att utforma en sådan modell som automatiskt passar alla nya och tillkommande system. Det smidigaste är därför att utforma en så pass generell modell som möjligt, en delvis generisk informationsmodell, med tillhörande ontologier för domänen, som går att utöka och förändra utan att infrastrukturen havererar.

Det kommer antagligen att behöva göras olika ontologier för olika domäner, men det är tänkbart att ha en struktur med en toppontologi och under den de domänspecifika ontologierna. Oavsett vilken lösning man väljer kommer en inventering av gemensamma begrepp för FM att behöva göras. Frågorna som togs upp i Avsnitt 2.3.4 som har att göra med informationshantering kommer tillsammans med andra att behöva besvaras.



Figur 5-2: Modular tjänstearkitektur för framtida logistiksystem

En klient i sammanhanget är den person som utnyttjar tjänsterna i systemet. För användaren är det transparent om tjänsten som erbjuds residerar inom det försvarsgemensamma nätverket eller externt. Användaren kan till exempel vara en logistikplanerare som utnyttjar tjänsterna och dessa kan i sig utnyttja information från varandra (se Figur 5-1 tidigare).

5.2.1 Logik – Vad, Var och Hur?

Logik måste byggas in i systemet (mellanlagret) för ett antal uppgifter. Dels måste format på data någonstans bli gemensamt. Dels kan data som representerar samma resurstyp (t.ex. personal) se olika ut. En representation av resursen Personal i ett delsystem kanske beskriver endast tre parametrar och utgår från personnummer i identifieringen, medan i ett annat kanske beskriver åtta parametrar, dessutom mer utförligt, och utgår från ett militärt id-nummer. Logiken i systemet måste kunna hantera dessa problemställningar.

Logiken i mellanlagret måste även kunna hantera den typ av förfrågningar som klienten ställer. Klienten ska inte själv behöva ta hänsyn till vilket system den ställer förfrågan till (detta är kanske inte möjligt att genomföra till 100%). Systemet ansvarar sedan för att förfrågningarna översätts så att de kan tolkas av de heterogena system de ska ställas till (ex.vis databaser), samt bidrar med logik för att utföra förfrågan i delsystemet om inget inbyggt stöd för detta finns (som t.ex. i en textfil).

Designfrågan som vi ställs inför är: Hur mycket logik ska byggas in i delsystemen versus hur mycket logik ska systemet (mellanlagret) bistå med? Kapslas logiken in i delsystemen så blir dels anpassningen av befintliga delsystem till logistiksystemet större och dels blir logiken mindre flexibel och inte lika lätt att återanvända. Implementeras däremot logiken som en komponent i mellanlagret kan den dels återanvändas, dels varieras utan att man behöver gå in i varje delsystem som är beroende av den logiken. Detta är dock mer komplext att implementera. Logiken kan även implementeras hos själva klienten, men det innebär samma nackdelar som om den implementeras i systemen samt att lösningen belastar klienten, vilket inte är önskvärt.

Det som rekommenderas här är att i den mån det är realistiskt implementera komponentbaserad logik i systemet, men där det är för komplext bygga in logiken i det system som ska integreras.

5.3 Möjliga lösningar

Under förstudien har vi tittat på olika tekniker och metoder för att kunna hitta möjliga lösningar till utveckling av mellanlagret. Vi har valt ut fem möjliga tillvägagångssätt med dagens tekniker, som vi anser är lämpliga och som svarar mot kraven, utan att ta hänsyn till lösningens lämplighet. Först diskuteras de olika lösningarna och därefter utvärderas de mot varandra för att kunna välja den mest lämpliga.

5.3.1 Egenutveckling

En möjlig lösning är att implementera mellanlagret själva och från början. Kommunikationsprotokoll och nätverkslösningar samt inbyggd logik implementeras direkt utifrån de krav och förutsättningar som ställs. Denna lösning blir skräddarsydd då all programvara kan anpassas helt till de underliggande logistiksystemen och total kontroll skulle has över all programvara. De underliggande systemen skulle inte heller behöva anpassas nämnvärt eftersom inkapslingen av dessa sker unikt för varje system. Denna lösning är dock inte så flexibel och realistisk, dels innebär en egen lösning på detta sätt antagligen en mycket stor arbetsinsats och dels riskerar man att låsa sig mot nuvarande system. Genom att inte följa några erkända standarder riskerar man även att äventyra interoperabiliteten mot externa system, samt återanvändning av komponenter och wrappers för systemen reduceras.

5.3.2 Web Services

Web services tillhandahåller standardiserade protokoll och tekniker som är erkända och stöds av industrin. Med hjälp av *web services* kan logistiksystem på ett flexibelt och återanvändbart sätt göras åtkomliga via webben. Det är säkert på så sätt att endast de tjänster som på förhand bestämts ska publiceras är åtkomliga. Om WS används som teknik genomförs först en tjänsteanalys för varje logistiksystem och en s.k. *wrapper* utformas utefter denna.

Nackdelarna med det här alternativet är att *web services* generellt är tillståndslösa och statiska, d.v.s. de förändras t.ex. inte med efterfrågan eller kan komma ihåg eller kombinera tidigare svar. Även om vissa lösningar för detta existerar, och det pågår arbete inom WS-området för att hantera dessa problem, så är de begränsade och det finns ännu ej några etablerade lösningar för detta.

Med WS finns även en risk för att mycket bandbredd utnyttjas då svaren på förfrågningar strömmas över nätverket och ofta filtreras på klientsidan. Det finns inte heller inbyggt någon naturlig garanti för att en förfrågan till en WS verkligen tas emot och bearbetas. Vidare kommer en webbtjänst inte ihåg om en förfrågan har gjorts flera gånger tidigare. Tjänsterna som systemen kan tillhandahålla är dessutom begränsade till de tjänster som publiceras och inte mer, men detta kan i säkerhetssammanhang ses som en fördel, då en WS endast tar emot väl definierad input och på specifika portar.

5.3.3 *Agenter*

Agenter har intelligens i form av inbyggd logik, vilket gör att de på ett mer dynamiskt sätt kan anpassa sig till nya förhållanden. De har minne och kan komma ihåg uppgifter de tidigare har utfört eller förfrågningar de har ställt. Då agenterna kan skräddarsys för varje uppgift går denna lösning att göra modulär. Dynamisk nätverksuppkoppling kan även hanteras av agenterna utan nämnvärda problem då de har minne och logik. Det vill säga även om en klient är nerkopplad kan agenten fortsätta att behandla klientens förfrågan, för att returnera resultatet när klienten kopplar upp sig igen.

Nackdelen med att bara ha agenter är att det inte finns någon erkänd standard för tekniken. Detta innebär att den lösning som utvecklas blir dels agentspecifik, dels reduceras återanvändningen av implementerad logik och wrappers etc. Dessutom innebär det bl.a. att det inte finns någon standardiserad säkerhetslösning. Varje agentramverk har löst frågan på olika sätt.

Att implementera hela mellanlagret med hjälp av agenter är en intressant lösning med tanke på den komponentbaserade flexibla infrastruktur tekniken erbjuder. Det existerar flera etablerade agentarkitekturer med sofistikerade lösningar för säkerhet och bandbreddsoptimering, och tekniken används i flera system i drift bl.a. för logistiksyften. Dock kan mobila agenter utgöra en viss overhead för nätverket, jämfört med exempelvis WS, om t.ex. enkla förfrågningar eller uppgifter ska utföras, då agenten förflyttar hela sig (programmet) över nätverket. Om bara icke-mobila agenter utnyttjas belastas bandbredden lika mycket som för webbtjänster.

5.3.4 *Kombination av Web Services och Agenter*

Agenter och *web services* är egentligen inte två motsatta tekniker, de ansvarar snarare för olika funktioner och kan med fördel kombineras (se Tabell 5-1 och jämför egenskaperna för Agenter och WS). De nackdelar den ena tekniken har, kompletteras av den andra (se Tabell 5-1 igen). Alla fördelar som respektive teknik erbjuder behålls dessutom vid en kombination av de båda. Kombinationen gör även att en bättre anpassning till olika uppgifter kan fås d.v.s. det blir mer flexibelt än att bara använda WS eller agenter. En annan fördel som fås är att kombinationen blir mer robust. Nackdelen är att det inte finns någon etablerad lösning för denna kombination. Men lösningen används av andra för logistiksyften och andra syften, och förslaget är positivt ur forskningsperspektiv och fördelaktigt med tanke på framtida utveckling av försvarets nätverk och struktur.

5.3.5 *Mobila Web Services*

Mobila Web Services är ett koncept som till viss del fortfarande ligger på forskningsstadiet. Det går i stort ut på att dra tjänstekonceptet till sin spets. I grunden använder man sig av agenter som inte bara kommunicerar med hjälp av WS-protokoll utan även deras funktionalitet och de tjänster de kan tillhandahålla publiceras med hjälp av WS. Fördelarna här är tydliga, ett standardiserat gränssnitt kan fås till alla komponenter samtidigt som agentlogiken ytterligare göms.

Lösningen blir mer transparent och alla fördelar som kombinationen WS och Agenter har fås även här. Trots att detta koncept fortfarande är outvecklat och ganska omoget verkar det vara mycket lovande. Det är antagligen inte alltför svårt att börja med en kombination av WS och agenter och sedan allteftersom konceptet mognar gå över mot en lösning med mobila *web services*. Därför rekommenderas att man avvaktar med att förlita sig till detta koncept.

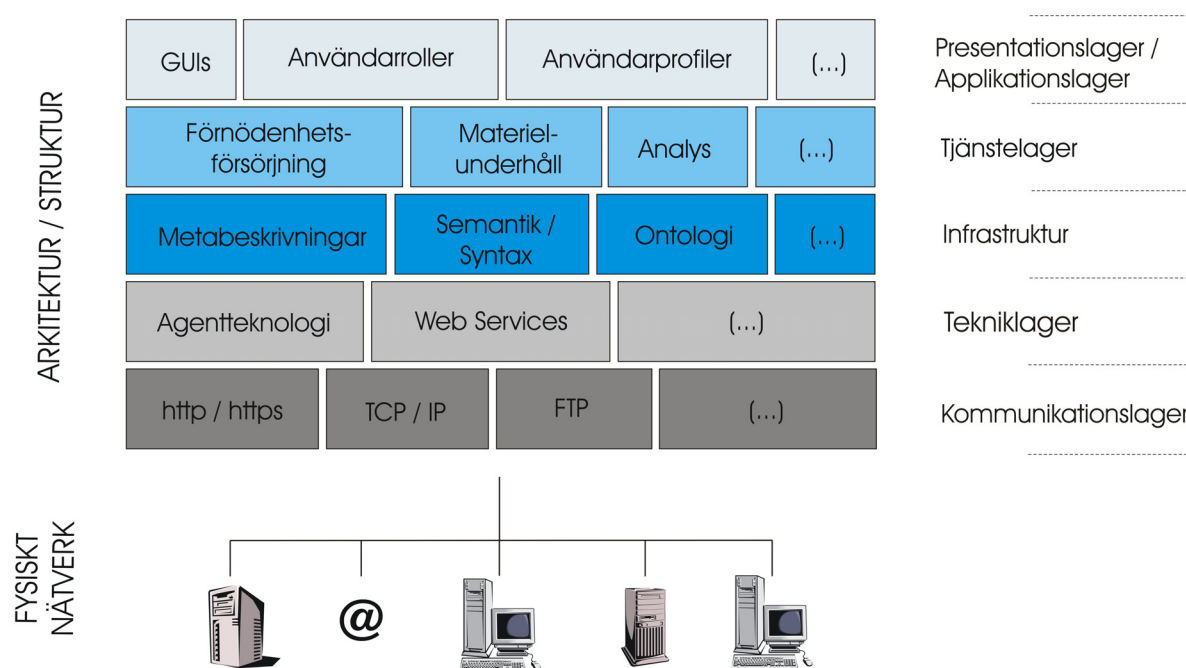
5.3.6 Sammanfattning och utvärdering

Ur de olika möjliga lösningar som vi har föreslagit har en sammanfattning på tabellform gjorts. När alternativen utvärderades gjordes detta teoretiskt och med användarfallet i Avsnitt 5.1 som bas. Teoretiskt utnyttjades de olika alternativen som lösning till den arkitektur som efterfrågas, och för varje lösning analyserades de möjliga fördelar och potentiella begränsningar för just den lösningen och utifrån syftet. Resultatet av analysen och utvärderingen presenteras i Tabell 5-1 på nästa sida.

5.4 Arkitekturförslag

Som ett resultat av genomgången av potentiella tekniker och metoder, analysen av krav och förutsättningar som gäller för det specifika området, samt djupdykningen i området agentsystem presenterar vi i detta avsnitt ett förslag till mellanlagret. Vårt förslag till hur en arkitektur ska utformas och implementeras baseras på en kombination av agenter (mobila såväl som stationära) och *Web Services*. Det agentramverk som rekommenderas är Cougaar.

Arkitekturen utvecklas som modulär. Detta för att öka möjligheterna för att lägga till funktionalitet i form av informationsmodeller, nya domäner, möjliga användarprofiler, nätverksprotokoll etc. i framtiden (se Figur 5-3). Information och kommunikation i arkitekturen baseras delvis på XML (inte nödvändigtvis agenterna emellan). XML används bl.a. för informationsmodellen som utgör gränssnitt mot logistiksystemen. Den ontologi som utnyttjas för modellen kan på detta sätt bytas ut i framtida utveckling utan att infrastrukturen påverkas. Dessutom kan fler modeller läggas till och vidareutvecklas utan att delsystemen behöver modifieras.



Figur 5-3: Översiktsbild av föreslagen modulär arkitektur för mellanlager samt ingående komponenter

	Egen utveckling	Bara WS	Bara Agenter	WS och Agenter	Mobila Web Services
Fördelar	+Total kontroll över programvaran +Kan anpassa programvaran helt till problemet	+Standardiserade protokoll +Stöds av industrin +Åtkomligt via Webben +Säkert	+Har logik d.v.s. intelligens +Minne +Modulärt +Dynamiskt +Hanterar dynamisk nätverks-uppkoppling +Existerar flera etablerade arkitekturer	+Alla fördelar från WS och Agenter +Bättre anpassning till uppgiften d.v.s. mer flexibelt (än bara WS eller bara Agenter) +Mer robust än bara WS	+Alla fördelar från WS och Agenter +Alla komponenter får ett standardiserat gränssnitt +All agentlogik göms och man får en mer transparent lösning
Nackdelar	-Följer ingen erkänd standard vilket kan försvåra interoperabilitet med andra system -Betydligt större arbetsinsats krävs	-Inget tillstånd sparas -Inget minne -Statiskt -Tar mycket bandbredd -Begränsad till publicerad tjänst	-Inga erkända standarder och gränssnitt -Ingen standardiserad säkerhetslösning -Om endast lite information behövs ⇒ stor overhead Icke-mobila agenter -Bandbredd -Mycket logik hos klienten	-Det finns ingen etablerad lösning	
Övrigt				*Används som teknik av andra för logistiksyften	*På forskningsstadiet *Tar tjänstebegreppet till sin spets

Tabell 5-1: Sammanställning av fördelar och nackdelar hos de arkitekturlösningar som diskuteras i Avsnitt 5.3

5.4.1 *Agenters funktionalitet i mellanlagret*

Agenterna i mellanlagret utnyttjas för att hantera förfrågningar och på ett autonomt sätt förse klienten med beslutsstöd och annan underlättande funktionalitet i systemet. En kombination av stationära och mobila agenter används som kan kommunicera med varandra. De stationära agenterna kan exempelvis residera hos en klient för att stödja klienten med rätt typ av presentation och information, medan en mobil agent sköter informationshanteringen i nätverket och mellan olika system. För att besvara en förfrågan kan den mobila agenten på ett effektivt sätt behöva kopiera sig själv och skicka kopiorna till olika källor. När agenterna har hittat det de söker samlas de, fusionerar och selekterar information för att tillsammans besvara förfrågan, innan slutligen en agent presenterar resultatet för klienten. En agent kan dessutom besvara en förfrågan trots dynamisk nätverksuppkoppling. När klienten inte är uppkopplad fortsätter agenten att besvara förfrågan. När klienten kopplar upp sig nästa gång får klienten resultatet.

Det kan tänkas finnas olika typer av agenter i sammanhanget för olika användare och olika uppdrag:

- Inhämtning/sökning av information (mobila agenter)
- Reservera resurser - Det kan behöva finnas agenter som specifikt reserverar (allokerar) resurser för klientens räkning (kan t.ex. utgöras av stationära agenter som endast existerar så länge resursen allokeras)
- Fusionera information från resurser (kräver alltså en viss logik och autonomitet(beslutsförmåga))
- Profilagenter (agenter som representerar och agerar utifrån en användares specifika profil)
- Administrativa agenter (Som har reda på om källor och resurser fortfarande fungerar, kanske håller reda på trovärdheten hos dessa osv.)

Agenter möjliggör även för en naturlig koppling till informationskällor på Internet. Agenterna kan på så vis hämta information från webben utan att de resurser som eftersöks nödvändigtvis är representerade av en webbtjänst. Agenten kan till exempel hämta hem en hel HTML- eller XML-sida till klienten för att gå igenom och tolka sidan själv, och besvara förfrågan som klienten hade.

5.4.2 *Webbtjänster i mellanlagret*

Webbtjänsterna används för att kapsla in (utgöra gränssnitt mot) relaterade logistiksystem och – resurser och publiceras som tjänster. Dessa kan sedan användas dels i det avsedda systemet, men även av klienter och framtida användare utanför detta.

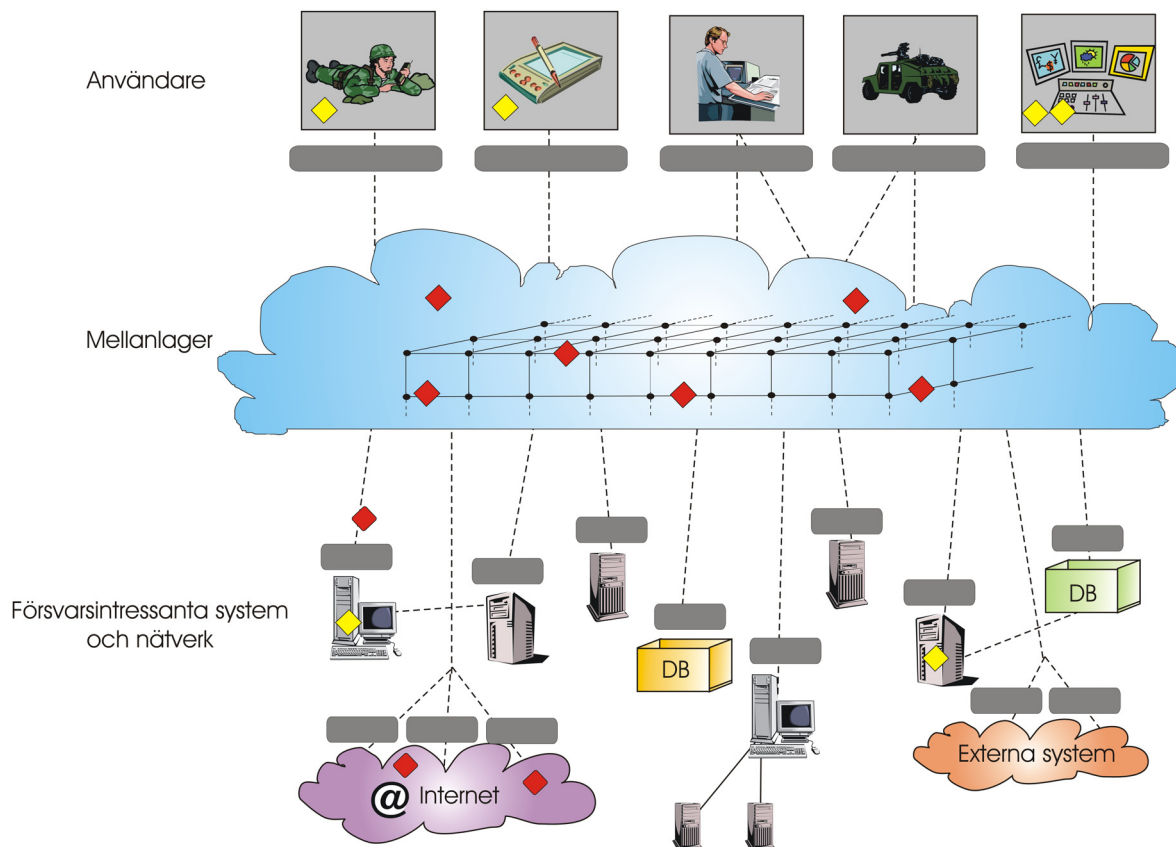
Man kan även tänka sig att viss logik, exempelvis transformeringsalgoritmer, beräkningslogik etc. finns implementerade som webbtjänster i nätverket. Om flera system behöver utnyttja samma funktioner slipper de på detta sätt ha logiken implementerad hos sig, och istället återanvänder de webbtjänsten för samma beräkningar.

5.4.3 *Integrering av Web Services och agenter*

Agenterna kan kommunicera med varandra och transporteras i nätverket. Men för att kunna komma in i en nod (resurs, databas, klient etc.) måste denna ha ett gränssnitt som hanterar emottagande av en agent. Gränssnittet kallas agentserver och kan finnas till exempel integrerad i webbtjänsten som kapslar in i noden (dvs i wrappern), eller bakom applikationen som en klient använder (i Figur 5-4 ses de wrappers – grå gränssnitt – som tar emot de mobila agenterna – röda objekten).

Agentservern implementeras med fördel som generisk, så att den kan ta emot alla olika typer av agenter, vilket gör att den ser likadan ut för varje nod. Detta medför att inget av delsystemen behöver modifieras om till exempel en ny agenttyp tillkommer.

Hos webbtjänsten kan även en statisk agent (icke-mobil) residera för att stödja tjänsten med klient- eller resursspecifik information (representeras av de gula objekten i Figur 5-4).



Figur 5-4: Figuren demonstrerar hur mellanlagret tillhandahåller sömlös kommunikation mellan användare och system (jämför Figur 1-2). I figuren demonstreras wrappers till tjänster (de grå objekten) samt hur agenterna platsar i infrastrukturen. Röda komponenter representerar mobila agenter och gula representerar stationära

5.4.4 Krav för systemet

För att designa och implementera mellanlagret är det viktigt att en kravanalys först genomförs. Kraven kommer att gälla dels systemet i helhet (inklusive webbtjänster och agentsystem samt kommunikationen i nätverket) och dels även de olika noderna (såsom klienter, databaser och andra resurser). I analysen av användarfallet identifierades ett antal sådana krav varav några exempel är:

- För agenter: en agentserver som tar emot och kan skicka iväg agenter måste finnas integrerad i varje nod (i varje gemensamt gränssnitt)
- Varje nod måste tillhandahålla ett gemensamt gränssnitt (datapubliceringslager)
- Unika nycklar måste representera resurserna (dessa kan utgöras av t.ex. personnummer, chassinummer, reg.nummer, id-nummer etc.)
- Gemensam representation av resurser (gemensam inom logistikdomänen eller med fördel försvarsgemensam)

5.4.5 Säkerhetspolicy

Vid utvecklingen av mellanlagret bör en säkerhetspolicy först utarbetas, som i första hand avser logistiksystemen, men som även går i linje med en eventuell policy för det framtida nätverksbaserade svenska försvaret. Precis som NAI [NAI Labs] utvecklar en säkerhetspolicy för Cougaar och UltraLog för användning inom det amerikanska försvaret, behöver en policy för konceptet WS och agenter tas fram och verifieras för det svenska försvaret och för logistikdomänen som avses. En sådan säkerhetspolicy deklarerar hantering av alla parametrar för säkerhetsmekanismer [NAI Labs]. Parametrarna inkluderar till exempel:

- Hur och när data ska krypteras
- Vilka attacker som kan förväntas och hur de ska hanteras
- Vilken beredsskapsstatus systemet har
- Vad som utgör en säker åtgärd/hållning/transport (för agenten)
- Hur identifiering av dataklassificering ska utföras

Utöver policyn behöver man även fastställa potentiella hot och risker och förbereda / anpassa konceptet för att klara dessa o.s.v.

6 Framtida arbete

En naturlig fortsättning på förstudien är att omsätta teori i praktik och ta tillvara på de rekommendationer som utarbetades i arbetet. Följande förslag på fortsatt arbete bygger på de arkitekturlösningar som beskrivs i Kapitel 5. Förslagen går i stort ut på att designa och implementera en prototyp som demonstrerar dels den eftersökta funktionaliteten, dels en komplett processkedja.

6.1 Utveckling av en prototyp för mellanlagret

En prototyp för mellanlagret utformas och utvecklas, där arkitekturen som implementeras kombinerar tjänstekonceptet med autonom och, för användaren, transparent informationshantering. Förslaget följer rekommendationer från rapporten och integrerar således agentteknik med konceptet *Web Services*. Enligt arkitekturförslaget förses logistiksystemen med ett *Web Service*-gränssnitt, en s.k. *wrapper*, så att de kan på ett plattformsoberoende sätt tillhandahålla de tjänster och information som eftersöks. På detta sätt kan dessutom *wrappern* till det behandlade systemet återanvändas i fortsatt arbete även om agenttekniken byts ut mot en annan. Som en första ansats räcker det att använda simulerade men representativa system, istället för riktiga.

För att hantera informationsflödet mellan system och användare utnyttjas agentteknik, där ramverket Cougaar förslagsvis används. Agenter utvecklas som är specifika för syftet och för att kunna tillhandahålla information och förfrågningar mellan de olika systemen och olika typer av användare. Logiken som utvecklas för agenterna designas i ett första steg så att de kan extrahera information från de underliggande logistiksystemen, fusionera och bearbeta den för olika ändamål. Mer avancerade uppgifter och strategier för agenterna kan läggas till efter hand. Olika typer av användare identifieras och information anpassas beroende på vem som gör förfrågan. Viss informationsmodellering kommer att krävas för att skapa en återanvändbar bas för framtida utveckling av mellanlagret, oberoende av vilka tekniker som väljs i utvecklingen.

De frågeställningar vi tycker behöver behandlas är till exempel:

- Verifiering av konceptet med agenter och *Web Services*: Det vill säga ett *Proof-of-concept*
- Utarbetning av riktlinjer för en mer långsiktig och generisk implementation av informationshantering i ett nätverksbaserat försvar
- Erfarenhet av begränsningar och möjligheter, samt den arbetsinsats (och ev. anpassning av befintliga ramverk) som krävs för att implementera agentlogik för syftet
- Besvara frågor som -Hur lätt eller svårt är det att kapsla in och byta ut de underliggande logistiksystemen?
- Hur pass mycket information i systemet går att anpassa till olika typer av användare och hur måste information modelleras för att vara användbar

Exempel på ingående arbetsmoment:

- Sätta sig in i och utveckla i valt agentramverk (för detta ges Cougaar som förslag)
- Implementera specifik logik som sedan används i agenterna (dvs. utgör autonomiteten och beslutsstödet i systemet)
- Välja olika typer av databaser för att simulera verkliga logistiksystem, samt lägga in information och implementera funktionalitet i dessa
- Utveckla WS-gränssnitt för varje databas

- Designa ett användarfall som senare utnyttjas för demonstration av den processkedja som avses
- Identifiera hur användaranpassad information till och från användare ser ut, samt anpassa informationen som presenteras för dessa efter kraven
- Utveckla en mindre ontologi för domänen som sedan används av agenterna
- Utveckla en demonstrator
- Testning och experiment av konceptet samt demonstratorn
- Skriva rapport och dokumentation
- Slutdemonstration

6.1.1 *Exempel på utökningar*

Förslaget beskriver genomförandet av en enkel prototyp så som den skulle kunna implementeras på den lägsta nivån. De logistiksystem som används är ickereella, men fullt dugliga för att använda endast till en prototyp. Den logik som implementeras i agenterna demonstrerar funktionaliteten, men möjligtvis är den inte så avancerad eller återanvändbar och så vidare. Förslaget skulle därför med fördel kunna utökas. Om riktiga logistiksystem såsom DIDAS eller FREJ används kan en bättre utvärdering av konceptet göras utifrån ett antal parametrar, till exempel erhålls en bättre bild av hur komplext det är att kapsla in redan existerande system i agenter och *Web Services*. Nedan presenteras fem förslag till utökningar, som är oberoende av varandra.

1. En korrekt ontologi skulle kunna uppföras som även kan återanvändas för domänen efter arbetet, oavsett om konceptet återanvänds eller ej
2. Extra tid och arbete skulle kunna avsättas för att göra verktygen och arkitekturen mer generiska, modulära och utbyggbara för att möjliggöra återanvändning på ett bättre sätt
3. Två agentramverk skulle kunna användas till samma uppgift för att på en bättre grund dels utvärdera dessa (mot varandra), samt dels kunna utvärdera eftersökt funktionalitet för syftet och identifiera problemställningar i sammanhanget bättre
4. Samarbete med industrin eller utlandet skulle kunna initieras för att höja kvalitén i utvecklingen
5. En korrekt utvärdering av parametrar såsom säkerhet, skalbarhet och robusthet skulle också kunna göras

En utökning enligt punkterna ovan skulle ge prototypen och resultatet en annan karaktär än det första förslaget, då de är baserade på riktiga data, verklig information, ontologi samt riktiga användare. Man skulle få en djupare analys och utvärdering av konceptet och av agentramverken och lättare kunna plocka det bästa ur varje ramverk. Man skulle få ett bättre test av parametrar såsom säkerhet, robusthet och skalbarhet. De utvecklade verktygen och delar i systemet skulle kunna återanvändas lättare även om prototypen som helhet inte skulle återanvändas. Flaskhalsar i systemet och informationsflöden skulle även till viss del kunna identifieras. Valda tekniker skulle valideras utifrån resultatet och konceptet skulle kunna valideras mer korrekt.

Ytterligare exempel på ingående arbetsmoment:

- Designa generisk arkitektur
- Utveckla återanvändbara WS-gränssnitt för underliggande logistiksystem

- Utveckla en ontologi för en verklig deldomän i logistik som sedan används av agenterna
- Implementera verktyg och funktionalitetskomponenter som sedan används av agenterna
- Identifiera de eventuella brister som behöver åtgärdas i konceptet och hur
- Jämföra agentramverken för att se vilket som bäst uppfyller kraven på en mer detaljerad nivå
- Testning av parametrar som säkerhet, skalbarhet och robusthet
- Testning av hur lätt respektive svårt det är att byta användare samt undersystem
- Sätta upp en modell av systemet och simulera det utifrån givna förutsättningar och tänkta framtida scenarios => gör det möjligt att identifiera flaskhalsar i systemet, informationsflöden samt validera valda tekniker utifrån resultatet

6.2 Egenutveckling av agentramverk

Ett alternativ som skiljer sig från det tidigare nämnda är att implementera en prototyp såsom beskrivet i första förslaget, men att samtidigt utveckla ett helt eget agentramverk som används till syftet. Detta skulle föra med sig att man skulle få ett agentramverk som var särskilt avpassat för uppdraget och problemdomänen. Dessutom skulle det utgöra en bättre utgångspunkt för att i ett senare skede kunna bedöma och utvärdera befintliga agentramverk för anpassning till syftet och den funktionalitet som de erbjuder. I arbetet skulle särskilt viktiga problemställningar enklare kunna identifieras än om ett existerande agentramverk utnyttjas och utvecklarna skulle få full kontroll och kännedom om funktionaliteten, komplexiteten samt även eventuella fel och brister i ramverket (vilka annars kan vara delvis dolda för utvecklaren). Trots fördelarna innebär detta alternativ en betydligt större arbetsinsats än föregående förslag, varför detta måste tas i beaktande om egenutveckling av ett agentramverk skall genomföras.

Referenser

- [AgDUS] Ag Kartläggning Drift- och underhållssystem. *Rapport AgDUS Slutrapport*. FMV ProjDU 14621:54553
- [AgentSpace] AgentSpace hemsida: <http://www.cs.strath.ac.uk/~if/agentspace/>, senast besökt Mars 2004
- [Aglets] IBM Aglets hemsida: <http://www.trl.ibm.com/aglets/>, senast besökt Mars 2004
- [Anderson et al.] R. Anderson, P. Feldman, S. Gerwehr, B. Houghton, R. Mesic, Securing the U.S. Defense Information Infrastructure: A Proposed Approach. Santa Monica, California, 1999. ISBN: 0833027131
- [Baker et al.] M.Baker, R.Buyya, D.Laforenza, *Grid and Grid technologies for wide-area distributed computing*. Software – Practice and Experience, 2002, John Wiley & Sons Ltd
- [Bond] Bond agents hemsida: <http://bond.cs.ucf.edu/>, senast besökt April 2004
- [CAG] *Cougaar Architecture Guide*, http://cougaar.org/docman/?group_id=17, senast besökt Juni 2004
- [CCS] Warwick Ford, *Computer Communications Security -Principles, Standard Protocols and Techniques*. Prentice Hall PTR. ISBN 0-13-799453-2
- [CDG] *Cougaar Developers' Guide*. http://cougaar.org/docman/?group_id=17, senast besökt Juni 2004
- [Cougaar] Cougaar hemsida: <http://www.cougaar.org>, senast besökt Juni 2004
- [Cougaar bildref] Bild från <http://www.ultralog.net/briefings.html>, med tillstånd av Dr. M. Greaves
- [Cougaar bildref 2] Bild från referens [CAG], med tillstånd av Dr. M. Greaves
- [D'Agents] D'Agents hemsida: <http://agent.cs.dartmouth.edu/>, senast besökt Juni 2004
- [DAML] DARPA Agent Markup Language hemsida: <http://www.daml.org/>, senast besökt Juli 2004
- [DIET Agents] DIET Agents hemsida: <http://diet-agents.sourceforge.net/>, senast besökt Maj 2004
- [DIK] Stefan Karlsson, TorBjörn Nilsson, *Översättning av data till information – en diskussion med hjälp av den infologiska ekvationen*. Avdelningen för industriell organisation, Luleå tekniska universitet www.ies.luth.se/org/Rapporter/ar2003_57.pdf, senast besökt Juni 2004
- [Ekström] Thomas Ekström, *Supply Chain Management –En evolution inom den civila logistiken, men en revolution inom den militära logistiken?* FOI Memo 777, Februari 2004

- [FIPA] FIPAs hemsida: <http://www.fipa.org/>, senast besökt Juli 2004
- [Foster et al.] I. Foster, C. Kesselman, J. Nick, S. Tuecke, *The Physiology of the Grid*. Open Grid Service Infrastructure WG, Global Grid Forum, Juni 2002
- [Gong] L.Gong, *JXTA: A Network Programming Environment*. Industry Report, Sun Microsystems Inc. Juni 2001. http://www.jxta.org/project/www/white_papers.html, senast besökt Juli 2004
- [Grasshopper] Grasshopper2 hemsida: <http://213.160.69.23/grasshopper-website/>, senast besökt Juni 2004
- [GTI] Magnus Nelénus, Per Winroth. *Gemensam tillgänglig information* Whitepaper, Saab AT, IPT 2002-10-21 Version 1.0
- [Hive] Hive hemsida: <http://hive.sourceforge.net/mkgray-thesis/html/node7.html>, senast besökt Juli 2004
- [IML] Martin Eklöf, *Informationshantering & -modellering till stöd för logistisk M&S –en förstudie*. Metodrapport FOI. December 2003. FOI-R—1128--SE
- [JADE] JADE hemsida: <http://jade.cselt.it/>, senast besökt Juli 2004
- [Janssen et al. 1] W.Janssen, T.Karygiaanis, *NIST Special Publication 800-19 - Mobile Agent Security*. National Institute of Standards and Technology, Augusti 1999
- [Janssen et al. 2] W.Janssen, *Countermeasures for Mobile Agent Security*. National Institute of Standards and Technology, November 2000
- [JXTA] JXTA-projektets huvudhemsida: <http://www.jxta.org>, senast besökt Februari 2004
- [LOGLED] Modelleringsrapport från LOGLED-projektet, *Modelleringsrapport – Projekt LOGLED Processmodelleringsseminarier samt kvalitetssäkring Maj 2004*. FMV Maj -04
- [MobileSpaces] MobileSpaces hemsida: <http://research.nii.ac.jp/~ichiro/mobilespaces/en/index.html>, senast besökt Juli 2004
- [MokSAF] MokSAF hemsida: <http://www-2.cs.cmu.edu/~softagents/moksaf.html>, senast besökt Juli 2004
- [Nagappan et al.] R.Nagappan, R. Skoczylas, R. Sriganesh, *Developing Java Web Services*. Wiley Publishing, 2003. ISBN: 0-471-23640-3
- [NAI Labs] NAI Labs Advanced Research, projektprofil för UltraLog: *Project Profile UltraLog* http://www.networkassociates.com/us/_tier0/nailabs/_media/research_projects/security_architecture/ultralog.pdf, senast besökt Juli 2004
- [NATO] NATO (1997), *Logistics handbook*, Bryssel; Senior NATO logisticians' conference secretariat NATO headquarters sid. 3-6
- [NBF] Huvudhemsidan för NBF: <http://www.mil.se/nbf/>, senast besökt Juni 2004

- [NOMADS] NOMADS-projektets hemsida: <http://nomads.coginst.uwf.edu/>, senast besökt Juli 2004
- [OWL] Web Ontology Language hemsida: <http://www.w3.org/2004/OWL/>, Senast besökt Juli 2004
- [Petrie] C. Petrie, *Agents versus WebServices*. PowerPoint-Presentation, November 2002. <http://snrc.stanford.edu/~petrie/fx-agents/xserv/oopsla/slide-7.html>
- [RDF] Resource description Framework hemsida: <http://www.w3.org/RDF/>, senast besökt Juli 2004
- [Rem.Nom] Försvarsmakten (1999-09-20) Remiss nomenklatur för försvarsmakten. Stockholm; Högkvarterns skrivelse 09 911:66532
- [RETSINA] RETSINA hemsida: http://www-2.cs.cmu.edu/~softagents/retsina_agent_arch.html, senast besökt Mars 2004
- [RIIB] Sofia Bacovic, Pernilla Svan, *Ett ramverk för integration och instansiering av begreppsmodeller*. Examensarbete DSV. Under tryckning Juni 2004
- [Satoh] I.Satoh, *A Framework for Building Adaptive Distributed Applications Using a Hierarchical Mobile Agent System*. Proceedings of the 2001 ACM Symposium on Document engineering, Atlanta, Georgia, USA, 2001
- [Seaway] CDM Technologies Inc. Hemsida, produktinformation Seaway. *Joint Decision-Support System for Sea Base Logistics Planning and Coordination*. http://www.cadrc.calpoly.edu/pdf/Seaway_Brochure.pdf, senast besökt Juli 2004
- [Suri et al.] N.Suri, J.Bradshaw, M.Breedy, P.Groth, G.Hill, R.Jeffers, T.Mitrovich, *An Overview of the NOMADS Mobile Agent System*. 2nd International Symposium on Agent Systems and Applications, ASA/MA2000, Zurich, Schweiz, September 2000
- [Tanenbaum et al.] A. Tanenbaum, M. Steen, *Distributed Systems – Principles and Paradigms*. Prentice Hall, 2002, ISBN: 0-13-088893-1
- [UltraLog] UltraLog hemsida: <http://www.ultralog.net>, senast besökt April 2004
- [UltraLog objs] UltraLog Objectives: <http://www.ultralog.net/objectives.html>, senast besökt Juni 2004
- [Zettergren] Mj Hans Zettergren, *Vad är Logistik?*, Beteckning 19 100:2019, C-uppsats 2002-06-04 FHS, http://bibliotek.fhs.mil.se/publikationer/uppsatser/2002/chp0002/zettergren_2019.pdf
- [VSHMOD] VSHMOD-UH –Informationsmodell med begreppsdefinitioner. CD-skiva med projektets olika rapporter från FMV
- [Weiss] G.Weiss, *Multiagent Systems – A Modern Approach to Distributed Artificial Intelligence*. MIT Press Cambridge England, 2000. ISBN: 0-262-23203-0
- [Williamson] H.Williamson, XML – The Complete Reference. Osborn/McGraw-Hill, Berkeley California USA, 2001. ISBN: 0-07-212734-1

[Wilson et al.]

L.F.Wilson, G.Cybenko, D.Burroughs, *Mobile Agents for Distributed Simulation..* Proceedings of the High Performance Computing Symposium (HPC'99), 1999

Appendix A –Logistik

Vad är logistik?

Logistik är ett svårdefinierat begrepp då det präglas av historien i olika länder. Logistikerna har stor betydelse för olika sorters militära insatser och speciellt i krig. Den fick sitt genombrott under det andra världskriget då framförallt USA uppmärksammade dess betydelse. Inom det svenska försvaret används det som ett samlingsbegrepp för ett antal delfunktioner. Enligt remissutgåva av nomenklatur [Rem.Nom] för Försvarsmakten definieras logistik på följande sätt; *Operativt samlingsbegrepp för den verksamhet som planeras, förbereds och genomförs för att trupp och förnödenheter skall finnas på rätt plats i rätt tid.* Detta innebär för det mesta ledning av operativ och strategisk underhållstjänst d.v.s. förnödenhetsförsörjning, materielunderhåll, kommunikationer och byggnader, sjukvård, förrådshantering och transporter samt den organisation som krävs för att genomföra militära operationer.

Inom NATO [NATO] har man en något vidare definition än i Sverige av ordet och funktionen logistik. Man har tio delfunktioner som skapar ett logistiskt stöd. Målsättningen inom Sverige är att en anpassning till NATO skall ske och att man därmed även skall bruka samma definitioner. De tio delfunktionerna i NATOs definition är:

- a) Materiel –produktion eller anskaffning av materiel för hela operationen. Detta innebär mottagning av material, distribution, lagring och underhåll.
- b) Försörjning –av alla typer av material och utrustning för att stödja de militära styrkorna.
- c) Underhåll och reparation –för att så fort som möjligt få materiel och utrustning i användbart skick.
- d) Service –denna funktion har med anskaffning av kvalificerad personal för att stödja de stridande trupperna. Anskaffning och distribution av kartor, posttjänst m.m. samt service som matsalar, tvättmöjlighet, badanläggningar, begravningsstjänst o.s.v.
- e) Ammunitionsröjning –är till för att lokalisera, märka ut, rapportera och oskadliggöra odetonerad ammunition.
- f) Transport och förflyttning –av förband och materiel.
- g) Förbindelse och anläggningar –en funktion för att stödja förbanden och övriga logistikfunktioner med att skapa och upprätthålla uppdragets transportvägar, byggnader och anläggningar.
- h) Sjukvård –tillhandahåller vård, omhändertagande och evakuering av sjuk, sårad och skadad personal.
- i) Avtal –alla avtal som behövs för att skaffa tillträde till operationsområde, skaffa underhåll och service samt annat som behövs för att operationen skall kunna genomföras.
- j) Ekonomi –varje nation finansierar generellt sin egen operation och ekonomifunktionen ser till att utföra operationen inom de ekonomiska ramar som finns.

För mer information om logistikens historia och skillnaderna mellan Sveriges, NATOs och Rysslands syn på logistiken se bl.a. [Ekström, Zettergren].

Exempel på existerande logistiksystem i FM

Inom AgDUS-projektet kartlades logistiksystemen inom det svenska försvaret. Det var framförallt system för materielunderhåll och förnödenhetsförsörjning. För mer information om projektet AgDUS se [AgDUS]. Författarna gjorde en snabbstudie av hur de största av dessa

system såg ut för att få en uppfattning om vilka sorts system det handlade om, hur de fungerar samt vad de är till för. De undersökta systemen var LIFT, DIDAS, FREJ och DELTA. Nedan följer en kort beskrivning, som är en något omarbetad version från AgDUS databas, av systemen. De slutsatser som författarna kunde dra ur detta material och samtal med deltagare i AgDUS-projektet finns i Kapitel 2.

LIFT

LIFT står för Lednings och Informationssystem för Förnödenhetsförsörjning samt Teknisk tjänst. Systemet stödjer och används av alla försvarsgrenar för hantering av materiel och teknisk tjänst samt för teknisk och viss ekonomisk uppföljning av materiel och förnödenheter. Huvudinriktningen är drift, underhåll samt grund- och förvaltningsdata. LIFT är uppdelat på två sorters delsystem, LIFT-Lokal och LIFT-Grund. LIFT-Grund är en central informationsbank för utveckling och distribution av normverk och systemtabeller. Dessa grunddata och mallar distribueras främst till lokala installationer av LIFT, vilka kallas LIFT-Lokal.

DIDAS

DIDAS FLYG och BAS, en förkortning av DriftDataSystem, är ett system som är framtaget för att stödja materielunderhållstjänsten avseende främst flygmateriel, såväl flygande som markbunden, men även annan basmateriel med större behov av underhållsuppföljning.

Systemet är ett verktyg för att säkra luftvärdigheten och stödja en fredsrationell drift och ett fredsrationellt materielunderhåll. Systemet innehåller funktioner för drift- och underhållsplanering av försvarets flygplan och helikoptrar samt tillhörande materieluppföljning vid försvarets flygflottiljer, baser, verkstäder, samt vid större externa underhållsverkstäder/materielleverantörer. Förutom funktioner för materieluppföljning och underhållsplanering finns funktioner för att t.ex. redovisa driftstörningar under flygning och på marken, underhållsplanering, uppföljning av flygförarnas flygtid, flygmateriel, drifttid o.s.v.

DIDAS Marktele är ett delsystem för funktionsuppföljningssystem för flygvapnets marktelemateriel som mot bakgrunden av taktiska krav ger underlag för åtgärder i syfte att uppnå och vidmakthålla önskad driftsäkerhet. Uppföljning görs vid centralanläggningar och bedrivs kontinuerligt för att ge information om funktionernas driftsäkerhet samt utpekning av svaga länkar i funktionskedjan. Tidigare bestod DIDAS Marktele av två delar, en för bruksenhetsuppföljning och en för funktionsuppföljning, idag är det huvudsakligen bara funktionsuppföljning i systemet.

FREJ

FREJ har hand om information för identifierande data, tekniska data och förvaltningsdata för Försvarmaktens förnödenheter. Det är för stöd för anskaffning, försörjning, förbandsproduktion och underhåll samt för dessa verksamheter nödvändig redovisning av förnödenhetsdata. FREJ används även för framtagning av reservdelskataloger samt sats- och tillbehörslistor. Informationen i FREJ överförs enligt särskilda rutiner till andra datorbaserade system inom Försvarmakten, som för sitt ändamål och syfte måste ha information i termer av enhetliga och ensade grund- och förvaltningsdata.

DELTA

DELTA är ett styr- och informationssystem för Försvarets reservmaterieförsörjning. Rutiner finns för lokal, regional samt central nivå. Den generella systemuppgiften är att ge datastöd åt FM, FMV och försvarsindustrin. Huvuduppgiften är att förnödenhetsförsörja underhållsaktörer genom att anskaffa och redovisa reservdelars tillgång, placering och pris, samt vid försäljning skapa underlag för fakturering och vid inleverans skapa underlag för att betala leverantörsskulder.

Appendix B –Agentsystem

AgentSpace

AgentSpace är ett namn som används för flera olika agentsystem som har utvecklats separat inom universitets- och forskningsvärlden. Det mest kända agentsystemet under denna benämning är det som Ichiro Satoh [Satoh] utvecklade i slutet av 90-talet och som sedan utvecklades till det MobileSpaces, ett mer gediget agentramverk för mobila agenter (se vidare MobileSpaces). Satohs tidigare projekt AgentSpace verkar inte längre vara aktivt, även om det diskuteras i agent-sammanhang fortfarande.

Ett annat AgentSpace, som är utvecklat på senare år, är även detta utvecklat inom universitetsvärlden [AgentSpace]. Systemet är utvecklat för att passa även tunnare klienter, såsom handdatorer.

Aglets

Aglets är ett ramverk för mobila agentapplikationer som skapades av IBM, specifikt för utveckling av agenter som ska kunna användas över Internet [Aglets]. Ramverket är anpassat för utveckling av agentbaserade affärssystem och system för handel över webben. Exempel på tillämpning är en kund som skickar iväg en förfrågan om exempelvis pris eller varutillgång med agenten. För att besvara frågan skickas agenten över Internet för att söka och samla in information. När den väl har blivit godkänd av den mottagande maskinen (som kan vara en databas, en server hos en leverantör etc.) kan agenten få tillgång till den data och de tjänster den söker, för att återföra dem till den väntande klienten. Den kan även interagera med andra *aglets*, på de platser den hämtar information från, för att lösa uppgiften tillsammans.

Skillnaden mellan *Aglets* och till exempel *Cougaar*-agenter (se nedan) sägs vara att *Aglets* transporteras över nätet för att exekveras på distans hos klienter. *Cougaar* stödjer också denna funktionalitet, men fokus ligger mer på att bistå med större stöd för komplexa agenter (vad gäller t.ex. regelbaserat beteende, komplext beteende, agentens minneshantering och hur de resonerar, regelverk för semantik etc.).

Eftersom många länkar för *Aglets* är nere och eftersom koden som går att ladda ner är oförändrad sedan 2002, antas projektet vara nerlagt, vilket är synd eftersom mycket har gjorts inom ramverket och flera böcker och artiklar har skrivits inom ämnet. Detta ramverk rekommenderas inte för vidare arbete.

aIsland

aIsland är ett ramverk för utveckling av mobila agenter för distribuerade multi-agentsystem och som ursprungligen utvecklades för att experimentera med området Artificiell Intelligens (AI). Allt baseras på den underliggande nätverksarkitekturen JXTA¹, och är därför såsom JXTA plattformsoberoende, helt implementerat i Java, helt gratis och *open source*. Det verkar vara ett rätt färskt projekt som inte är helt underhållet, och rekommenderas därför inte för fortsatt arbete. Men det har däremot en del fördelaktiga tjänster som kan undersökas, till exempel en editor för att skapa agenter mha. JavaScript, auktoriserad information hanteras på sofistikerade och väl utvecklade sätt osv. Sista versionen släpptes i februari 2003 och har nr. 0.1.2 [JXTA].

¹ JXTA är en P2P-teknik som har utvecklats och utvecklas fortfarande som *OpenSource* med syftet att tillsammans med industrin och intresserade organisationer försöka uppbringa en gemensam standard för P2P-system [JXTA].

Bond

Bond är ett multiagent-system som i grunden baseras på agentramverket JADE (se nedan [Bond]). Detta innebär att Bond, liksom JADE, är baserat på Java, följer FIPA-standarderna, samt bygger på en komponentbaserad arkitektur. Bonds vidareutveckling av JADE har skett med fokus på att utveckla GRID-system. Agenterna i Bond har en hierarkisk syn på strategier och plan för agenterna och beskrivningsspråket för agenterna baseras på Python. I Bond är strukturen kring agenternas beteende mer utvecklad än i JADE och är dynamiskt vad gäller implementering, och uppladdning av strategier m.m.

Senaste versionen av Bond kom i februari 2003 och är ett agentsystem som är för specifikt för att användas till militär logistiktillämpning.

Cougaar

Cougaar – Cognitive Agent Architecture – är en Java-baserad arkitektur för utveckling av storskaliga distribuerade agentapplikationer, dvs. system eller ”samhällen” (societies). Arkitekturen utvecklades under DARPA’s *Advanced Logistics Project* (ALP), inom försvaret i USA, för användning inom militär logistik [Cougaar]. Det är dessutom ett open source projekt, vilket bidrar till arkitekturens utbredning. Projektet härstammar ur två forskningsprojekt under DARPA, varav det ena har arbetat med utveckling av agentsystem för storskaliga, snabba, distribuerade beräkningar för logistisk planering. Det andra projektet hanterade informationstekniker och mycket skalbara agentsystem för logistik i kaotiska miljöer. Arkitekturen är därför utvecklad för att vara stabil och komponentbaserad och består med en grund för att bygga mobila agenter som kan kommunicera med varandra.

Civila tillämpningsområden för Cougaar inkluderar inbyggda ”*sensor webs*”, och ”*business intelligence*”. Militärt används Cougaar t.ex. som underliggande arkitektur i UltraLog-projektet [UltraLog], som är ett av de DARPA-projekt som tog fram Cougaar (det senare ovan).

D’Agents

Såsom många agentsystem så utvecklas agentsystemet D’Agents inom universitetsvärlden, på Dartmouth College, och för att användas som experimentell forskning kring agenter och relaterade tekniker och mekanismer. Målet med D’Agents är att stödja applikationer som kräver inhämtning, organisering och presentation av distribuerad information i dynamiska (slumpmässiga) nätverk. De forskningsämnen som har hanterats (med framgång) i arbetet är t.ex. säkerhetsmekanismer, stöd för mobila och endast delvis kopplade datorer, verktyg för resurshantering, automatisk indexering av resurser samt många fler.

Agentsystemet är inte avsett att vara en komplett produkt och vidareutvecklas inte heller i sig själv. Senaste versionen av mjukvaran är daterad 2002 och är således inaktuell. Däremot kan forskningsresultat och framtagna metoder vara värda att se extra på vid praktisk implementering av agentsystem.

DIET Agents

DIET Agents är en multiagent-plattform som är utvecklad i Java [DIET Agents]. Den utvecklades som del av DIET-projektet och släpptes som *open source* när projektet avslutades. Enligt [DIET Agents] utvecklades plattformen med hjälp av en botten-upp-design som användes för att försäkra att plattformen är lättvikt, skalbar, robust, adaptiv och utökningsbar. Den är särskilt lämplig för att snabbt utveckla p2p-prototyp-applikationer, och/eller adaptiva, distribuerade applikationer som utnyttjar *bottom-up*-tekniker. I DIET har man inte satsat på säkerhetsfrågor, utan snarare på skalbarhet och robusthet. Ramverket är gratis att ladda ner från hemsidan [DIET] och senaste versionen av plattformen (version 0.93) släpptes i januari 2004. Den första publika versionen av DIET Agents släpptes i Augusti 2003.

Hive

Hive är en decentraliserad, distribuerad plattform för mobila agenter [Hive]. Med Hive kan system skapas som kan knytas till och söka information över Internet och det var detta som var det stora syftet när Hive utvecklades. Hive tillåter att enheter kan kommunicera och kopplas ihop, enligt [Hive] utan för stor anpassning till enhetens arkitektur. Arkitekturen baseras på tre olika delar:

- S.k. Hive-celler, som används som agentservrar och som tillhandahåller en infrastruktur för att söka resurser, stöd för kommunikation mellan agenter m.m.
- S.k. skuggor som är *Hive drivers* för resurser och enheter i nätverket, det vill säga den mjukvara som sammankopplar enheten med systemet. Dessa är specifika för den enhet som integreras.
- Hive-agenter som kommunicerar med varandra och interagerar med enheterna.

För att agenterna ska hitta, dels varandra och dels enheterna, så tillhandahåller Hive ett syntaktiskt/semantiskt schema för resurskartläggning och –sökning. Hive utvecklades i forskningssyfte och släpptes med öppen källkod och dokumentation. Dock är den senaste versionen från 1999 och ingen vidare utveckling förväntas.

Grasshopper

Grasshopper är en agentplattform för utveckling av adaptiva kommunikations- och informations-applikationer. Plattformen är utvecklad så att *add-ons*, alltså tilläggsdelar, kan läggas till med extra funktionalitet. Ett exempel är en FIPA-*add-on* som möjliggör för agentutveckling som följer FIPA-standarderna. Plattformen är helt Java-baserad och stödjer utveckling av mobila agentsystem. I Grasshopper kan agenter grupperas efter typ eller domän, och de olika grupperna kan ha olika hantering och egenskaper. Kärnplattformen tillhandahåller mycket funktionalitet för säkerhet, hantering, kommunikation och så vidare. Grasshopper finns även för mobila klienter såsom handdatorer.

Grasshopper är gratis att ladda ner från hemsidan [Grasshopper]. Sista versionen av Grasshopper heter numera Grasshopper 2 och kom i januari 2003. Forumet och utvecklingen kring ramverket samt diskussionsforumet för utveckling är aktivt. Ramverket tillhandahåller inte öppen källkod, men de *add-ons* som existerar är öppna. Detta gör att produkten går att vidareutveckla, men till viss del är begränsad till produkttillverkaren, vilket gör att det är mer fördelaktigt att använda en helt öppen produkt.

JADE

JADE står för *Java Agent Development Framework*. Det är en plattform (ett ramverk) för utveckling av agentbaserade P2P-applikationer och det är *open source* [JADE]. Det sägs förenkla implementeringen av multi-agentsystem, genom att använda ett mellanlager, som följer FIPA-specifikationer, samt ett antal grafiska verktyg som stödjer debugging och driftsättning av systemet. Agentplattformen kan distribueras över maskiner (som inte behöver ha samma OS) och konfigurationen kan kontrolleras via ett fjärr-GUI. Konfigurationen kan även ändras i *run-time* (dvs. under agentens exekvering) genom att flytta agenter från en maskin till en annan när det behövs.

JADE är helt implementerat i Java och nuvarande version fungerar på minst version 1.4 av programmeringsspråket. På grund av synergin mellan JADE-plattformen och LEAP-biblioteken kan JADE tillhandahålla en agentplattform som följer FIPA och dessutom stödjer mobila Java-miljöer som J2ME etc. LEAP-biblioteken har utvecklats externt JADE men kan laddas ner som en *add-on*. JADE-plattformen är helt gratis och distribueras av TILAB. Den senaste versionen är JADE 3.1 och släpptes i december 2003.

MobileSpaces

MobileSpaces, som är utvecklat under ledning av Ichiro Satoh [MobileSpaces], bygger vidare på agentsystemet AgentSpace (se AgentSpace ovan). MobileSpaces är ett multiagent-system av mobila agenter som migrerar både status och kod när agenten förflyttas till en annan nod. En särskilt egenskap hos systemet är att agenter inte bara kan migrera från nod till nod, utan flera agenter kan migrera in i en enda agent. Systemet vidareutvecklas kontinuerligt och är gratis att ladda ner från hemsidan [MobileSpaces], men är ett universitetsprojekt och således inte en komplett produkt som lämpar sig för skarp implementation i existerande system.

NOMADS

NOMADS är ett projekt som ämnar utveckla ett antal distribuerade och agentbaserade system i Java [Suri et al.]. För detta syfte har de utvecklat en ny Java Virtuellt Maskin (JVM), AromaVM. Den stora fördelen med AromaVM gentemot existerande JVM:er är att AromaVM tillhandahåller mekanismer för att spara status m.m. för exekverande trådar. Detta medför att mobiliteten för agenterna stärks och kan garanteras bättre. Dessutom har AromaVM kontrollmekanismer för resurser, som kan användas för att lokalisera och allokera resurser för agenternas räkning.

AromaVM verkar vara en intressant produkt att se närmare på i sammanhanget avancerade distribuerade system och agentutveckling. NOMADS däremot, som går att ladda ner gratis från hemsidan [NOMADS], är inte lika intressant, då den senaste versionen av denna släpptes 2000 och inte mycket vidareutveckling sker av systemet.

RETSINA

RETSINA står för Reusable Environment for Task Structured Intelligent Network Agents. Det är en infrastruktur för multiagentsystem för heterogena agentsamhällen. RETSINA har enligt [RETSINA] applicerats inom ett flertal olika domäner såsom portföljhantering (finansmarknaden), personliga (användaranpassade) webbtjänster, bokhandel och auktioner på nätet, logistikplanering inom militära domäner, samt för mobil och trådlös kommunikation. RETSINA är helt distribuerat och baseras på konceptet att alla agenter är noder (peers) som kommunicerar *Peer-to-peer* och härigenom formar distribuerade samlingar av noder (likt många andra agentsystem). RETSINA MAS styrs alltså helt decentraliserat och tillhandahåller tjänster för att den distribuerade kommunikationen ska ske så smidigt som möjligt.

RETSINA består av fyra olika typer av agenter:

- Gränssnittsagenten som användaren interagerar med och får resultat ifrån Uppgifts-agenten.
- Uppgiftsagenten som utför uppgifter på klientens begäran möjligen i samarbete med andra Uppgiftsagenter.
- Informationsagenter som på ett effektivt sätt tillhandahåller tillgång till en mängd heterogena informationsresurser.
- Mellanagenter som utnyttjas för att matcha agenter som efterfrågar en tjänst med agenter som tillhandahåller den tjänsten.

Vad gäller logistik så används agenterna till ett mjukvarusystem som heter MokSAF [MokSAF]. De som har utvecklat RETSINA är *the Intelligent Software Agents Lab* som även har kombinerat RETSINA-agenter med Web Services för att få stabila lättsökta tjänster på Internet! RETSINA verkar vara ett väl utvecklat agentramverk, men då kod och utveckling inte är öppen för allmänheten rekommenderas inte RETSINA för användning och vidareutveckling.

Tryllian

Tryllian är ett företag som tillhandahåller produkter för att utveckla adaptiva affärsapplikationer. Tryllian-system används främst till att integrera affärsapplikationer och –system och produkterna är därför anpassade till detta syfte. Systemen som utvecklas baseras på vad Tryllian själva kallar ”*Dynamic Smart Components*” (DSCs), men som i praktiken är mobila agenter. DCSs är enligt Tryllian säkra, mobila och till viss del intelligenta. För utveckling tillhandahåller de ett gediget utvecklingspaket (Agent Development Kit) med vilket avancerade, storskaliga och säkra agentsystem kan utvecklas i Java-miljö.

Eftersom Tryllian har funnits på marknaden länge och har en väl etablerad teknik samt tillhandahåller smarta lösningar på vanligen uppkommande problem (såsom säkerhet) så pratas det ofta om Tryllian i agentsammanhang. Tryllian är med säkerhet fördelaktigt att utnyttja för organisationer som vill utveckla stabila agentsystem. Dock är produkten kommersiell och är därför inte bara kostnadsbelagd, utan även begränsad i vidareutveckling (något som är av stor vikt vid arbetet i detta projekt). Visserligen går agentramverket att tilläggas funktionalitet på ett smidigt sätt i form av egenutvecklade JavaBeans, men ingen öppen källkod skeppas för närvarande med produkten.