ENTERPRISE JavaBeans

- UR ETT SYSTEMUTVECKLINGSPERSPEKTIV -

Magisteruppsats i informatik, Göteborgs Universitet

Jonas Bergqvist j@skip.informatik.gu.se
Anders Ericsson anders@skip.informatik.gu.se
VT 2001 Handledare: Stefan Olsson

Sammanfattning

Det höga tempot som råder inom systemutvecklingsvärlden för internetbaserade programvaror ställer stora krav på de företag som utvecklar denna mjukvara. Krav på hög prestanda och driftsäkerhet samt bra funktionalitet skall uppnås samtidigt som utvecklingen skall gå så fort som möjligt och till så låg kostnad som möjligt.

Vår uppsats fokuserar på utvecklandet av flerskiktade, distribuerade system i denna situation. För att uppnå ett bra resultat vid utveckling av flerskiktade, distribuerade system undersökte vi om ramverket *Enterprise JavaBeans* (EJB) kunde vara ett hjälpmedel för systemutvecklare. EJB är ett standardiserat ramverk för komponentintegration.

Syftet med uppsatsen var att undersöka vilka svagheter och styrkor som EJB har ur en systemutvecklares perspektiv. Vår problemställning var därför att komma fram till *på vilka sätt det är bättre att använda ramverket Enterprise JavaBeans för att bygga ett distribuerat, flerskiktat system än att utveckla samma funktionalitet i Java utan att använda något ramverk*.

Problemställningen avgränsades till att behandla enbart systemutvecklingsaspekten.

Vårt arbete bedrevs i fem faser med följande metoder: litteraturstudie, kvalitativa intervjuer med javautvecklare utan erfarenhet av ramverket, prototyputveckling, kvalitativa intervjuer med erfarna EJB-utvecklare, samt en kvantitativ analys av vanligt ställda frågor på internet rörande ramverket.

Vår slutsats var att EJB-ramverket är bättre för att utveckla distribuerade, flerskiktade system på många sätt än traditionell systemutveckling i Java. De huvudsakliga orsakerna till detta var de tjänster som ramverket erbjuder systemutvecklaren i form av automatisk hantering av samtidighet, transaktioner, beständighet, distribuerade objekt och namngivning. Samtidigt visade studien att EJB inte lämpar sig för alla typer av systemutvecklingsprojekt och att systemutvecklaren därför noga bör utvärdera nyttan av tjänsterna i sitt aktuella projekt.

1 Introduktion

1.1 Bakgrund

I internetvärlden ställs det höga krav på de företag som utvecklar mjukvara. Förutom att mjukvaran eller webbplatserna skall vara lättanvända och trevliga att titta på, så ställs det höga krav på snabbhet, tillgänglighet och felsäkerhet.

Samtidigt som systemutvecklarna skall möta alla dessa krav, så har företagen de arbetar för ofta mycket bråttom med att lansera sin nya mjukvara eller webbtjänst.

Detta sätter utvecklare och projektledare i en besvärlig sits, då de skall väga snabb utvecklingstid gentemot god prestanda och funktionalitet.

Denna situation är dock inte ny, utan är något som präglat systemutvecklings-processen under hela mjukvarubranschens existens. Länge har systemutveklare därför drömt om hjälpmedel som kan hjälpa dem att arbeta effektivare och även få dem att prestera ett bättre resultat.

Vår uppsats fokuserar därför på en lovande så kallad komponentarkitektur från SUN Microsystems kallad *Enterprise JavaBeans* (EJB). Vi studerar denna arkitektur som ett ramverk för att snabbare och lättare kunna utveckla komplexa mjukvarusystem.

Denna introduktion tjänar till att presentera bakgrunden till vårt arbete och att introducera läsaren till den rika flora av begrepp och förkortningar som präglar avancerad systemutveckling av idag. Många av de begrepp som presenteras förklaras därför mer ingående i teoriavsnittet.

1.1.1 Problemområde

Vårt problemområde inbegriper utvecklingen av distribuerade, flerskiktade system och de utmaningar och problem man ställs framför som utvecklare av sådana.

1.1.1.1 Flerskiktade system – söndra och härska

Nuförtiden är skiktning av system något mycket vanligt. Skiktning innebär oftast att man delar in ett system i väl avskilda delar som hanterar olika delar av systemet. Ett exempel på skiktning är den treskiktade arkitekturen (se teorikapitlet). Här delar man in systemet i ett lager bestående av gränssnitt mot användare och andra system. Det mellanliggande lagret består av all affärs- och programlogik. Slutligen kommunicerar det mellanliggande lagret med ett tredje, som ofta är ett datalagringslager, vilket oftast är instruktioner för att kommunicera med en databas.

Man uppnår många fördelar med att skikta system på detta vis. Bland annat går det ofta mycket lättare att lägga till ett nytt presentationssätt (Tångring, J. 2000).

Ponera att vi exempelvis har utvecklat ett system för att boka biljetter via ett webb-gränssnitt och har separerat all vår affärslogik i ett mellanlager. Eftersom affärslogiken är separerad från presentationssättet behöver vi inte utveckla det på nytt utan utvecklarna behöver endast skriva det nya presentationslagret (för t.ex. WAP).

Flerskiktade system behöver inte bara vara treskiktade, utan kan även ha fler eller färre lager. Därför talar man om flerskiktade system som flerskiktsarkitekturer (*eng.* n-tier architecture).

Att jobba med flerskiktade system har visat sig vara ett bra tillvägagångssätt för att uppnå bra strukturerade system. Det gör det lättare att skapa ett system som består av fristående moduler där det är lätt att byta ut de enskilda delarna.

Den största nackdelen med att skikta ett system är att man måste vara noga vid designen av komponentmodellen. Att skikta kräver alltså en relativt stor arbetsinsats. Systemutvecklarna måste ofta utveckla ett eget ramverk för hela systemet. Det vore därför önskvärt med ett färdigt ramverk som gör det lätt att skikta system.

1.1.1.2 Distribuerade system

I många systemutvecklingsprojekt måste man utveckla system som är distribuerade. Detta innebär att systemet inte utför alla programanrop på en och samma maskin. Ofta är detta fallet i flerskiktade system där man placerar de olika skikten på olika maskiner. En maskin kanske hanterar gränssnittslagret gentemot klienterna, en annan hanterar affärslogiken och en tredje maskin inhyser databasen.

Detta innebär att flera maskiner och användare måste dela på gemensamma resurser. Ta exemplet med en nätbokhandel. Flera användare har tillgång till att se och reservera de böcker som finns i systemet.

Vad händer om två personer samtidigt försöker köpa samma vara och det bara finns en kvar? Kanske debiterar systemet bägge kunderna för varan, men det finns bara en bok att skicka iväg.

Problem som det i exemplet ovan rör koncepten *samtidighet* och *transaktionshantering*.

1.1.1.3 Samtidighet och transaktionshantering

Samtidighet betyder att två saker kan ske samtidigt i ett mjukvarusystem. I ett enanvändarsystem är detta sällan några problem, en användare utnyttjar oftast bara en resurs åt gången. Så fort vi blandar in fler användare så uppstår dock problem som de i exemplet ovan.

Vad händer i ett system där två processer samtidigt försöker manipulera data? Ofta uppstår problem med felaktig data. Därför måste systemutvecklarna hantera samtidigheten i systemet. Detta kan göras på flera sätt. Ett tillvägagångssätt är att se till att bara en process i taget kan komma åt en resurs.

Ibland räcker dock inte ens detta till. Vad händer om flera resurser i rad skall behandlas och en av resurserna plötsligt har ändrats av en annan process eller om något annat går fel?

Problemet är då att processen redan har utfört en del ändringar i systemet och dessa har ju ansetts som valida, eftersom ingen annan process har rört dem samtidigt. Det vore då önskvärt att kunna ha de tidigare ändringarna ogjorda. För att kunna göra något åt detta problem behöver man transaktioner.

En transaktion är en kedja av instruktioner som använder sig av en eller flera delade resurser (Monson-Haefel, 2000, s. 273). Ett exempel kan vara:

- Registrera en vara som beställd
- Kontrollera kontokortsinformation
- Debitera kunden

- Skicka varan

Säg att något går fel vid steg tre. Kanske är systemets kommunikation med kontokortsutgivaren nere. Då kan vi ju inte skicka varan och vara säkra på att få vår betalning. Och vi vill ju inte heller att varan skall fortsätta vara beställd, eftersom ingen annan kund då kan beställa den.

Med transaktionshantering menar man att ett system kan konstruera en uppsättning instruktioner i form av en transaktion och sedan kontrollera att antingen alla eller ingen av instruktionerna utförs. Går något fel i kedjan så skall systemet alltså göra de tidigare instruktionerna ogjorda.

Som man kan förstå kan detta vara ett omfattande jobb att hantera om man inte har något färdigt ramverk för transaktioner och samtidighet.

1.1.1.4 Ett lovande ramverk

Enterprise JavaBeans (EJB) från Sun är en komponentarkitektur som utlovar snabbare utvecklingstid för klient/server-applikationer med hög prestanda och tillförlitlighet (Zona Research, 1999).

Att hantera samtidighet och transaktioner kan ta stor del av utvecklingsarbetet. Ett ramverk som kan underlätta hanteringen av samtidighet och transaktioner, distribuerade objekt, beständig lagring och säkerhet ter sig mycket lovande. Hantering av samtidighet och transaktioner är två av EJB:s primära tjänster (Monson-Haefel, 2000, kap. 3).

EJB är enkelt sagt komponenter som tillåter mjukvara att kommunicera i flerskiktade klient/server-miljöer och även över intra- och extranet. EJB bör därför göra det lättare att utveckla flerskiktade distribuerade lösningar där distribuerade objekt behöver kommunicera med varandra. Detta gör i så fall att systemutvecklarna kan fokusera mer på att utveckla och implementera affärslogiken (Hemrajani, 1999).

Analysföretaget Gartner Group hävdar att EJB kommer att vara en av de två dominerande teknikerna för internetutveckling inom en snar framtid (Olars Jakobsson, 2000).

1.1.1.5 Transaction Processing Monitors

Tidigare system för att hantera transaktioner i till exempel bankvärlden har utjgorts av så kallade *Transaction Processing Monitors (TPM).* Dessa fugerar praktiskt taget som ett helt operativsystem för ett affärssystem. De hanterar allt ifrån transaktioner, och resurshantering till feltolerans. Problemen med dessa är att de ofta är helt knutna till en viss tillverkare och att logiken är procedurellt uppbyggd. En TPM har därför inget stöd för kommunikation mellan distribuerade objekt. Den kan alltså inte dra nytta av de fördelar man får av en objektorienterad systemutvecklingsprocess.

1.1.1.6 Distribuerade objekt

CORBA (Common Object Request Broker Architecture) är en standard för att hantera distribuerade objekt. Problemet med CORBA är att den bara ger utvecklaren tillgång till ett ramverk för att skriva program med distribuerade objekt, det är i huvudsak bara ett kommunikationsprotokoll.

CORBA ger inte utvecklaren tillgång till vare sig hantering av transaktionshantering eller samtidighet eller några andra liknande tjänster.

1.1.1.7 Component Transaction Monitors

Enterprise JavaBeans är en komponentmodell för så kallade *Component Transaction Monitors* (CTM).

En *komponentmodell* är en uppsättning klasser och interface som måste användas på ett visst sätt för att isolera och kapsla in en viss uppsättning funktionalitet (dessa koncept behandlas ingående i teoriavsnittet).

Varje CTM kan stödja en eller flera sådana komponentmodeller. Microsoft Transaction Server är ett exempel på en CTM som bara stödjer sin egen komponentmodell. Detta leder till att utvecklarna låser sig till just Microsofts CTM.

EJB är däremot en standardiserad komponentmodell. Detta betyder att utvecklare som gör system med EJB kan använda sig av vilken CTM som helst, bara den stödjer EJB-standarden.

En CTM hanterar de tjänster som komponentmodellen behöver. Dessa är bl. a. stöd för distribuerade objekt och en infrastruktur för transaktioner.

1.1.1.7.1 EJB-servrar

Enterprise JavaBeans kan endast användas tillsammans med en CTM. Dessa kallas då för *EJB-servrar*. En EJB-server är oftast integrerad i vad som kallas en applikationsserver eller tillämpningsserver. En sådan applikationsserver tillhandahåller därför ofta mer funktionalitet än att bara vara en EJB-server, den kan till exempel fungera som webbserver eller ha egna funktioner för att hantera speciell programlogik eller databasåtkomst (Monson-Haefel, 2000). Applikationsservrar och CTM beskrivs mer ingående i teoriavsnittet.

1.2 Uppdragsbeskrivning

För att närmare kunna undersöka möjligheterna med EJB samarbetade vi under resultatinhämtningen med företaget Xdin AB i Göteborg. De har precis beslutat att försöka införa Enterprise JavaBeans i sina kommande produkter.

Företaget är mellanstort med ett par hundra anställda uppdelade på olika produktområden och avdelningar. Systemutvecklingsavdelningen består av ungefär 30 personer, varav hälften arbetar med utveckling för Java-plattformen.

Avdelningen utvecklar mjukvara för att stöjda kundernas affärsprocesser och interna system. En stor del av utvecklingen består i att ta fram klient/server-lösningar av olika storlek.

Arbetet innebär ofta att koppla samman äldre och nyare system med webb-klienter. Detta arbete kan vara mycket tidskrävande och kräver ofta att man skiktar systemet i en flerskiktsarkitektur. Oftast innebär detta att man delar upp systemet i ett treskiktssystem där man separerar gränssnitt, programlogik och datalagring.

Vårt arbete på Xdin inbegrep att specifiera, designa och utveckla en prototyp för ett nyhetssystem med hjälp av EJB. Resultatet av vårt arbete användes sedan i vår uppsats som en stor del av resultatsunderlaget.

1.2.1 Syfte

Uppsatsens syfte är att undersöka vilka svagheter och styrkor som Enterprise JavaBeans (EJB) har ur en systemutvecklares perspektiv.

Genom syftet kommer vi att kunna prestera ett resultat som kan användas vid val av systemutvecklingsarkitektur för flerskiktade, distribuerade system.

1.2.2 Problemställning

För att kunna uppfylla vårt syfte har vi valt att begränsa vår problemställning.

På vilka sätt är det bättre att använda ramverket Enterprise JavaBeans för att bygga ett distribuerat, flerskiktat system än att utveckla samma funktionalitet i Java utan att använda något ramverk?

Genom att besvara denna frågeställning kommer vi och systemutvecklarna på företaget få en bra orientering om EJB:s styrkor och svagheter, vilket kommer att göra det lättare att veta när EJB bör användas i systemutvecklingsprojekt.

Vi har i vår problemställning utgått ifrån att det är bättre att använda Enterprise JavaBeans. Detta eftersom det är vad som teknologin utlovar. Studien kan även resultera i ett resultat som negerar detta påstående. Skulle så vara fallet kommer skälen till detta att redovisas.

1.2.3 Avgränsning

Vi har avgränsat vår uppsats att behandla först och främst EJB som ett ramverk för systemutvecklare att använda sig av när de utvecklar distribuerade, flerskiktade system. Detta innebär att vi har fokuserat på den eventuella nytta just systemutvecklaren kan uppnå genom att använda sig av ramverket.

Eftersom vi också fokuserar på just system utvecklade i Java, så inbegriper inte uppsatsen att jämföra EJB kvalitativt med andra teknologier eller ramverk. De jämförelser som görs syftar enbart till att belysa EJB-ramverkets uppbyggnad.

1.2.4 Disposition

I avsnitt två redovisar vi för teorier och tidigare forskning relaterade till uppsatsen. Detta inbegriper en genomgång av distribuerade objektsystem, komponentmodeller och ramverk för flerskiktade distribuerade system.

I avsnitt tre beskriver vi den samling av metoder som vi använt för att uppnå syftet med uppsatsen. Dessa metoder består av en litteraturstudie, kvalitativa intervjuer, prototyputveckling och en mindre kvantitativ studie bestående av en analys av FAQ:er (Frequently Asked Questions) på Internet.

I avsnitt fyra beskrivs arbetet och prototyputvecklingen exemplifieras. Systemet och systemutvecklingsmetoden beskrivs översiktligt.

Uppsatsens resultat redovisas i avsnitt fem.

I avnitt sex för vi en diskussion kring problemställningen och försöker att besvara den.

2 Teori

Detta kapitel skall ge läsaren en inblick i de områden som är av intresse när man studerar det ramverk som EJB utgör. Vi redogör också för tidigare forskning inom dessa områden.

Först presenteras en överblick av vad distribuerade objektarkitekturer innebär. I detta ingår en kort redogörelse över teknikerna *Java RMI* (Remote Method Invocation) och CORBA.

Därefter presenteras kortfattat konceptet med klient/server-arkitektur och hur dessa passar in i ett flerskiktat ramverkstänkande.

Nästföljande avsnitt behandlar konceptet med ramverk för flerskiktade distribuerade system. Vi ger exempel på generella och specifika ramverk och systemarkitekturer för systemutveckling.

Slutligen studerar vi Enterprise JavaBeans mer i detalj. Vi beskriver hur ramverket är uppbyggt, hur det fungerar och hur det är tänkt att användas.

2.1 Distribuerade system

Distribuerade system gör det möjligt att skapa mer tillgängliga affärssystem (Monson-Haefel, 2000). Detta är möjligt eftersom olika delar av systemet kan vara lokaliserade på fristående datorer och dessa kan i sin tur tänkas stå på helt olika geografiska platser. Distribuerade system innebär alltså att affärslogik och affärsdata kan nås från helt andra platser än där de existerar rent fysiskt. Ett systems användare (kunder, affärspartners och utomstående personer till exempel) kan alltså nå ett visst affärssystem från nästan var som helst.

Den senaste utvecklingen inom distribuerade system kom med intåget av distribuerade objekt. Distribuerade objektteknologier som Java RMI, CORBA och Microsofts DCOM gör att objekt som finns på en viss maskin kan nås av klientapplikationer på flera andra datorer.

I inledningen talade vi om *TP monitor systems* (transaktionsbehandlingsövervakningssystem). Dessa är så kallade *legacy*-system; gamla system som länge dominerat affärssystemsidan. Dessa använder sig av en treskiktsarkitektur där presentation, affärslogik och databas delas in i tre skilda lager. Affärslogiken i dessa system är oftast skrivna i äldre, procedurella programmeringsspråk som COBOL eller PL/1 (Ibid).

Genom att införa distribuerade objektteknologier i stället för dessa system kan man ersätta den procedurella affärslogiken med affärsobjekt (business objects).

2.1.1 Objektorienterad affärslogik

Flexibilitet, påbyggbarhet och återanvändbarhet. Detta är de tre axiomen för objektorienterad systemutveckling (Ibid). Distribuerade objektarkitekturer kan föra in dessa fördelar även på affärssystemområdet.

Ett företags utveckling är dynamisk. Detta innebär att ett företags produkter, processer och mål utvecklas över tiden (Ibid). Genom att kapsla in affärslogik och affärsdata i affärsobjekt kan man uppnå dessa fördelar i ett stort affärssystem. Affärsobjekten blir flexibla, påbyggbara och återanvändbara eftersom de är objektorienterade. Detta leder

till att systemet kan utvecklas i takt med att företagets affärer och omgivning förändras (Ibid).

2.1.1.1 Serverkomponenter – affärsobjekt som Lego-bitar

När man talar om affärsobjekt gör man detta i en kontext av så kallade *serverkomponenter* (*eng.* server-side components). Dessa serverkomponenter, eller *distribuerade affärsobjekt*, kan ses som legobitar att användas för att bygga (pussla) ihop en affärsverksamhet i ett datorsystem.

För att utveckla dessa legobitar behövs en arkitektur. Denna kallas då för *server-komponentmodell* (*eng.* server-side component model).

Denna komponentmodell används på mellanlagret på en applikationsserver, vilken hanterar komponenterna under körning och gör dem tillgängliga för fjärrklienter (Ibid). Komponentmodellen förser utvecklaren med en uppsättning funktionalitet som gör det enkelt att uveckla distribuerade affärsobjekt och att sätta samman dessa till affärslösningar.

Serverkomponenterna (legobitarna) kan sedan i sin tur köpas och säljas som vilken paketerad programvara som helst. Redo att användas i ett nytt systembygge. Eftersom de är anpassade till en viss serverkomponentmodell kan de enkelt flyttas till en annan applikationsserver som stödjer den givna komponentmodellen (Ibid).

2.1.2 Component Transaction Monitors

I inledningen beskrev vi översiktligt vad en applikationsserver och en CTM var. Här utreder vi begreppen ytterliggare och sätter dem i ett gemensamt sammanhang.

Applikationsservrar är en speciell typ av avancerad mjukvara som växte fram i och med att behovet av att hantera komplexiteten i systemutveckling i internetvärlden ökade allt mer. En applikationsserver består oftast av en kombination av flera olika teknologier. Dessa kan till exempel vara webbservrar, ORB:ar (Object Request Brokers), *MOM* (message-oriented middleware), databaser och så vidare (Ibid).

Men en applikationsserver kan också fokusera på en viss teknologi, som till exempel distribuerade objekt. De typer av applikationsservrar som vi är intresserade av i vår uppsats är just de som är baserade på en sådan teknologi, som till exempel CORBA, Java RMI eller Microsofts DCOM.

2.1.2.1 Object Request Brokers (ORB)

Applikationsservrar som är baserade på distribuerad objekt varierar kraftigt i sin grad av sofistikering. De enklaste säkerställer kopplingen mellan klienten och de distribuerade objekten. Dessa kallas för ORB:ar, som vi nämnt innan. ORB:ar låter klientapplikationer lokalisera och använda distribuerade objekt på ett enkelt sätt. Det har emellertid visat sig att de inte fungerar tillräckligt bra i miljöer med väldigt många användare och en hög transaktionsvolym (Ibid).

ORB:arna erbjuder alltså en kommunikationskanal för distribuerade objekt, vilket är användbart i sig. De erbjuder däremot inte en robust infrastruktur för att hantera stora användarmängder och affärskritiskt arbete.

Komponentmodellen som ORB:arna erbjuder är dessutom ganska begränsad och ger inga möjligheter för automatisk hantering av transaktioner, samtidighet, beständighet

och andra systemtjänster. Om sådana tjänster finns så måste utvecklaren explicit använda sig av dem i sina program eller i värsta fall skapa dem själv (Ibid).

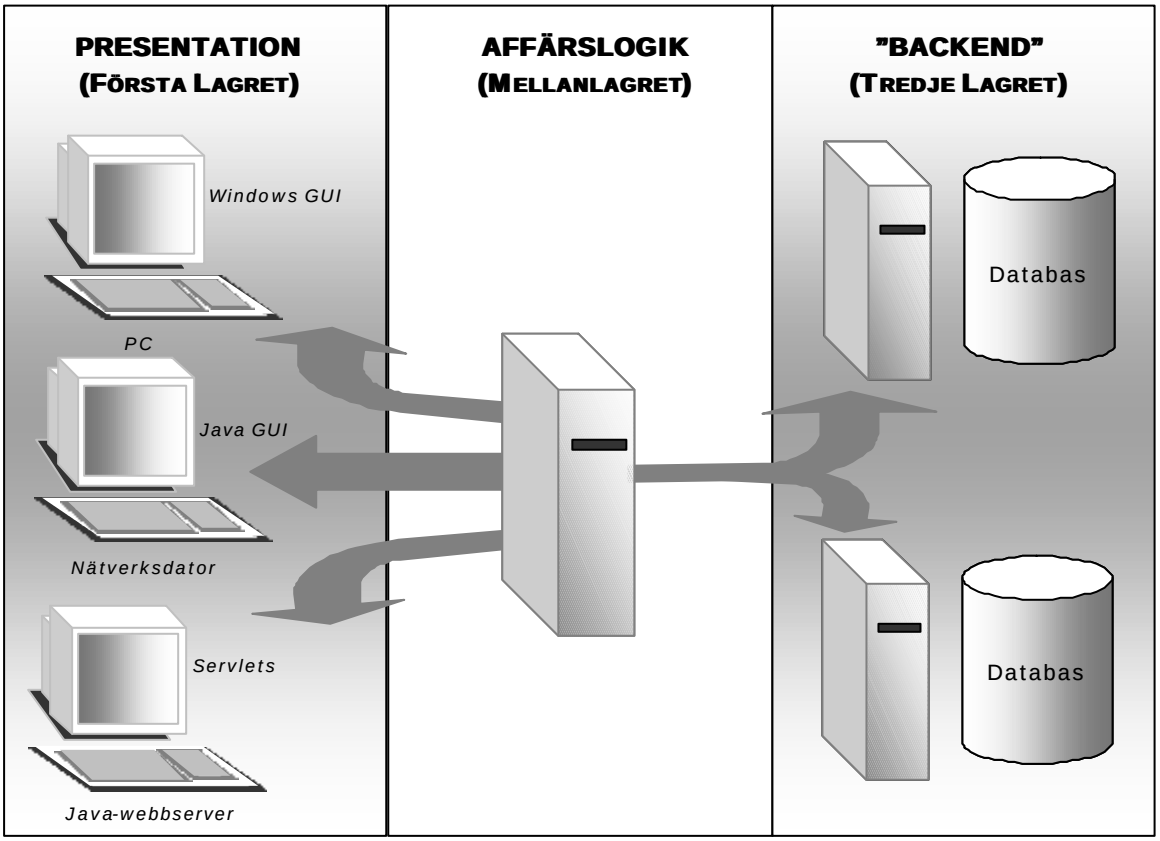
En CTM är den mest sofistikerade formen av applikationsservrar för distribuerade objekt. Termen myntades 1999 av Anne Thomas från Patricia Seybold Group. CTM:er kan sägas vara en hybrid av TP-monitorer och ORB-teknologier (Ibid). De har dels en kraftfull serverkomponentmodell som gör det enklare för utvecklare att skapa, använda och driftsätta affärssystem. Dels erbjuder de en robust infrastruktur som automatiskt kan hantera transaktioner, samtidighet, beständighet och resurshantering (Ibid).

2.1.2.2 Distribuerade objektarkitekturer

I det här stycket diskuterar vi hur distribuerade objekt funkar i praktiken. Vi visar också att distribuerade objektarkitekturer är grunden för alla moderna treskiktsarkitekturer.

2.1.2.2.1 Treskiktsarkitektur

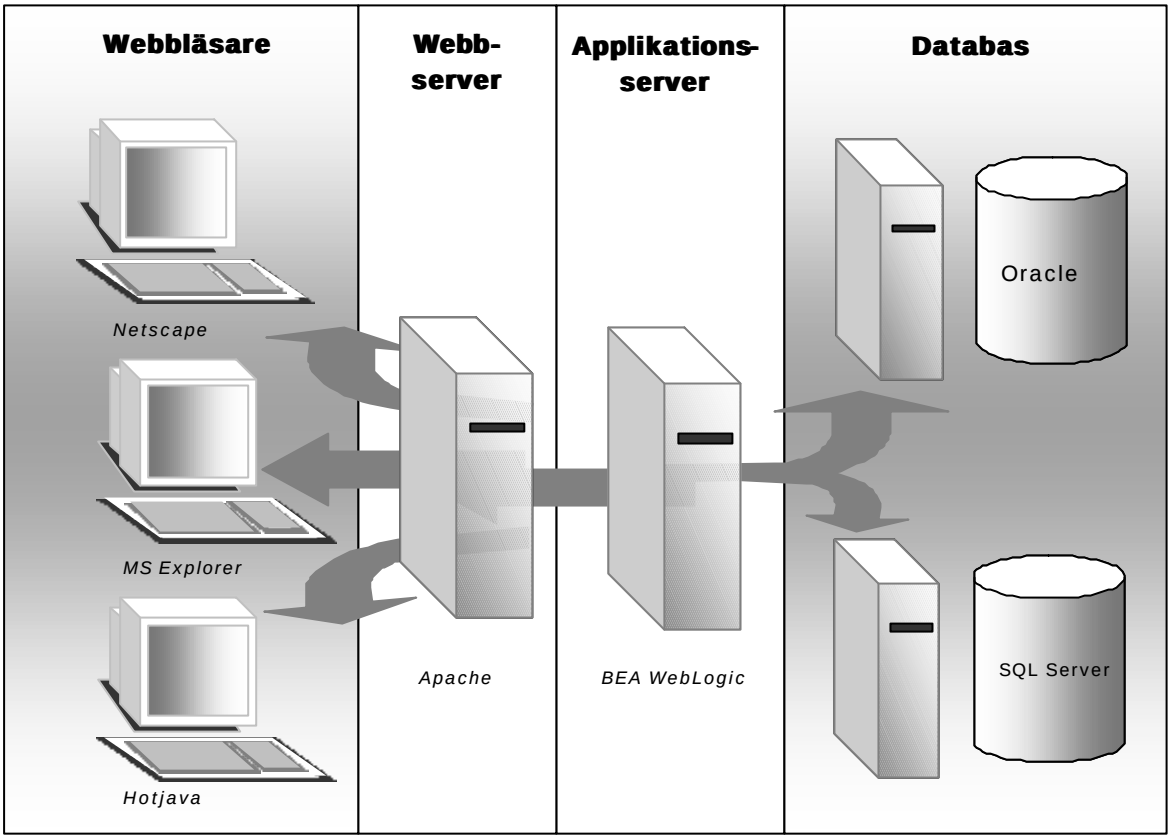
I bilden nedan visas en typisk treskiktsarkitekur. Presentationslogiken finns på klienten (första lagret), affärslogiken i mellanlagret ch andra resurser som databas etc. finns i det tredje lagret.



Figur 2.1 Treskiktsarkitektur. Egen illustration.

2.1.2.2.2 Fyrskiktsarkitektur

En fyrskiktad arkitektur har fyra lager, t.ex.: webbläsare, webbserver, applikationsserver och databas (Kjellgren,1998, s. 11-12).



Figur 2.2 Exempel på fyrskiktsarkitektur. Egen illustration.

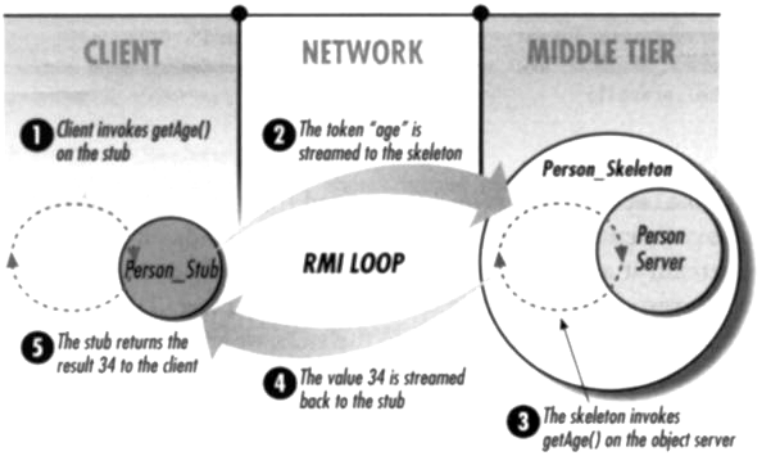
Observera att man i vissa fall inte räknar skiktet med webbläsaren och därför kallar även ovanstående för en treskiktsarkitektur.

2.1.2.2.3 Den distribuerade objektarkitekturen

Alla protokoll för distribuerade objektarkitekturer är baserade på samma grundläggande arkitektur. Denna går ut på att få ett objekt på en dator att se ut som om det fanns på en annan dator (Ibid).

Nätverkskommunikationslagret för distribuerade objektarkitekturer är egentligen ganska enkla till sin uppbyggnad. Det finns i grunden tre delar till denna arkitektur: objektservern, skelettet (*eng.* skeleton) och stumpen (*eng.* stub).

Objektservern är affärsobjektet som finns i mellanlagret. Just termen ”server” kan i detta sammanhang te sig lite konstigt, men för att skilja objektet i mellanlagret från de andra två så är det ett adekvat begrepp. Objektservern är en objektinstans som har ett eget unikt tillstånd (Ibid). Varje objektserver har matchande skelett- och stumpklasser som är byggda just för den typen av objektserver. Till exempel för det distribuerade affärsobjektet `Nyhet` skulle det alltså finnas en `Nyhet_Stub`-klass samt en `Nyhet_Skeleton`-klass. Som syns i bilden (exemplifierat med ett Person-objekt) nedan så finns objektservern och skelettet i mellanlagret och stumpen på klienten.



Figur 2.3 RMI-loop med ett Person-affärsobjekt. (Monson-Haefel, 2000, s. 12)

Stumpen och skelettet ansvarar för att få det att se ut som att objektservern, som ju ligger i mellanlagret, existerar lokalt på klientmaskinen (Ibid). Detta sker genom någon form av *remote method invocation*-protokoll (RMI). Det finns många former av RMI-protokoll. De används för att kommunicera metodanrop över ett nätverk. Varje instans av objektservern i mellanlagret hanteras av en instans av den matchande skelett-klassen. Skelettet lyssnar efter att stumpen skall skicka ett anrop till den. Detta sker på en viss port och IP-adress.

Stumpen finns alltså hos klienten och är sammankopplad med skelettet över nätverket. Man kan säga att stumpen agerar som ett slags surrogatobjekt på klienten för objektservern. Stumpen ansvarar för att via skelettet vidarebefordra alla metodanrop från klienten till objektservern (Ibid).

Figuren visar hur det går till när ett metodanrop kommuniceras från klienten till serverobjektet och tillbaka.

2.1.2.3 Hybrid mellan ORB och TP Monitor

Man kan se en CTM som en lyckad hybrid mellan en ORB och en TP Monitor (Ibid). Eftersom en CTM både tillhandahåller en distribuerad objektarkitektur som en ORB, samt en serverkomponentmodell och en robust miljö för transaktioner, resurshantering och feltolerans som en TP Monitor.

2.2 Mjukvaruarkitekturer – ramverk

Vi har nu behandlat distribuerade system och andra teknologier som är relaterade till Enterprise JavaBeans. De ger en nödvändig förförståelse för att förstå vad EJB är och hur det kan användas.

I detta avsnitt behandlar vi mjukvaruarkitekturer eller ramverk (*eng.* frameworks). Vi ger en introduktion till vad ett ramverk kan innebära rent generellt och ger exempel på olika sådana.

2.2.1 Generella ramverk

Objektorienterade ramverk är ett sätt att fånga det som är karakteristiskt för ett visst tillämpningsområde, exempelvis distribuerade system. Ramverk är en sorts ”skelett” som man kan utveckla en applikation utifrån (Lundberg, 1998, kap. 1).

Detta skelett fångar in de vanligaste kännetecknen inom ett visst tillämpningsområde och gör det möjligt att återanvända dem. När man arbetar inom ett tillämpningsområde där det finns ett ramverk är det förmodligen effektivare att använda det än att börja från början och göra allt själv. Om någon exempelvis har tagit fram ett ramverk för hur man utvecklar bokningssystem, så kan vanligtvis mycket vinnas på att använda detta.

2.2.1.1 Exempel

Ett ramverk kan vara en samling klasser som interagerar med varandra. Ramverket består då av två gränssnitt. Ett gränssnitt, *API* (Application Programmer's Interface), som utvecklaren anropar och ett annat som består av abstrakta klasser. Utvecklaren måste skapa egna klasser som ärver från ramverkets abstrakta klasser när han skall utveckla en egen produkt med hjälp av det.

De egenutvecklade klasserna specifierar ramverket för det aktuella projektet. Eftersom utvecklaren måste göra sin egen anpassning av ramverket krävs det av honom en djup förståelse av hur ramverket fungerar. Denna typ av ramverk kallas därför för *white box*-ramverk, eftersom de kräver en förståelse för ramverkets interna uppbyggnad (Lundberg, 1998, kap. 2).

2.2.1.2 Återanvändning

Mellan mjukvaruutvecklingsprojekt finns det mycket som det är möjligt att återanvända, både själva designen av systemet och koden för det.

De flesta utvecklare tycker att det är viktigast att återanvända designen. Orsaken till detta är att en bra design anses vara den viktigaste faktorn för att ett mjukvaruprojekt blir framgångsrikt. Ramverk kommer troligen att bli ett viktigt verktyg för återanvändning, både av kod och design (Lundberg, 1998, kap. 3).

2.2.1.3 Standardiserade ramverk

Ett standardiserat ramverk kan ge ett bra och generellt verktyg för mjukvaruutveckling. Genom att använda en standard så slipper man problemet med att olika delsystem använder olika ramverk och att de därför inte kan samverka på ett tillfredsställande sätt. CORBA-projektet är ett exempel på en framgångsrik standard för ett ramverk för distribuerade objektsystem (Lundberg, 1998, kap. 7).

2.2.2 Ramverk för komponentintegration: komponentarkitekturer

En kommande trend inom skapandet av komplexa system är användandet av ramverk för komponentintegration.Vad är då en komponent?

1996 hölls en europeisk konferens om objekt-orienterad programmering (EECOOP) där man enades om att definiera en komponent enligt:

"En mjukvarukomponent är en enhet med väl specificerade gränssnitt och enbart explicita kontext-beroenden. En mjukvarukomponent kan tas i drift fristående och kan sättas samman av tredje part." (Hansson, 1999, kap. 2.3).

Ramverk för komponentintegration föreskriver en designarkitektur, som gör att man kan använda tredjepartskomponenter när man skapar en applikation (Sousa och Garlan, 2000, s. 1). Sådana ramverk, som också kallas för komponentarkitekturer, blir allt viktigare för kommersiella system (Sousa och Garlan, 2000, s. 2).

För att göra det möjligt att använda tredjepartskomponenter måste designarkitekturen vara standardiserad.

Ramverket beskriver den övergripande arkitekturen för en applikation. Detta görs i termer av grundläggande komponenttyper. Ramverket beskriver även de gränssnitt (*eng*. interface), som dessa komponenttyper skall ha, dvs. gränssnittstandarder.

Ramverket beskriver också infrastrukturen som stödjer integrationen av de grundläggande komponenttyperna genom delade tjänster och kommunikationskanaler. Infrastrukturen är återanvändbar (Sousa och Garlan, 2000, s. 3).

Tjänster som en applikationsutvecklare antagligen vill ha av komponentarkitektur är till exempel att flera servrar kan dela på uppgifter utan att klienten vet var arbetet görs. Det berör alltså frågor om lastbalasering, klustring etc. (Hansson, 1999, kap. 2.5)

Genom att tillhandahålla en återanvändbar infrastruktur som plattform för komponentintegration av tredjepartsmjukvara minskar kodskrivandet som behöver göras för att dessa tredjepartskomponenter skall kunna kommunicera med vandra, Sousa och Garlan (2000, s3).

I nästa kapitel presenterar vi Enterprise JavaBeans, ett objektorienterat ramverk för flerskiktade distribuerade system och komponentintegration.

2.3 Enterprise JavaBeans

I introduktionen presenterade vi Enterprise JavaBeans som en komponentmodell för Component Transaction Monitors. I detta kapitel fördjupar vi oss i hur en sådan fungerar och vilka byggstenar EJB som ramverk består av.

Det viktiga att komma ihåg med EJB är att den är en *standardiserad* serverkomponentmodell för CTM:er. Detta innebär alltså att alla produkter, CTM:er, som stödjer denna komponentmodell kan använda sig av EJB:er som utvecklats i enlighet med denna standard. Standarden är beskriven i en löpande specifikation från SUN Microsystems (DeMichiel et. al., 2001; Matena & Hapner, 1999) uppdelad i olika versioner.

2.3.1 Historia

Microsoft var först med att introducera en fullvärdig CTM i form av sin Microsoft Transaction Server (MTS) 1996. Andra stora tillverkare av mjukvaruplattformar som till exempel BEA och IBM hade lovande produkter på gång, men alla låg på utvecklingsstadiet medan MTS redan fanns på marknaden (Monson-Haefel, 2000).

Ett annat problem för de icke-Microsoft-baserade produkterna var att de alla använde CORBA som tjänst för distribuerade objekt och ingen ytterliggare standard. CORBA i sig var inget stort problem, det är en öppen standard och kan användas på alla plattformar. Det stora problemet var att alla tillverkare hade sin egen komponentmodell. Detta skulle givetvis ha inneburit att komponenter som utvecklats för en CORBA-CTM inte skulle fungera i en annan (Ibid).

Object Management Group (OMG) som utvecklade CORBA-standarden höll därför på att utveckla en standardiserad komponentmodell. Utvecklingsarbetet med denna gick dock alltför långsamt. I alla fall för den framväxande CTM-marknaden där nu Microsoft hade total dominans.

När så SUN släppte sin specifikation av en standard för serverkomponenter 1997 så togs den emot med öppna armar av alla de som hade behov av CTM-system på andra operativsystem än Microsofts (Ibid).

SUN var också kloka nog att inte definiera ramverket på en för låg nivå. Så länge som de som utvecklade CTM:er för EJB såg till att alla de tjänster som komponenterna skulle ha tillgång till enligt specifikation fanns tillgängliga, så gjorde det ingenting hur detta implementerades på systemnivå (Ibid).

2.3.2 Övergripande arkitektur

Detta avsnitt beskriver den grundläggande arkitekturen för ett system byggt på EJB-ramverket. Avsnittet utgår ifrån den senaste EJB-specifikationen (version 2.0) av DeMichiel et. al. (2001) samt boken ”Enterprise JavaBeans” av Richard Monson-Haefel (2000). I designen av vår prototyp utgick vi från den tidigare specifikationen (version 1.1) av Matena (1999). Detta eftersom det i skrivande stund ännu inte finns några produkter som stödjer den senaste specifikationen då den ännu inte är fastställd.

Syftet med att utgå även från den senaste specifikationen är att öka uppsatsens intressevärde, eftersom den nya specifikation snart kommer att vara den som används. Det ökar också användarvärdet av rapporten då den visar på kommande möjligheter med ramverket.

2.3.2.1 Introduktion

I avsnittet om generella ramverk tog vi upp exemplet med ett tänkt ramverk för bokningssystem. I detta avsnitt följer vi upp detta exempel och använder det för att belysa hur ett sådant hypotetiskt ramverk skulle se ut i EJB. Vi utökar exemplet till att innefatta ett bokningssystem för rum på ett hotell.

Vi använder detta exempel till att visa hur s.k. *enterprise-bönor* (*eng.* enterprise beans) distribueras som affärsobjekt, vilket är själva fundamentet i EJB-ramverket (Monson-Haefel, 2000).

Vår presentation av EJB är mycket övergripande och går inte in på många av de, ur ett tekniskt och programmeringsmässigt perspektiv, intressanta detaljerna av ramverkets uppbyggnad. Att göra detta skulle vara att grovt överskatta omfattningen av denna uppsats. För att sätta det hela i ett sammanhang kan sägas att den senaste EJB-specifikationen (DeMichiel et. al., 2001) är på över 500 sidor. För den intresserade hänvisar vi därför till detta dokument, samt rekommenderar även varmt Richard Monson-Haefels bok (Monson-Haefel, 2000).

2.3.2.1.1 Komponentmodellen

I förra kapitlet talade vi om ramverk för komponentintegration. Där definierades också vad en komponent var. Enterprise JavaBeans serverkomponenter finns i två distinkta typer: *entitetsbönor* (*eng.* entity beans) och *sessionsbönor* (*eng.* session beans), som definieras i den första specifikationen.

Entitetsbönor modellerar affärskoncept och kan uttryckas i form av subjektiv. En entitetsböna kan alltså till exempel representera en hotellgäst, ett hotellrum, en nyckel och så vidare. Entitetsbönor representerar alltså objekt i verkliga livet (Ibid). Dessa objekt finns ofta lagrade i någon form av databas i ett system.

Sessionsbönor är däremot en utökning av klientapplikationen i ett system (DeMichiel et. al., 2001). En entitetsböna som en Hotellrumsböna till exempel, förser oss med

metoder för att göra saker med denna entitet (rummet), kanske att ändra antalet sängar i det. Den säger däremot ingenting om det sammanhang i vilket detta görs (Monson-Haefel, 2000). Om vi till exempel skall boka in en hotellgäst på ett visst rum ett visst datum så ingår det antagligen flera aktiviteter i denna process. Vi skall ha reda på uppgifter om gästen, kolla om rummet är ledigt det aktuella datumet, boka in gästen om så är fallet och ta betalt för vistelsen.

För att hantera sådana här skeenden eller affärsprocesser finns sessionsbönan. I vårt fall skulle vår aktivitet kunna representeras av en Bokningsböna, som skulle använda sig av både en Hotellgästböna och en Hotellrumsböna med mera.

En sessionsböna är, till skillnad från entitetsbönan, inte beständig. De lagras inte i någon form av beständig lagringsmedia, som till exempel en databas. De utför endast de aktiviteter som de inbegriper och är sedan klara. En sessionsböna kan däremot ha en indirekt eller direkt inverkan på databasen, då den använder sig av entitetsbönor, som ju lagras i databasen. Sessionsbönor kan också ha direkt åtkomst till en databas, men de lagrar alltså aldrig information om *sig själva* där.

2.3.2.1.1.1 EJB-bönor är konfigurerbara komponenter

EJB bönor kan definieras som komponenter i en transaktionsorienterad distribuerad applikation (DeMichiel et. al., 2001, s.41).

Ett viktigt kännetecken för en EJB-böna är att den kan anpassas vid driftsättning av ett system (*eng.* deployment). Detta sker genom att man definierar olika ”miljöattribut” (*eng.* deployment properties). En samling bönor kan alltså paketeras och anpassas för olika driftsmiljöer. Detta gör att EJB-ramverket uppfyller kraven på att mjukvarukomponenter kan tas i drift fristående och kan sättas samman av en tredje part, vilket togs upp i avsnittet om ramverk för komponentintegration.

En EJB-böna representarar alltså affärsobjekt och affärslogik. Transaktionshantering och säkerhet är helt separerade från bönan och tas om hand av den CTM som finns i driftsmiljön (Ibid, s. 41-42).

2.3.2.1.1.2 En komponentarkitektur

EJB-arkitekturen är alltså en komponentarkitektur. Tillämpningsområdet är utvecklingen och installationen av distribuerade komponentbaserade affärsapplikationer (Ibid, s. 25)

Målet med EJB-arkitekturen är att den skall bli den standard som används när man bygger distribuerade objektorienterade affärsapplikationer i Java (Ibid, s. 29).

2.3.2.2 Bönornas konstruktion

För att förstå enterprisebönornas utformning beskriver vi här översiktligt hur man skapar en sådan böna.

För att realisera en enterpriseböna så måste man definiera två *interface*, samt en eller två klasser. Interface är klasser som enbart definierar vilka metoder som utvecklaren måste ha med i sin version av klassen. De definierar *inte* hur dessa metoder skall implementeras i sig själva. Interface-analogin är en speciell egenskap för Java som programmeringsspråk (Gosling 2000).

Remote interface – detta interface definierar vilka bönans affärsmetoder är och presenterar dessa för övriga världen, dvs. de klienter som använder bönan.

Home interface – detta interface definierar bönans så kallade *livscykelmetoder*, vilka innefattar metoder för att skapa nya bönor, ta bort bönor och att hitta bönor.

Bönklassen – denna klass är där den verkliga implementeringen av bönans affärsmetoder finns. Det intressanta ur ett programmeringsperspektiv är dock att bönklassen vanligen inte implementerar bönans home- eller remote-interface. Den måste dock ha metoder som matchar sigaturerna av de metoderna som är definierade i remote-interfacet och även några av de metoder som finns i home-interfacet (Monson-Haefel, 2000). Detta kan verka konfunderande för en javaprogrammerare i början.

Primärnyckeln – primärnyckelklassen är en mycket simpel klass som entitetsbönor använder för att mappa sin identitet mot en databas.

Anledningen till att bönklassen inte implementerar de ovan nämnda interfacen beror på att klienten aldrig interagerar med bönklassen *direkt*. Interaktionen sker alltid via metoderna i home- och remote-interfacen (Ibid). Även en böna som anropar en böna går via interfacen.

2.3.2.2.1 Bean Container

Det sker mycket interaktion mellan en böna och dess server. Denna interaktion hanteras av en s.k. *"container"*. Denna är ansvarig för att presentera ett uniformt gränssnitt mellan en böna och dess server (DeMichiel et. al., 2001, s. 56).

Ofta används termerna "server" och "container" som om de vore samma sak, vilket inte riktigt är fallet. Sammanblandningen sker oftast på grund av att alla server-tillverkare också utvecklar sin container och de kan då te sig som om de vore samma enhet. För en systemutvecklare märks skillnaden mycket sällan och sammanblandningen av uttrycken är därför ingen större katastrof (Monson-Haefel, 2000).

En container är i vilket fall ansvarig för att skapa nya instanser av bönor och att se till att dessa lagras korrekt av servern och så vidare. Container(server)-tillverkare förser utvecklaren med många verktyg som hjälper till att skapa mappningen mellan entitsbönor och databaser. Andra verktyg hjälper till med att skapa en mängd genererad kod utifrån utvecklarens egen kod. Det är sen denna generarde kod som utför det verkliga arbetet bakom kulisserna (Ibid). Det är också denna kod som sedan i praktiken implementerar de två interfacen och detta är alltså anledningen till att utvecklarens egen klass inte måste göra det (Ibid). Container:n genererar också stub-kod (som vi behandlade i avsnittet om distribuerade objektarkitekturer) för klienten.

I figuren nedan visas hur de olika bitarna av en enterpriseböna fördelas mellan klienten och EJB-containern efter att den driftsatts (deploy:ats) i EJB-servern. Vi beskriver driftsättningsprocessen längre fram i texten.

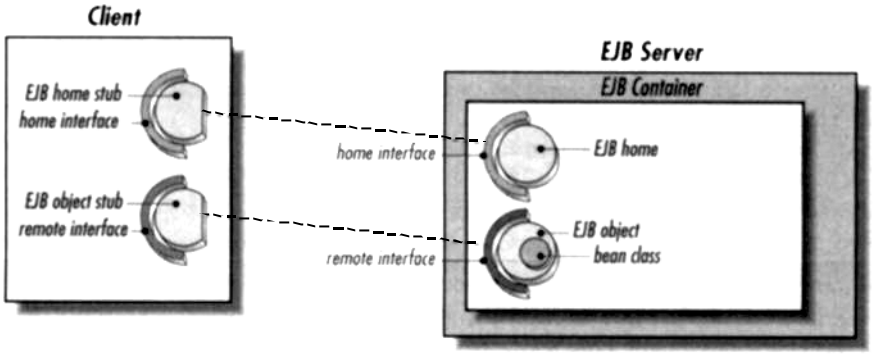


Bild 2.4 EJB-arkitekturen (Monson-Haefel, 2000, s. 39).

2.3.2.3 Entitetsbönor

Entitetsbönor (*eng.* entity beans) är alltså de komponenter som representerar affärsobjekten för det system som utvecklas. Dessa bönor är alltså *beständiga*, de lagras alltså i någon form av media, oftast en databas (DeMichiel et. al., 2001, s. 48).

Det finns två typer av entitetsbönor och olikheterna består i hur lagringen av den data de representerar går till. När en förändring i en entitetsbönas tillstånd sker så resulterar detta i en ändring i databasen.

Containern håller reda på när detta inträffar, men själva lagringsprocessen kan antingen ske helt automatiskt eller med viss hjälp av utvecklaren (Monson-Haefel, 2000).

Fördelen med entitetsbönor gentemot direkt databasåtkomst är att det blir mycket lättare för utvecklaren att göra ändringar på en böna än att skriva en komplicerad SQL-sats (Ibid, s. 154). Det enda utvecklaren behöver göra är att anropa en metod på det aktuella objektet och så sköter servern om uppkopplingen mot den faktiska databasen etc.

2.3.2.3.1 Container-hanterad beständighet

Bönor vars datalagring sker helt med hjälp av EJB-servern kallas för *CMP-bönor*. CMP står för Container-Managed Persistence (container-hanterad beständighet). Dessa bönor är generellt sett de enklaste att utveckla, men de kräver mer arbete under driftsättningsfasen.

Container-hanterad beständighet innebär att man endast definierar vilka data en entitet skall ha under utvecklingen, man skapar inte då motsvarande databasstruktur. Detta gör att utvecklaren endast behöver skapa bönan utifrån hur den bäst passar in i affärslogiken (Ibid). Hur bönan skall fungera gentemot databasen bestämt vid driftsättningen.

Den största fördelen med container-hanterad beständighet är att bönan kan definieras oavhängigt från databasen, vilket givetvis gör den lättare att flytta mellan olika miljöer mellan vilka databasstrukturen kan variera kraftigt (Ibid.).

Nackdelen består i att de kräver mycket sofistikerade verktyg vid driftsättningen för att koppla bönans fält gentemot databasen, vilket kan ställa till problem om en böna skall mappas mot en mycket komplex datastruktur i databasen. En böna kanske i ett visst fall representeras av en sammanslagning (*eng.* join) av flera tabeller och ett inedakvat verktyg kan då ställa till problem.

2.3.2.3.2 Bönhanterad beständighet

I vissa fall kanske de datastrukturer som behöver representeras av en entitetsböna är mycket komplexa, som vi talade om tidigare. Entiteten kanske definieras av komplexa SQL-joins, en kombination av olika databaser, eller andra resurser som äldre (legacy) system, vilka kanske inte kan kommas åt via SQL.

Entitetsbönor som kännetecknas av sådana omständigheter bör implementeras med så kalad *bönhanterad beständighet* (*eng.* bean-managed persistence). Dessa bönor kallas därav för *BMP-bönor*.

BMP-bönor måste alltså användas när driftsättningsvertygen inte är tillräckliga för den givna systemutvecklingsuppgiften (Ibid, s. 179).

Nackdelarna med BMP-bönor är att de generellt sett kräver mer arbete för att definiera bönan. Utvecklaren måste själv förstå och utveckla all logik för att skapa, uppdatera och ta bort data som är associerad med en entitet (Ibid, s. 180).

Den andra stora nackdelen med denna böntyp är att bönhanterad beständighet gravt inverkar på komponenternas portabilitet, eftersom de gör bönorna anpassade för just den givna situationen och driftsmiljön (Ibid.).

Det arbete som måste göras i praktiken är att utvecklaren för varje av de speciella EJB-metoderna i en entitetsböna måste skriva kod som till exempel hämtar upp den aktuella datan från en tabell och sätter de olika fälten i bönan. Utveckling av BMP-bönor liknar därför mer traditionell systemutveckling, men utvecklaren drar fortfarande nytta av alla de tjänster som finns i ramverket (Ibid).

2.3.2.4 Sessionsbönor

En sessionsböna representerar alltså ett arbetsflöde (Ibid, s. 219). De kan utföra sitt arbete genom att använda sig av den delade datan. Detta kan ske dels genom att en sessionsböna använder en eller ett flertal entitetsbönor samt också genom att ha direkt åtkomst till olika datalager (Ibid).

Hur är då sessionsbönorna relaterade till entitetsbönorna? Man kan likna situationen vid den vid en teater där sessionsbönorna står för manus och regi och där entitetsbönorna och övriga resurser är skådespelare och rekvisita. Sessionsbönorna hanterar alltså all interaktion mellan data i ett EJB-system.

Sessionsbönor kan även de delas upp i två typer: *stateless session beans* (tillståndslösa sessionsbönor) och *stateful session beans* (sessionsbönor med tillstånd) (DeMichiel et. al., 2001, s. 70-71). Vi använder även i fortsättningen den engelska benämningen på dessa bönor för enkelhetens skull.

En prestandamässig fördel med att klienterna så långt som möjligt enbart interagerar med EJB-servern genom sessionsbönor är att detta drar ner på nätverkstrafiken. Istället för att själv hantera alla bönor som ingår i ett moment (och också vara uppkopplad mot alla dessa distribuerade objekt) så låter man sessionsbönan sköta denna interaktion vilken då sker direkt i servern mellan de ingående bönorna (Monson-Haefel, 2000, s. 44). Dessutom drar det ned på minnesanvändningen och processorutnyttjandet i klientprogramvaran eftersom denna då inte behöver ha lika många objekt och stub-objekt instantierade.

2.3.2.4.1 Sessionsbönor utan tillstånd

En sessionsböna utan tillstånd (*eng.* stateless session bean) används för att representera en samling relaterade tjänster, som bönan kan utföra åt klienten (Ibid, s. 220).

Det som menas med att de inte har något tillstånd är att de inte behåller någon form av information mellan anropen till den. Om vi har en sessionsböna som till exempel hämtar upp en aktiekurs från en databas, så behöver den göra bara just det. Den har heller ingen relation med de eventuellt andra metoder som bönan tillhandahåller (Ibid).

Därför måste all information som bönans metoder behöver skickas med anropet till den. Den tillståndslösa bönan kan liknas vid de tjänster som de traditionella procedurella transaktionssystem (som t.ex. de TP-monitorer vi talade om i inledningen) använder sig av.

Det positiva med den tillståndslösa sessionsbönan är att de är relativt enkla att utveckla och att de också är mycket effektiva rent prestandamässigt (Ibid, s. 220). Detta beror dels på att de inte är beständiga och att de inte heller är knutna till en viss klient och därför inte tar upp lika mycket resurser på servern (Ibid.).

Sådana här bönor används ofta för tjänster som är generella och återanvändbara i hela systemet. Alla tjänster eller aktiviteter som kan utföras i ett metodanrop är kandidater för att realiseras med en tillståndslös sessionsböna (Ibid).

2.3.2.4.2 Sessionsbönor med tillstånd

En sessionsböna som har tillstånd (*eng.* stateful session bean) används för att representera processer där det är tvunget att knyta en viss böninstans till en viss klient. Detta kallas att de upprätthåller ett konversationstillstånd (*eng.* conversational state) med sin klient (DeMichiel et. al., 2001, s. 69).

Detta innebär att en och samma instans aldrig används av samma klient (Monson-Haefel, 2000, s. 44). I avsnittet om komponentmodellen tog vi exemplet med en Bokningsböna. Det är ju ganska självklart att en viss bokningsprocess är knuten till en viss kund. Hade inte så varit fallet, så hade kanske en gäst bokats in på en annan gästs hotellrum.

En Bokningsböna kanske har de två metoderna `bokaRum` och `beställ`. Klienten presenterar kanske användaren först med ett val där hon fyller i vilken typ av rum hon vill ha samt vilken dag hon skall bo på hotellet. Därefter kontrolleras bokningen genom att klienten anropar metoden för att boka rum. Går detta bra så kommer sessionsbönan att ha noterat att detta gått bra och kanske fått ett bokningsnummer knytet till det kundnummer som gästen har i systemet.

Därefter presenteras användaren med en skärm där hon ombedes fylla i betalningssätt och annan information. Därefter anropar klientprogramvaran beställmetoden i sessionsbönan. På EJB-servern har nu instansen av sessionsbönan den information den behöver för att genomföra en beställning. Klientprogramvaran får sedan tillbaka information om hur beställningen gått.

Detta exempel illustrerar hur en affärsprocess knyts till en viss klient och hur sessionsbönan upprätthåller ett tillstånd om hur denna process fortlöper. Programmeringsmässigt innebär detta att en sessionsböna kan dela data mellan metodanrop (Ibid).

Den stora skillnaden på serversidan mellan denna böntyp och den tillståndslösa sessionsbönan är att varje stateful sessionsböna behöver en egen instans på servern för varje klient, medan en tillståndslös böna kanske bara behöver ha en handfull instanser för att serva hundra- eller tusentals klienter (Ibid, s. 45).

2.3.2.5 Meddelandedrivna bönor

Den senaste EJB-specifikationen (DeMichiel et al. 2001, s 48-49) beskriver en helt ny typ av böna: den meddelandedrivna bönan (*eng.* message-driven bean). Denna böna representerar inte heller den beständig data. De meddelandedrivna bönorna är tillståndslösa, transaktionsdelaktiga serverkomponenter som hanterar asynkrona meddelande via teknologin *Java Message Service* (JMS).

Asynkrona meddelanden låter tillämpningar att kommunicera genom att utbyta meddelande som gör avsändaren oberoende av mottagare. Avsändaren skickar alltså iväg sitt meddelande och behöver inte vänta på att en mottagare skall ta emot eller behandla meddelandet (Ibid).

Meddelandedrivna bönor kan användas för att till exempel integrera ett EJB-baserat system med ett legacy-system där direktkommunikation inte är möjlig på grund av exempelvis långa svarstider.

2.3.3 EJB:s resurshantering och primära tjänster

Vi har nu introducerat den grundläggande arkitekturen bakom ramverket EJB: hur enterprisebönor är uppbyggda, hur de fungerar och hur de är relaterade till EJB-servern.

Dessa komponenter definierar en gemensam modell för distribuerade serverkomponenter i component transaction-monitorer (CTM).

En anledning till att en CTM är en så bra plattform för distribuerade objekt är för att de gör mer än bara distribuerar objekt (som t.ex. CORBA gör). De hanterar också de resurser som de distribuerade objekten använder sig av (Ibid, s. 49). För att kunna hantera stora mängder av objekt och användare, så måste en CTM vara mycket kapabel att hantera hur de distribuerade objekten använder sig av minne och processorkraft (Ibid).

Förutom att vara bra på att hantera distribuerade objekt, så tilhandahåller de många tjänster som gör det enklare för klienterna att använda sig av objekten på rätt sätt. En CTM stödjer oftast följande sex primära tjänster: *samtidighet, transaktionshantering, beständighet, objektdistribuering, namngivning* och *säkerhet*. Dessa tjänster gör att utvecklaren får den rätta sortens infrastruktur för att kunna utveckla ett framgångsrikt flerskiktat system (Ibid.)

2.3.3.1 Resurshantering

Ju mer användare som är klienter till ett system, desto fler distribuerade objekt och resurser behövs. Vid något tillfälle kommer antagligen möjligheterna att hantera denna ökande mängd att ta slut om man hela tiden skapar nya objekt.

EJB stödjer två mekanismer för att göra det enklare att hantera stora mängder av bönor när ett system är igång: *instanspooler* (*eng.* instance pooling) och aktivering (*eng.* activation).

Vi beskriver här översiktligt dessa bägge mekanismer.

2.3.3.1.1 Instanspooler

Att ha resurser i så kallade *pooler* är ingenting nytt. En ofta använd teknik är att hålla databasuppkopplingar i en pool, så att affärsobjekten i ett system kan dela på datbasåtkomstmöjligheterna. Istället för att hela tiden skapa nya uppkopplingar för varje objekt så delar alla objekten på de som finns i poolen. Detta sparar resurser för hela systemet, eftersom skapandet av nya uppkopplingar ofta kostar en hel del resurser (Ibid, s. 50).

En CTM kan använda sig av poolning av serverkomponenterna, vilket EJB gör. Eftersom alla klienter till ett EJB-system endast interagerar med remote-interfacen, så vet de inte något om den faktiska böninstansen på servern och kommer inte heller åt denna direkt på något sätt (Ibid.).

Eftersom klienterna aldirg kommer åt bönorna direkt så finns det ju inget större skäl till att ha en separat kopia av en böna för varje klient. Istället kan servern hålla sig med ett lämpligt antal instanser och sedan kopiera in och ur data från dem allt eftersom de behövs (Ibid.).

På grund av denna mekanism har alla bönor en livscykel och tre tillstånd i denna livscykel: inget tillstånd (*eng.* no state), poolat tillstånd (*eng.* pooled state) och redotillstånd (*eng.* ready state). Denna livscykel illustreras i bilden nedan.

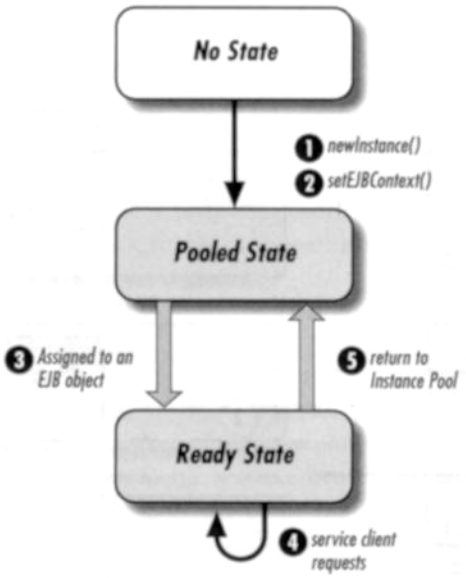


Bild 2.5 En böninstans livscykel (Monson-Haefel, 2000, s. 53).

När en böninstans inte har något tillstånd har den inte blivit instantierad än. Detta tillstånd definieras enbart för att definiera början och slutet på en böninstans livscykel.

I poolat tillstånd har bönan blivit instantierad men är inte ännu tilldelad till något *EJB-objekt*. Ett EJB-objekt kan sägas vara en mellanhand på servern mellan klientens remote-interface och den faktiska böninstansen (Ibid, s. 36).

I redotillståndet har böninstansen associerats med ett EJB-objekt och är redo att svara på klientens anrop av dess affärsmetoder.

Genom att använda sig av instanspoolning kan EJB-containern hantera ett stort antal klienter med ett relativt litet antal faktiska instanser. Containern kan också reglera

antalet instanser i poolen allt eftersom belastningen på servern ökar eller minskar (Ibid, s. 55).

2.3.3.1.2 Aktivering och passivering

Endast entitetsbönor och tillståndslösa sessionsbönor hanteras med instanspooler. Eftersom en stateful sessionsböna upprätthåller ett konversationstillstånd med sin klient, måste varje böninstans vara unik.

För att hushålla med resurserna för även denna typ av böna, så använder sig containern av en så kallad *aktiverings- och passiveringsmekanism*. När en EJB-server behöver dra ned på resursutnyttjandet så kan den välja att ta bort stateful sessionsbönor från minnet (Ibid).

För att göra detta så behöver bönans tillstånd sparas ned till ett beständigt lagringsmedia. När sedan klienten anropar en metod på EJB-objektet så skapas en ny instans av bönan och det sparade tillståndet överförs på denna, nya instans (Ibid).

2.3.3.2 Primära tjänster

Detta avsnitt fokuserar på de tjänster som ger ett ökat värde för utvecklingen av distribuerade applikationer. Eftersom utvecklaren kan använda sig av dessa, så får han eller hon väldigt mycket funktionalitet ”gratis”, som annars antagligen skulle ta lång tid att utveckla.

2.3.3.2.1 Samtidighet

EJB-servrar hanterar samtidighet automatiskt. Ramverket förhindrar samtidig åtkomst till böninstanser. Flera klienter kan ha tillgång till samma *EJB-objekt*, men enbart en klient åt gången kan komma åt den *faktiska böninstansen*. De andra får vänta tills den första klienten är klar. Och om metodanropet till en böna är del av en större transaktion, så kan inte instansen användas förrän alla delar av transaktionen är klara (Ibid, s. 59).

Samtidigheten i EJB är bara applicerbar på entitetsbönor. Stateful sessionbönor är knutna till bara en klient, så de behöver inte hantera detta eftersom problemet inte kan uppstå. Tillståndslösa sessionsbönor bryr sig inte om ifall flera klienter anropar den eftersom den inte har något tillstånd som klienterna behöver dela på (Ibid, s. 58).

2.3.3.2.2 Transaktionshantering

En transaktion är en uppsättning instruktioner som utförs tillsammans (Ibid, s. 62). Transaktioner är *atomiska* till sin natur. Detta innebär att alla instruktioner i en transaktion måste utföras korrekt för att transaktionen skall anses ha lyckats.

En EJB-server övervakar alla transaktioner och ser till att de utförs korrekt. Detta sker automatiskt och utvecklaren behöver inte skriva någon kod för att hantera en bönas delaktighet i en transaktion (Ibid.). Det enda som behövs är att bönans transaktionsegenskaper sätts när bönans driftsätts. Detta är fallet med så kallade containter-hanterade transaktioner (*eng.* containter-managed transactions). Systemutvecklaren kan dock aktivt välja att hantera transaktionerna själv genom bönhanterade transaktioner (*eng.* bean-managed transactions).

2.3.3.2.3 Beständighet

Denna tjänst är endast för entitetsbönor, eftersom de är beständiga till sin natur. Så fort något sker med en bönas tillstånd, så ser EJB-servern till att lagra denna förändring till ett beständigt lagringsmedium (oftast en databas).

Den faktiska implementationen av hur detta görs är specifik för varje tillverkare av EJB-servrar, men eftersom tjänsten ligger på en abstraktionsnivå högre upp så behöver inte utvecklaren ta hänsyn till detta faktum (Ibid, s. 63).

2.3.3.2.4 Objektdistribuering

Det finns idag i huvudsak tre tjänster för att hantera distribuerade objekt. De är CORBA, Java RMI och DCOM. Alla dessa tjänster använder sig av olika nätverksprotokoll, men de uppnår i praktiken samma mål: *platsoberoende* (Ibid, s. 67). DCOM används främst i Microsoft-miljöer och stöds knappt på någon annan plattform. CORBA är inte knutet till något operativsystem eller programmeringsspråk över huvud taget och är därför ett bra val när man vill integrera olika system som har utvecklats i olika miljöer (Ibid). Java RMI är en abstraktion för att hantera distribuerade objekt. Det är i sig inte knutet till något specifikt protokoll, men dess användning har i praktiken varit begränsad till *Java Remote Method Protocol* (JRMP), vilket har lett till att det mest har använts för enbart rena Java-applikationer (Ibid). Nyligen har man introducerat möjligheten att köra Java RMI över CORBAs protokoll IIOP.

EJB-klienter behöver inte bry sig om vilket protokoll de använder sig av, det enda de känner till är bönans remote- och home-interface. Så länge som EJB-servern stödjer EJB-klientens vy så kan vilket protokoll för distribuerade objekt som helst användas.

För att det skall fungera att köra system från olika tillverkare kräver EJB 2.0-specifikationen att tillverkarna stödjer ett protokoll baserat på CORBA/IIOP. Dessutom kan de välja att stödja även andra protokoll (DeMichiel et al.,2001, s. 49). Det skulle till exempel till och med vara möjligt för en EJB-klient att använda sig av DCOM, som protokoll (Monson-Haefel, 2000). Bilden nedan illustrerar hur klienternas vyer kan stödjas av olika protokoll.

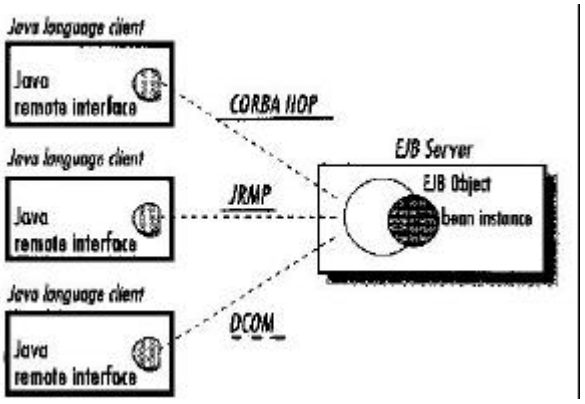


Bild 2.6 EJB-klienter i Java stödda av olika protokoll livscykel (Monson-Haefel, 2000, s.68).

EJB gör det också möjligt för servrar att stödja klienter skrivna i ett annat språk än Java. Ett exempel på detta är mappningen mellan EJB och CORBA som har tagits fram av Sun (Krishnan, 1999). Denna gör det möjligt för klienter som använder sig av CORBA att komma åt och använda sig av enterprisebönor. En CORBA-klient kan, på grund av dess programspråksoberoende, skrivas i praktiskt taget vilket språk som helst, t.ex. C++, Ada, Smalltalk eller till och med COBOL. Kanske kommer också en

mappning mellan EJB och DCOM att komma till stånd. Detta kommer i så fall göra det möjligt att skriva klienter i t.ex. Visual Basic eller Delphi för att komma åt enterprisebönor (Ibid). Bilden nedan visar hur klienter skrivna i olika språk kan komma åt en EJB-server.

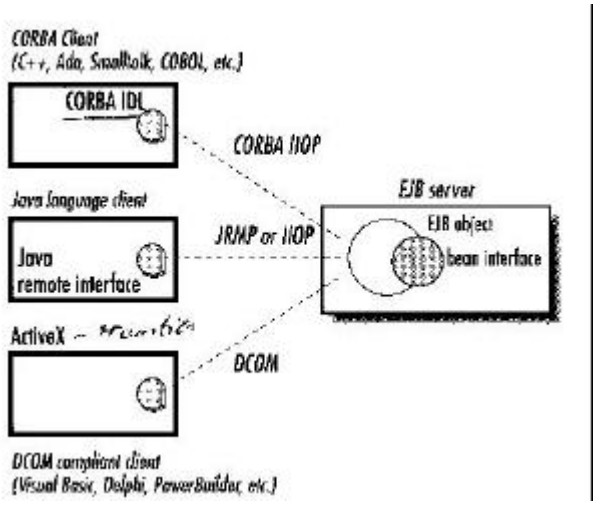


Bild 2.7 EJB som används av olika distribuerade klienter livscykel (Monson-Haefel, 2000, s. 69).

De olika protokollen som kan användas mellan klienten och servern är bra på olika saker. Vi går i denna uppsats inte närmare in på en jämförelse mellan dessa fördelar och nackdelar. Generellt kan dock sägas att Javas JRMP är bra i en homogen miljö med enbart Java-klienter, men i en heterogen systemmiljö är CORBA IIOP att föredra (Ibid, s. 69).

2.3.3.2.5 Namngivning

Alla tjänster för distribuerade objekt, som CORBA eller EJB, använder sig av en namngivningstjänst av något slag. En namngivningstjänst används för att klienterna skall kunna ha en möjlighet att lokalisera de distribuerade objekten över nätverket (Ibid.).

För att kunna göra detta så måste en namngivningstjänst ha två saker: objektbindning (*eng.* object binding) och en sök-API (*eng.* lookup API). Objektbindning innebär att man ger ett distribuerat objekt ett namn som det identifieras med. I vårt exempel med Hotellrum-bönan så skulle dess home-interface kunna bindas till namnet ”minabeans.Rum” eller bara ”hotellrum”.

Genom att använda sig av en sök-API så kan klienten komma åt namgivningstjänsten och där söka efter objekt på deras olika namn. Klienten ansluter sig alltså till en distribuerad tjänst och ber tjänsten om att få tillgång till en remote-referens till ett visst objekt (Ibid).

För Java-klienter används *JNDI* (Java Naming and Directory Interface) som sök-API. JNDI stödjer nästan alla sorters namngivnings- och katalogtjänster. En EJB-server som stödjer klienter som inte är skrivna i Java måste också stödja en namngivningstjänst som kan användas med det aktuella protokollet för distribuerade objekt (Ibid).

2.3.3.2.6 Säkerhet

Det finns tre olika sorters säkerhet som en Enterprise JavaBeans-server kan stå för: *autentisering*, *åtkomstkontroll* och *säker kommunikation*. Enbart åtkomstkontroll behandlas dock i EJB-specifikationerna (DeMichiel et al., 2001; Matena & Hapner, 1999).

Autentisering innebär enkelt sagt att man identifierar användare och avgör om dessa har rätt att komma åt en viss resurs. Denna säkerhetsmekanism är dock ganska otillräcklig eftersom den enbart kräver ett användarnamn och ett lösenord. Om detta är den enda säkerhetsnivån i ett system så kan det vara svårt att värja sig mot att systemet används av fel personer (Monson-Hafel 2000, s. 71).

Genom att lägga på olika säkerhetspolicy på användarna så kan man bättre kontrollera vilka resurser de får komma åt inom ett system. Detta är vad åtkomstkontroll innebär. Vissa användare kan till exempel bara få ta del av information, men inte ändra den (Ibid).

Genom att kryptera kommunikationen mellan en klient och en server så får systemet en säker kommunikation, vilket gör att man inte kan få säkerhetsläckor om någon avlyssnar nätverkskommunikationen.

De flesta EJB-servrar stödjer säker kommunikation, ofta genom *SSL* (Secure Socket Layer) och även någon mekanism för autentisering, men det är som sagt inte ett krav i specifikationerna.

3 Metod

Då vår uppsats fokuserar på ett ramverk för systemutveckling har vi valt att kombinera resultat från ett flertal olika vetenskapliga metoder för att kunna undersöka fenomenet både utifrån en mer teoretisk, samt en mer praktisk utgångspunkt. Vi använde oss i uppsatsen av fem olika sätt att samla in information och dessa användes under fem faser som utfördes i tur och ordning.

Undersökningens fem faser var:

1. En omfattande litteraturstudie under hösten 2000 och delvis under våren 2001.
2. Intervjuer med personer som arbetar med flerskiktade distribuerade system, men ännu inte arbetar med EJB. Syftet var här att få fram vilka kriterier som är viktiga vid en jämförelse mellan utveckling av flerskiktade, distribuerade system utvecklade med eller utan EJB.
3. En kvalitativ jämförelse mellan en EJB-prototyp och ett referensalternativ. De två prototyperna utvecklades i Java. Det ena med hjälp av EJB och det andra utan. Båda prototyperna utvecklades med hjälp av en etablerad objektorienterad systemutvecklingsmetod. Resultatet jämfördes sedan mot de kriterier som ställts upp i fas två.
4. Intervjuer med personer som arbetar med EJB om vad de tycker är fördelar och nackdelar med att arbeta med EJB jämfört med att utveckla i Java utan att använda ett standardiserat ramverk. Dessa intervjuer utfördes för att kunna jämföra våra egna resultat från prototyputvecklingen med andras erfarenheter av att utveckla med EJB.
5. En kvantitativ studie av en EJB-FAQ (Frequently Asked Questions) på Internet. Denna syftade till att få fram vilka problem som är vanligt förekommande bland EJB-utvecklare.

Att använda sig av flera olika insamlingsmetoder rekommenderas av Easterby-Smith et al. (1991). De nämner dock att det är viktigt att använda dem med eftertanke. Vi kommer därför motivera våra val av metoder.

Nedan följer en presentation och motivering till val av våra metoder, samt en beskrivning av tillvägagångssättet vid metodernas utnyttjande. Vi beskriver också, i fallet med intervjuerna, motiveringen till valet av försökspersoner.

3.1.1 Litteraturstudie

Enterprise JavaBeans är en relativt ny teknologi. Den första specifikationen från SUN släpptes inte förrän 1997. Efter det dröjde det ett år eller två innan verkligt färdiga EJB-servrar kom ut på marknaden. På grund av detta finns det därför inte så många böcker ute på marknaden som handlar om EJB i sig. Detta gjorde det dock ganska lätt att sovra bland det existerande utbudet. Vi har studerat en större mängd av de existerande böckerna på marknaden och gjorde en översikt på dessa.

Parallellt med detta letade vi resurser på Internet och kom då i kontakt med några webbplatser som handlade om EJB. Där hittade vi diverse artiklar och rapporter som vi studerat.

Förutom att läsa om EJB, så studerade vi också litteratur om UML-baserade utvecklingsmetoder, distribuerade system, klient/server-applikationer, samt artiklar

om andra och relaterade teknologier som CORBA och RMI. Vi har även läst mycket i de officiella specifikationerna för EJB från SUN Microsystems (DeMichiel et al., 2001; Matena & Hapner, 1999).

Då vi redan under hösten hade läst in oss på den systemutvecklingsmetod vi använt oss av ägnade vi våren 2001 till att läsa främst böcker om EJB parallellt med att vi genomförde prototyputvecklingen.

Under hela arbetets gång ägnade vi också relativt stor tid till att studera dokumentation till den EJB-server som vi utvecklade vårt system för.

Fokus på litteraturstudierna var att kunna framställa ett teoretiskt bakgrundsmaterial som kunde användas för att analysera EJB som ett ramverk för systemutveckling och relatera detta till andra generella och specifika ramverk.

3.1.2 Intervjuer

Undersökningen omfattade två omgångar av kvalitativa intervjuer, vilka utgjorde fas två och fyra i vår undersökning.

Våra kvalitativa, ostrukturerade intervjuer är utförda i enlighet med Jan Trosts bok ”Kvalitativa Intervjuer” (Trost, 1997).

Motiveringen till att använda kvalitativa intervjuer är att de är bra på att hitta mönster och att förstå hur människor resonerar (Trost, J. 1997, s. 15-16). De mönster vi söker efter i det här fallet är vilka olika bedömningsgrunder olika systemutvecklare har när de bedömer nya teknologier och arbetssätt. Det som utmärker kvalitativa intervjuer är blan annat att man ställer frågor, som man får komplexa och innehållsrika svar på (Ibid, s. 7).

Vid intervjuer kan man välja att bara använda fördefinierade frågor, de är då mycket strukturerade (Ibid, s. 21). Vi har valt att utgå från vissa grundläggande frågor och sedan ställa följdfrågor. Detta innebär att intervjuernas standardiseringsgrad blir låg, då vissa följdfrågor endast ställs till vissa respondenter.

3.1.2.1 Intervjuomgång 1

Syftet med de första intervjuerna i fas två av studien var att kartlägga de kvaliteter som systemutvecklare kan tänkas använda för att bedöma ett ramverk för utveckling av distribuerade, flerskiktade system. Resultatet från dessa innebar att vi kunde vi sätta upp en rad kvalitativa kriterier för att bedöma slutresultatet av vår systemutveckling. Dessa kriterier användes sedan för att kunna utvärdera resultatet av vår prototyputveckling i fas tre.

För att få fram dessa kriterier valde vi att intervjua yrkesverksamma systemutvecklare, vilka arbetar med flerskiktade distribuerade system för en klient/server-miljö. De personer som vi intervjuade arbetade alla på Xdin och valdes ut på förslag av gruppchefen utefter de kriterier vi givit honom.

- Antal. Vi valde att intervjua fyra personer. Att analysera kvalitiv information tar ganska lång tid och vi bedömde det som att 4-5 personer var en tillräckligt stor mängd för att göra en bra analys.
- Ålder. Vi strävade inte efter att få någon speciell spridning på ålder, eftersom vi inte ansåg det vara relevant i sammanhanget. Respondenterna var alla i trettioårsåldern.

- Bakgrund. Vi ville att personerna skulle ha en bakgrund där de hade arbetat med hantering av samtidighet. Att kunna Java var inte något krav. Hälften av personerna hade mycket goda Java-kunskaper medan de andra två inte hade arbetat med Java. Ingen av de intervjuade hade någon praktisk erfarenhet av EJB sedan tidigare.
- Eftersom vi gjorde intervjuerna under hösten innan vi hade börjat med vår prototyputveckling, så hade vi ingen personlig relation till respondenterna, men vi hade träffat alla utom en vid ett tidigare tillfälle.

3.1.2.2 Intervjuomgång 2

Vår utvärdering av de kriterier som ställs på EJB från de första intervjuerna skulle till största delen komma att grundas på våra egen resultat från prototyputvecklingsfasen. Vi insåg därför tidigt att vårt resultat därför skulle bli ganska begränsat till det enstaka fallet. Därför ansåg vi det viktigt att kunna jämföra våra egna erfarenheter med andras för att kunna dra mer generella slutsater av resultaten.

För att få ett sådant jämförelsematerial valde vi att intervjua yrkesverksamma systemutvecklare, vilka arbetar med EJB. Detta skedde i fas fyra av vår undersökning, efter det att vi utvecklat vår prototyp. De personer som vi intervjuade arbetade på AU-System respektive Cell Network.

Vi ansåg att detta material kan göra resultatet av utvärderingen mer generell, eftersom mångårig erfarenhet då speglar de slutsatser som görs. Genom att på detta sätt jämföra vårt eget resultat med andras erfarenheter hoppades vi kunna ge vårt resultat en högre tillförlitlighet.

Respondenterna i dessa intervjuer valdes utifrån nedanstående kriterier:

- Antal. Vi valde att intervjua två personer. Att antalet blev så lågt berodde främst på svårigheten att hitta respondenter som använt och behärskar EJB.
- Ålder. Vi strävade inte efter att få någon speciell spridning på ålder då vi inte ansåg det var relevant i sammanhanget. De intervjuade var bägge i trettioårsåldern.
- Bakgrund. Vi ville att personerna skulle ha en bakgrund där de hade arbetat med EJB, vilket de hade bägge två.
- Vi hade ingen tidigare personlig relation till respondenterna.

3.1.3 Prototyputveckling

För att kunna få erfarenhet av hur EJB-ramverket kan användas i ett systemutvecklingsprojekt valde vi att utveckla en egen prototyp med ramverket. Genom att ute på ett företag genomföra ett mindre systemutvecklingsprojekt med en erkänd systemutvecklingsmetod skulle vi därigenom få ett material att applicera de framtagna jämförelsekriterierna på.

För att kunna jämföra byggandet av en flerskiktslösning med och utan EJB valde vi att implementera samma system i två olika versioner. Genom att arbeta fram en specifikation i samråd med beställaren ute på företaget fick vi det underlag vi behövde för att analysera, designa och utveckla systemet.

Bägge systemen byggdes för att uppnå kraven i den grundläggande specifikationen. Det ena systemet byggdes med hjälp av Enterprise JavaBeans och en

tillämpningsserver. Det andra systemet byggdes helt med egna klassbibliotek. Som utvecklare blir man då tvungen att hantera samtidighet, transaktionsfunktionalitet och beständighet helt själv. Vi valde emellertid att inte implementera en egen transaktionshantering då vi inte trodde att det var rimligt inom uppsatsens tidsrymd.

Systemet som byggdes med Enterprise JavaBeans gjordes som en fyrskiktlösning medan det andra systemet byggdes som en treskiktslösning. Det fjärde lagret i fyrskiktslösningen består av applikationsservern som ligger mellan klientlogiken (i vårt fall en webbserver och servletmotor) och databaslagret.

Systemutvecklingsmetoden vi använder är en något modifierad form av Larmans UML-baserade metod (Larman, 1998). Vi har beskrivit den utförligt i Bilaga 1. Våra inskränkningar i metoden beskrivs övergripande i kapitlet om prototyputvecklingen, samt utförligt i bilagan.

Syftet med med prototyputvecklingen var att kunna utröna vilka av de framtagna jämförelsekriterierna som uppfylldes av de bägge systemen och därigenom kunna avgöra på vilka sätt EJB är bättre eller sämre för att bygga distribuerade, flerskiktade system.

3.1.4 FAQ-studie

Att studera FAQ-frågor eller nyhetsgrupper kan jämföras med observation (Easterby-Smith et. al, 1991). Genom att observera och analysera de frågor som ställts och de svar som givits i en FAQ inriktad mot EJB, så försökte vi utröna vilka fördelar och nackdelar EJB har för systemutvecklare.

Ett problem med metoden var att vi inte kunde få reda på någonting om de inblandade personernas yrkesbakgrund eller genomsnittliga kompetensnivå. Däremot fick vi reda på vilka problem som uppstått i verkligheten jämfört med de löften som ges från SUN och tillverkare av EJB-servrar.

En fördel med denna typ av observation är att de observerade inte har påverkats av någon yttre influens, som till exempel en intervjuares kanske ledande frågor.

På webbplatsen JGuru[1] läste vi sammanlagt 279 frågor och svar relaterade till EJB.

För att kunna analysera FAQ-frågorna var det tvunget att gruppera in dem i olika kategorier för vårt ändamål. Specifik fokus lades på kategorier som var relaterade till vår problemställning, till exempel de tjänster som utgör hela grunden för ramverket. Vi valde därför att utgå från kategorin *tjänster* med *underkategorierna lagring, transaktioner, säkerhet, samtidighet* och *distribuerade objekt*. Frågor som inte passade in bland dessa kategorier lades under andra, lämpliga kategorier som skapades efter behov.

Hur vi har valt vilken kategori som en viss fråga tillhör kan inte alltid enkelt förstås. Kategoriseringen måste ibland relateras till det givna svaret för att kunna validera valet.

Efter att frågorna var kategoriserade undersökte vi dem kvantitativt för att se inom vilka kategorier det ställdes många frågor. Därefter försökte vi analysera hur pass svåra problem dessa frågor innebar för utvecklaren som ställt frågan. Denna analys

[1] http:// www.jguru.com/

slutade i ett resultat som visar vilka av EJB:s tjänster som verkar fungera bäst att använda sig av.

4 Prototyputvecklingsmetoden exemplifierad

Detta avsnitt syftar till att ge läsaren en större inblick i det stora arbete som utvecklingen av prototypen innebar. Vår förhoppning är att avsnittet skall göra det lättare att ta till sig de resultat som redovisas från denna fas i resultatavsnittet.

4.1 Introduktion

Vi har i vårt arbete med prototypen valt att använda en objektorienterad systemutvecklingmetod som finns beskriven i Larman (1998).

Som notationsteknik har användes UML (Unified Modeling Language) vilket bland annat finns beskrivet i Fowler och Scott (1999). Vi har även kompletterat med vissa notationssätt från Larman (1998).

Systemutecklingmetoden som beskrivs i Larman (1998) är anpassad för större systemutvecklingsprojekt och vi har därför tagit bort en del bitar, som inte har varit relventa för ett mindre projekt som vårt. Larman beskriver i sin metod till exempel hur man skall driva mer omfattande utveckling under flera år med flera parallella projekt.

Under hösten 2000 utförde vi en planerings- och beredningsfas under vilken vi arbetade fram en övergripande specifikation av systemet. Sedan utförde vi tre utvecklingscykler under våren 2001. Varje cykel omfattade analys, design, programkonstruktion och en enklare testfas.

Den första bestod i att utveckla en applikation som bara var anpassad för att klara av en samtidig användare. Detta gjordes för att i första hand färdigställa all funktionalitet.

Den andra cykeln resulterade i en treskiktad distribuerad fleranvändarapplikation som skrevs i Java utan att använda EJB. Den applikationen hanterade inte transaktioner. Under den tredje och sista cykeln utvecklade vi en fyrskiktad fleranvändarapplikation med hjälp av EJB.

Syftet med att jobba i olika cykler är att man under varje cykel skall kunna utveckla en färdig, fungerade produkt som man sedan kan bygga vidare på under nästa cykel. Vi har valt att använda detta tillvägagångsätt med ett delvis annat syfte. Den andra cykeln hade i vårt projekt endast till uppgift att ge oss ett referensalternativ till den tredje cykeln. Vi skulle inte annars ha gjort systemet i den andra cykeln, om vi inte hade haft ett vetenskapligt syfte med det.

4.2 Beskrivning av systemet

Systemet är en företagsintern anslagstavla där de anställda skall kunna skriva och läsa meddelanden som rör saker som är av intresse för dem i deras arbete, men där det även finns möjlighet att lägga in meddelanden av privat karaktär.

En användare kan läsa och lägga till nyheter i systemet. Efter att en anställd lagt in en nyhet kan sedan de som tillhör dennes grupp eller avdelning läsa nyheten. Användarna kan även ta bort de nyheter de själva lagt till om de så önskar.

Nedan följer en beskrivning av de användarscenarier som ingår i cykel 1. Dessa beskriver den funktionalitet som finns i systemet.

Titta på aktuella nyheter

Aktör: nyhetsläsare

Användaren loggar in eller väljer "visa nyheter".

Alla nyheter som är aktuella tidsmässigt och för de grupper (ex. på grupp kan vara ett visst affärsområde) som användaren tillhör visas. Andra gruppers nyheter måste väljas aktivt.

En länk till författarens "personprofilsida" visas för varje nyhet.

Titta på privata nyheter

Aktör: nyhetsläsare

Användaren väljer “visa privata nyheter”.

Alla privata nyheter som är aktuella visas.

Skapa nyhet

Aktör: nyhetsinläggare

Användaren väljer “skapa nyhet”.

Fyller i rubrik, text, kategori, klassificerad/icke klassificerad (klassificerad betyder att den aldrig skall vara synlig för någon som jobbar på en annan avdelning), grupper, www-länk, slutdatum.

Man skall kunna välja att göra en nyhet synlig för en eller flera grupper.

Ta bort egen nyhet

Användaren väljer en av sina egna nyheter och väljer att ta bort den.

Aktör: nyhetsinläggare

Ta bort andras nyheter

Aktör: administratör

Administratören väljer att radera en nyhet ur systemet.

Lägg till/ta bort/ändra person

Aktör: administratör

Administratören lägger in en ny person eller väljer att ta bort en existerande.

Administratören kan också ändra uppgifter för en person.

Lägg till/ta bort/ändra avdelning/grupp

Aktör: administratör

Administratören lägger in en ny avdelning/grupp eller väljer att ta bort en existerande.

Administratören kan också ändra uppgifter för en avdelning/grupp.

4.2.1 Kommentar

Användarscenarierna *Lägg till/ta bort/ändra grupp* samt *Lägg till/ta bort/ändra person* utvecklades endast under cykel 1. Under cykel 2 och 3 kopplade vi oss mot Xdin befintliga persondatabas.

4.3 Systemutvecklingsprocessen

4.3.1 Planerings- och beredningsfasen

Under hösten utförde vi planerings- och beredningsfasen. Den innefattade alltså bland annat en övergripande beskrivning av systemet:

"Systemet är en företagsintern anslagstavla där de anställda skall kunna skriva och läsa meddelanden som rör deras intressen i organisationen."

Dessutom togs systemfunktioner fram som nedanstående:

> **Nyhetsfunktioner R1.1**
>
> Kunna lägga in nyheter.

Dessutom gjorde vi enkla användarmönsterspecifikationer för systemet. Dessa var en enkel beskrivning av olika användarscenarier form av fri text. En användarmönsterspecifikation kan se ut som nedan.

> **Essentiella användarscenarier, ej expanderade**
>
> **Skapa nyhet**
>
> Aktör: nyhetsinläggare
>
> Användaren väljer "skapa nyhet".
>
> Fyller i rubrik, text, kategori (alla eller privat) , klassificerad/icke klassificerad, grupper, www-länk, slutdatum.
>
> Man skall kunna välja att göra en nyhet synlig för en eller flera grupper.

Därefter gjorde vi en konceptuell modell i form av ett klassdiagram. Nästa steg skulle ha varit att fördela användarscenarierna mellan utvecklingcyklerna. Detta steg utfördes inte eftersom vi endast gjorde samma användarscenarier, fast på ett mer avancerat sätt, under cykel två och tre. Man kan därför säga att vi utförde nya systemutvecklingsscenarier.

Sedan gjorde vi en tidsuppskattning av utvecklingcyklerna. Ursprungligen trodde vi att de skulle ta ungefär lika lång tid (2 veckor, dvs 160 mantimmar per cykel) att utföra. Men det visade sig att analysen, samt dokumentationen för denna, under cykel ett och implementationen under cykel tre tog betydligt längre tid än så.

4.3.2 Utvecklingscykler

En utvecklingscykel skall bestå av följande faser:

- Förbättra planerna
- Revidera dokumentationen
- Analys
- Design
- Konstruktion
- Testning

Vi fick avstå från en mer utvecklad testfas för att begränsa tidsåtgången. Eftersom dokumentationen av analysen och framförallt designen är det mest intressanta i vår undersökning har vi valt att redovisa dem nedan.

4.3.3 Analys

Under analysfasen utarbetade vi essentiella användarscenarier, förbättrade de statiska strukturdiagrammen av de konceptuella klasserna, gjorde ”systemsekvensdiagram”, kontrakt för systemoperationer och tillståndsdiagram för konceptuella klasser.

Egentligen skulle man även ha utarbetat en s.k. *glosbok* med begreppsförklaringar men vi valde att inte göra det, eftesom de flesta begreppen i systemet är triviala (*Nyhet* och *Användare* t.ex.) och vi ansåg oss tvungna att begränsa tidsåtgången för att hinna utföra allt arbete inom den givna tidsramen.

Eftersom vi inte lade till några nya användarscenarier under cykel 2 och cykel 3 utfördes ingen analysfas under dessa cykler.

Först gjode vi expanderade essentiella användarmönsterbeskrivningar vilket är mera utvecklade användarmönsterbeskrivningar som är oberoende av hur gränssnittet exakt skall se ut. Nedan följer ett exempel ur dokumentationen för dessa.

Skapa nyhet

Aktör: Nyhetsinläggare

Syfte: Lägga till en nyhet till systemet.

Beskrivning: Användaren skapar en nyhet som skall kunna ses av de specificerarde grupperna.

Typ: expanderat essentiellt.

Kors ref:

Typiskt händelseförlopp:

Aktörens handling	Systemets gensvar
1. Användaren väljer lägga in nyheter	2.Systemet presenterar användaren med de uppgifter som skall fyllas i: • rubrik • text • *eventuell kategori (privat)*

	• *klassificerad/ icke klassificerad* • www-länk • slutdatum Användaren skall kunna välja att göra en nyhet synlig för en eller flera grupper/avdelningar.
3. Användaren väljer/fyller i uppgifterna som systet kräver.	
4. Användaren väljer att valda alternativ skall sparas	5 Systemet sparar valda alternativ
	6 Systemet presentera en bekräftelse

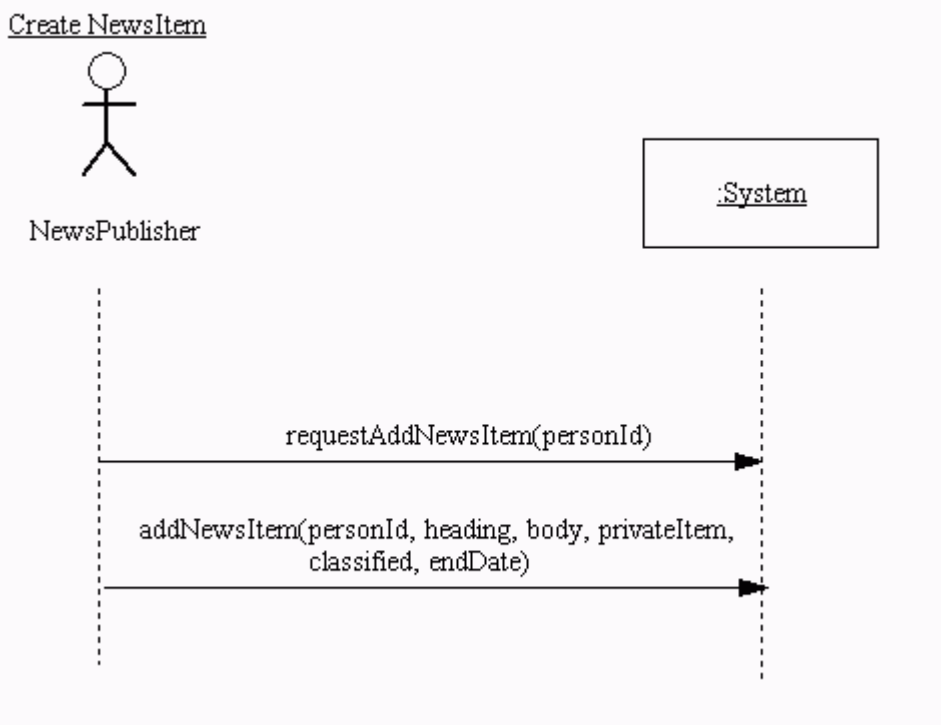
Alternativa händelseförlopp:

Rad 4: Användaren väljer att avbryta inläggandet av nyhet

Rad 6 Om något alternativ inte kan presenteras ett felmedelande.

Sedan förbättrade vi den konceptuella modellen som vi gjorde under planerings- och beredningsfasen.

Därefter gjorde vi systemsekvensdiagram för varje vanlig sekvens av händelser för alla användarscenarier som hanteras i den aktuella utvecklingscykeln. Ett exempel följer nedan.



Därefter gjorde vi systemoperationskontrakt för varje systemoperation i systemsekvensdiagramen som vi gjort tidigare under samma utvecklingscykel.

Create NewsItem

Namn:	requestAddNewsItem(personId:int)
Ansvar:	Visa sida där användaren kan välja att relevanta data.
Typ:	System
Korsreferens:	Systemfunktioner:R1.1
Noter:	-
Undantag:	-
Output:	-
Pre-conditions	-
Post –conditions	-

Namn:	addNewItem(personId:int, heading:String, body:String, categoryId:int, classified:Boolean, endDate:Date, groupId:int[])
Ansvar:	Skapa en newsItem och lagra den
Typ:	System
Korsreferens:	Systemfunktioner:R1.1
Noter:	Undataget skall inte kunna inträffa om man inte hackar html-sidan
Undantag:	Visar ett felmedelande om body eller head innhåller för stor textmassa.
Output:	-
Pre-conditions	-
Post –conditions	En instans av News, nws, skapades. nws.heading = heading nws.body = body nws.privateItem = privateItem nws.endDate = endDate nws.startDate = den tidpunkten metoden anropas. nws.newsId = ett unikt id från en löpande

	sekvens. nwn associeras med de avdelningar vars departmentId har skickats med. nwn associeras med den person som har skapat nyheten.

Ett objekt som alltid regerar på samma sätt på en händelse kan man kalla *tillståndsoberoende*. Eftersom vi enbart arbetade med tillståndsoberoende analysobjekt gjorde vi inga tillståndsdiagram.

4.3.4 Design

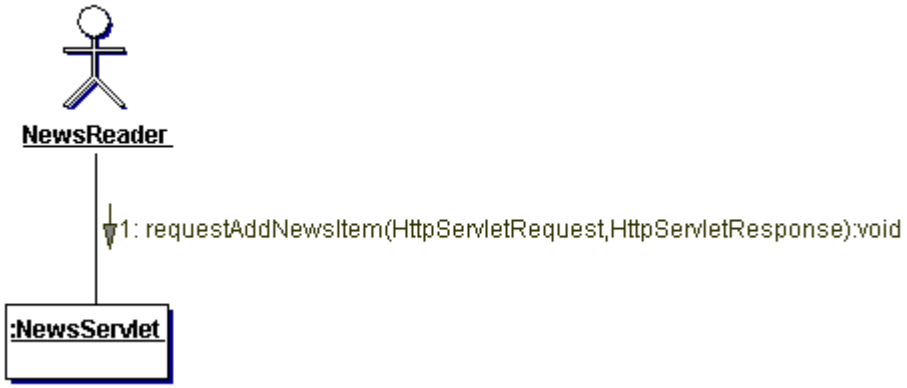
Först gjordes verkliga användarmönsterspecifikationer för alla användarscenarier. I dessa ingick bilder över hur användargränssnitten skulle se ut.

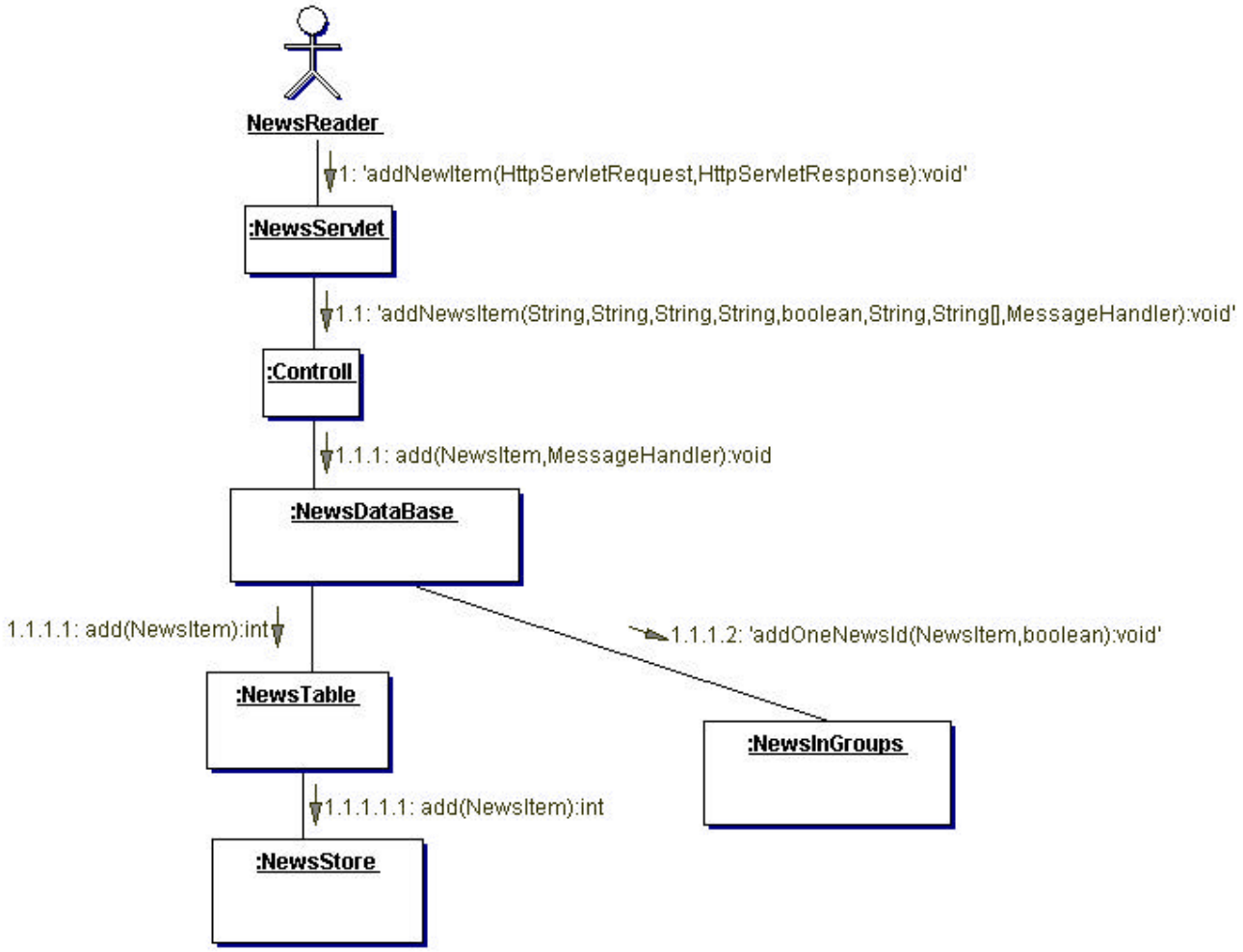
Eftersom systemarkitekturen delvis bestämdes av att vi skulle göra en jämförelse mellan cykel 2 och cykel 3 och delvis av att Xdin önskade en fyrskiktarkitektur på cykel 3 gjorde vi inte den utvärdering av systemarkitekturen som annars skulle gjorts just här utan något tidigare.

För varje systemoperationskontrakt skall det skapas ett collaboration-diagram. Syftet med systemoperationskontrakt är att skapa en bra grund för vilka klasser som skall finna och vilka metoder som dessa skall ha. Till exempel:

Collaboration-diagram, cykel 2

Create NewsItem




NewsReader
1: 'addNewItem(HttpServletRequest,HttpServletResponse):void'
:NewsServlet
1.1: 'addNewsItem(String,String,String,String,boolean,String,String[],MessageHandler):void'
:Controll
1.1.1: add(NewsItem,MessageHandler):void
:NewsDataBase
1.1.1.1: add(NewsItem):int
1.1.1.2: 'addOneNewsId(NewsItem,boolean):void'
:NewsTable
:NewsInGroups
1.1.1.1.1: add(NewsItem):int
:NewsStore

Collaboration-diagram, cykel 3

Create NewsItem

Klassdiagram

Vi gjorde även klassdiagram för att få en översikt av arkitekturen. Klassdiagrammen för alla utvecklingscykler återfinns i slutet av bilaga 4.

5 Resultat

I detta avsnitt presenterar vi uppsatsens resultat. Först presenteras de utvärderingskriterier som togs fram på grundval av intervjuer med systemutvecklare utan tidigare erfarenhet av EJB eller CTM:er.

Därefter visas överskådligt resultaten från prototyputvecklingen och våra erfarenheter av denna. Detta följs av en genomgång av de aspekter av EJB som erfarna EJB-utvecklare ansåg vara relevanta för ramverket.

Sist presenteras resultatet från den kvantitativa studien av FAQ-frågor.

5.1 Utvärderingskriterier

Intervjuobjektens intervjuer sparades till beständigt media i form av inspelning till MiniDisc-format. Därefter fördes intervjuerna över till MP3-format. Dessa filer kan nås av den intresserade via Internet (se bilaga 6).

Syftet med de första intervjuerna var som nämnts tidigare att utröna vilka kriterier erfarna systemutvecklare ställer på ett ramverk för att utveckla distribuerade, flerskiktade system. De intervjuade hade ingen tidigare erfarenhet av EJB.

Efter att ha genomfört intervjuerna gjorde vi en sammanfattning av de viktigaste synpunkterna som intervjuobjektern framförde. Här nedan presenteras denna sammanfattning i form av rubriker baserade på de olika kriterier respondenterna framförde. Resultatet presenteras tillsammans med utvalda citat.

5.1.1 Prestanda

Gemensamt för alla fyra respondenterna är att de ansåg att användandet av ett ramverk inte får resultera i sämre prestanda. Prestanda är oftast det som är absolut viktigast för dem och deras kunder. Särskilt den upplevda prestandan för användarna måste vara hög.

5.1.2 Lättare utveckling

Alla respondenterna tyckte dessutom att det vore mycket intressant att använda ett ramverk där de får hjälp med olika tjänster, som de annars själva måste utveckla funktionalitet för. De gillade tanken på att få mycket ”gratis” och att kunna få fokusera på affärslogiken.

Tjänster som de vill se i ett ramverk inbegriper *transaktions-* och *samtidighetshantering*. De hoppas även på en snabbare utvecklingstid för sina projekt.

”Jag förväntar mig att få mycket gratis. Jag ser EJB inom konceptet J2EE och då förväntar jag mig mycket gratis av detta ramverk. Jag skall inte behöva bry mig om säkerhet, transaktioner eller lastbalansering. Inte det här med synkroniseing heller. […] Kan jag slopa allt detta så fokuserar jag på det affärslogiska problemet.”
– Respondent C

5.1.3 Komplexa datastrukturer

Något som intresserar särskilt två av de respondenterna, som är de mer erfarna Java-utvecklarna (C och D) är möjligheten att kunna representera och hantera komplexa datstrukturer där data finns lagrade i flera tabeller med många kopplingar (*eng.* joins) mellan tabeller. De är lite osäkra på om detta kommer att vara möjligt, men ser det som viktigt för att de skall vara intresserade av ramverket.

”Entity-bönorna som mappas mot en databas. Jag ser inte riktigt fördelarna med dem. Särskilt inte vid komplicerade datastrukturer. Joins för både läsning och uppdatering. Hur gör man det på ett effektivt sätt med hjälp av entitybönor?” – Intervjuobjekt C.

“Det är det jag tycker att alla verktyg försöker generalisera för mycket. En rad i en tabell etc. Så funkar det inte i verkligheten.” - Respondent D.

5.1.4 Komponenter och återanvändning

En av de intervjuade (respondent B), som även är gruppchef, anser att möjligheten att kunna skapa återanvändbara komponenter är ett måste. Problemet med dagens arbetssätt är att man ofta riskerar att, som han säger, uppfinna hjulet igen. Eftersom de är en liten organisation så är det i dagsläget inget stort problem, men B tror att problemet kan växa om organisationen blir större. Då skulle en standardiserad plattform vara till hjälp.

“Vi kan göra en plattform så att de [utvecklarna] kan använda samma saker åt flera kunder. Flytta resurser. Snabbare serva kunden. Folk blir mer samkörda med en bra plattform och det går snabbare att utveckla och vidareutveckla kundens program.” – Respondent B.

Utvecklarna själva tror att ett komponentbaserat ramverk kan tvinga dem att tänka mer på vad varje komponent och modul verkligen skall göra.

“Man kan komma undan mycket spaghettilösningar idag. Man måste fundera mer på bönans nytta när man gör en EJB. Tvinga sig själv i början och göra allt lätt genom att tänka mer skiktat. Göra många, istället för stora klasser.” – Respondent D.

5.2 Prototyputvecklingen

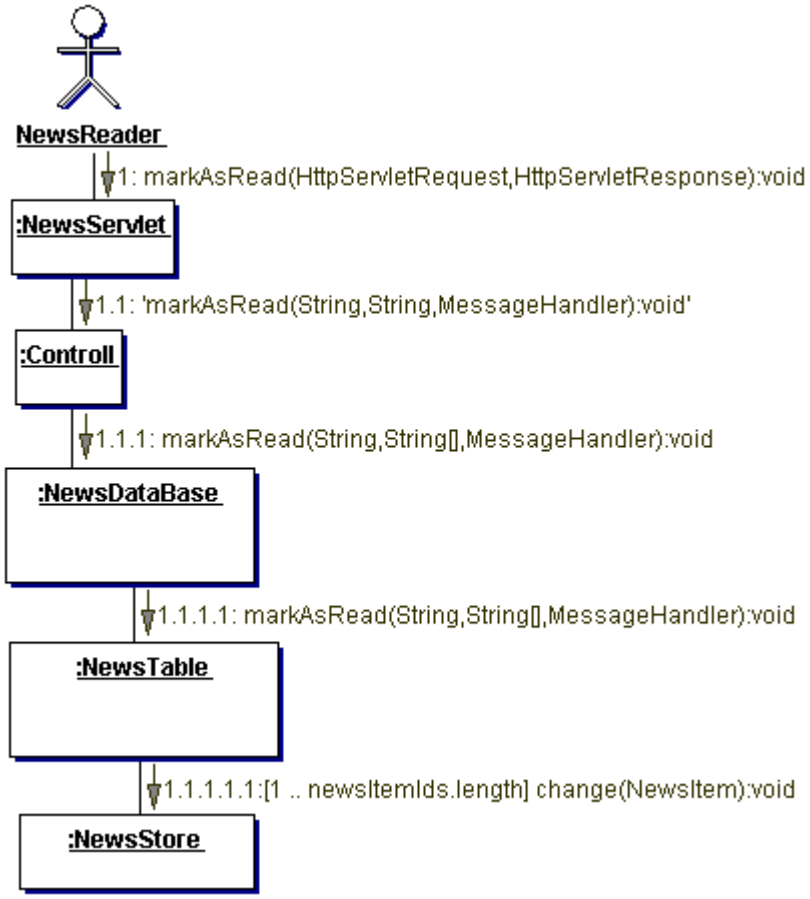
5.2.1 Designdokumenten

Nedan följer en beskrivning av de resultat som prototyputvecklingen resulterade i. De ges i form av collboration-diagram relaterade till EJB:s tjänster. Att vi valt detta sätt att representera resultaten beror på att just EJB:s tjänster är det som är själva grunden i ramverket och det bör då vara dessa som utvärderas. Rubrikerna under varje diagram motsvaras därför av EJB:s tjänster. För en utförligare redovisning av resultaten hänvisar vi till analys- och designdokumenten i bilaga 4.

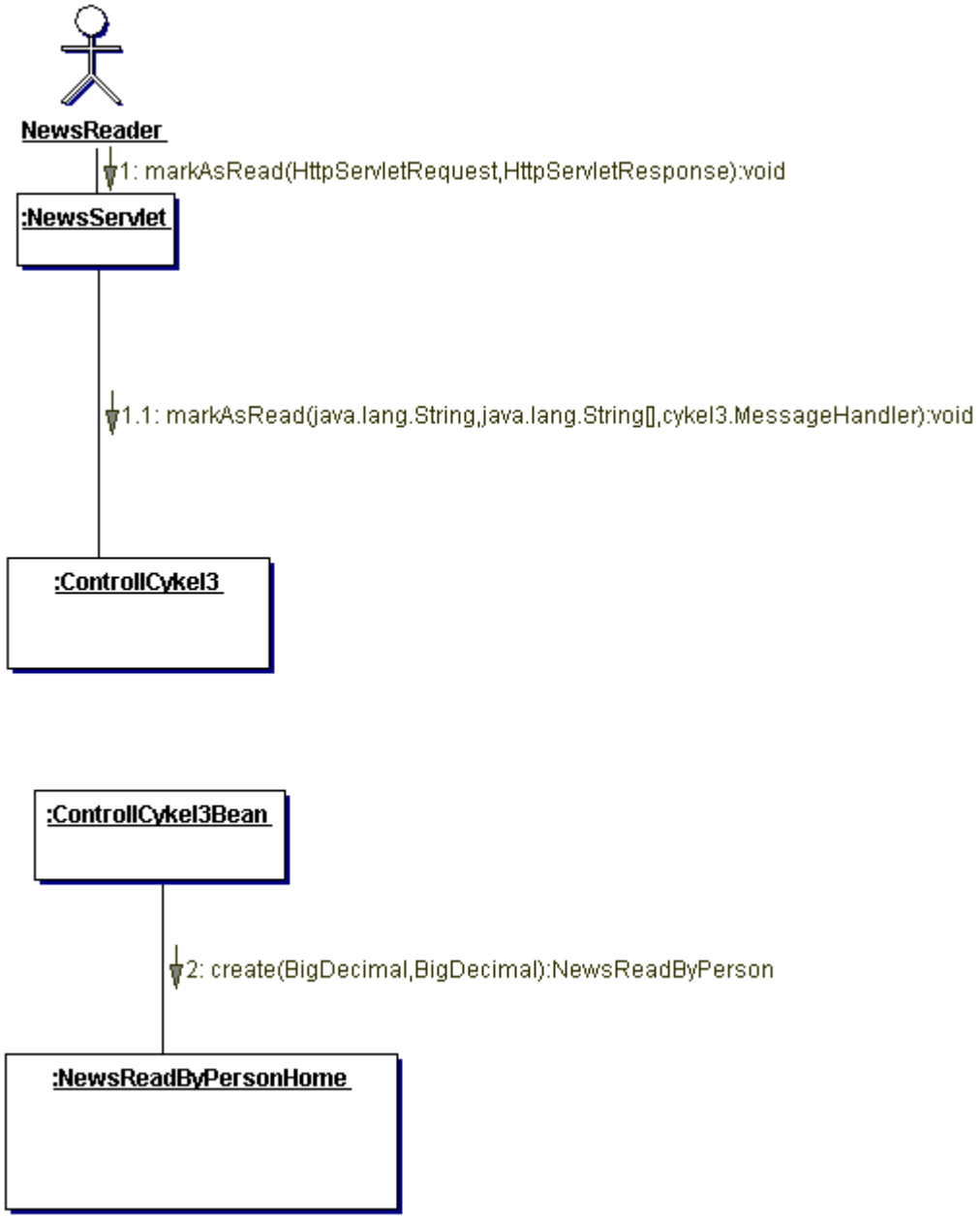
Det som är allmänt för alla collboration-diagrammen beskrivs under det första, *MarkAsRead*.

5.2.1.1 MarkAsRead – en insättning av ett objekt

MarkAsRead är en funktion som används då användaren vill markera att han har läst en nyhet.



Figur 5.1 MarkAsRead, cykel 2



Figur 5.2 MarkAsRead, cykel 3

5.2.1.1.1 Samtidighet

Samtidigheten sköts av containern i EJB-fallet medan det i designen av cykel 2 förutsätts att vissa av metoderna skall synkroniseras (programmeraren hanterar då explicit samtidigheten i sin kod). Detta gäller för alla funktionerna.

5.2.1.1.2 Transaktionshantering

Ovanstående är en funktion som inte kräver någon transaktionhantering, eftersom den bara berör en typ av objekt som lagras (NewsItem), den utför alltså bara en isolerad instruktion.

5.2.1.1.3 Beständighet

Lagringen i prototypens EJB-design sköts via en entitetsböna (*NewsReadByPerson*) och den som designar systemet behöver inte bestämma på vilket sätt detta sker (relationsdatabas, objektdatabas etc.). Inte heller den som implementerar bönan behöver bestämma lagringsättet om han använder CMP-entitybönor, vilket vi senare gjorde. I det fallet bestäms lagringen vid installationen av bönan.

Lagringen i designen av cykel 2 är gjord med intentionen att objektet skall serialiseras (sparas som en binär objektrepresentation) ner till fil.

5.2.1.1.4 Namgivning och tillhandahållande av distribuerade objekt

NewsServlet-klassen skall kunna ligga på en webbserver som är på en annan maskin än resten av klasserna. Detta möjliggörs genom EJB:s tjänster för namngivning och tillhandhållande av distribuerade objekt. Genom namngivningstjänsten hittar NewsServlet sessionsbönan ControllCykel3. Sedan sköts kommunikationen mellan applikationen på webbservern och applikationservern med EJB:s tjänst för distribuerade objekt. Dessa tjänster används på ett liknade sätt i alla funktionerna.

Något motsvarande finns inte för designen av cykel 2. Utan där krävs det att webbserven även har tillgång till alla klasserna lokalt. Om vi skulle designat för liknade (distribuerad) funktionalitet där skulle vi fått använda oss av CORBA eller RMI vilket hade krävt en kraftigt större arbetsinsats jämfört med EJB.

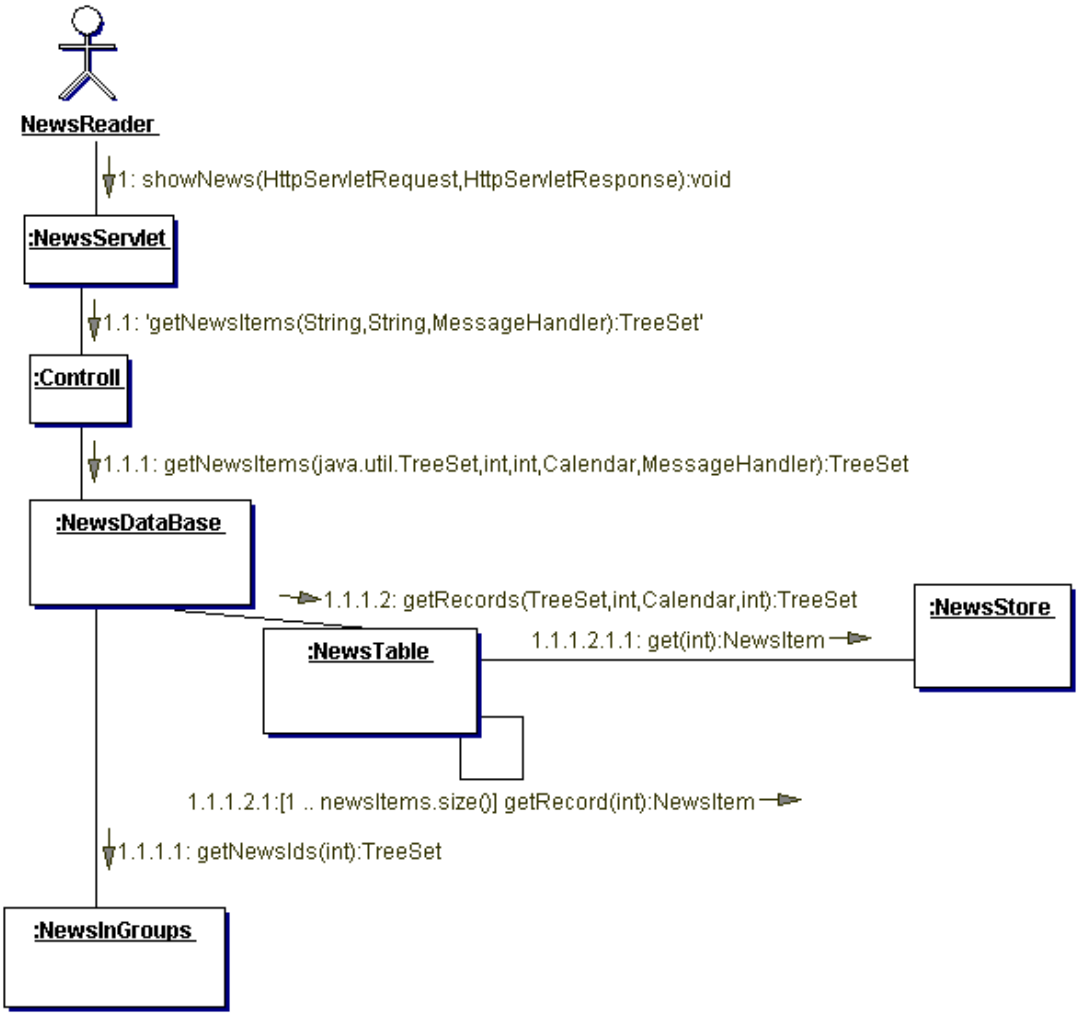
5.2.1.1.5 Säkerhet

Säkerheten i form av inloggningsfunktion och att olika användare har olika behörighet att göra olika saker framgår inte av diagrammet. Det löser vi i EJB genom att användarna får olika roller tilldelade sig när de loggar in i systemet och att de därför har tillgång till olika resurser. EJB:s tjänster för säkerhet används på ett liknade sätt i alla funktionerna.

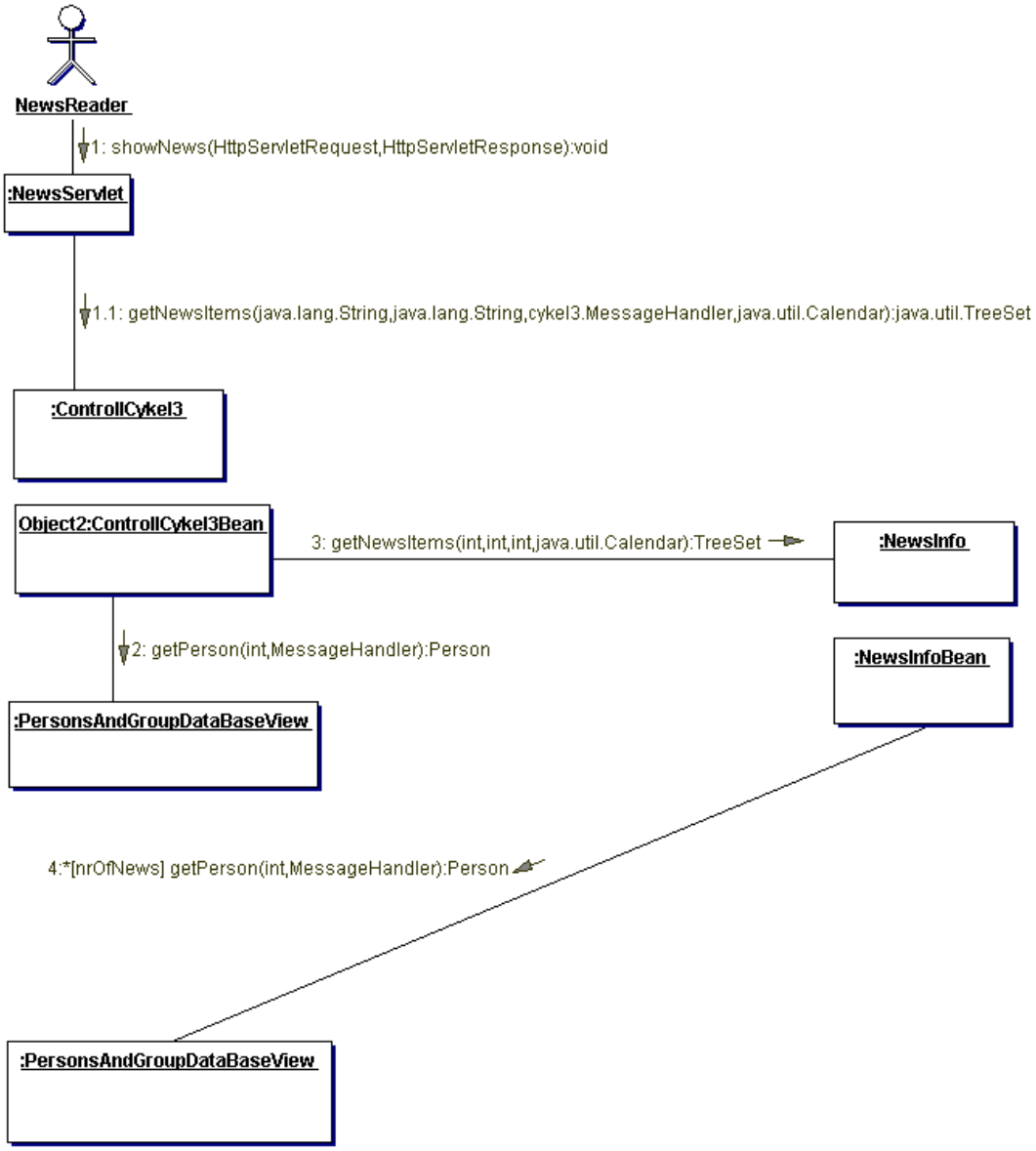
I cykel 2 har vi valt att inte implementera någon autentisering för att begränsa oss tidmässigt.

5.2.1.2 ShowNews – avläsning

ShowNews är en funktion som används när en användare vill läsa nyheter. Den hämtar alla nyher som inte tidigare har markerats som lästa och som är relevanta för användaren med avseende på hans grupptillhörighet.



Figur 5.3 ShowNews, cykel 2



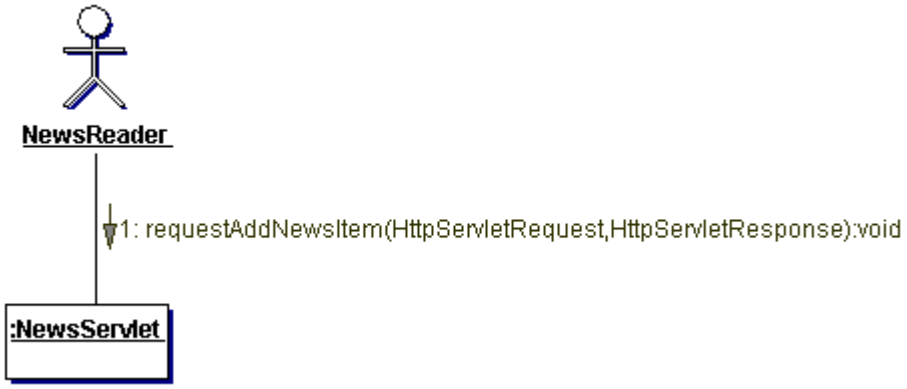
Figur 5.4 ShowNews, cykel 3

5.2.1.2.1 Beständighet

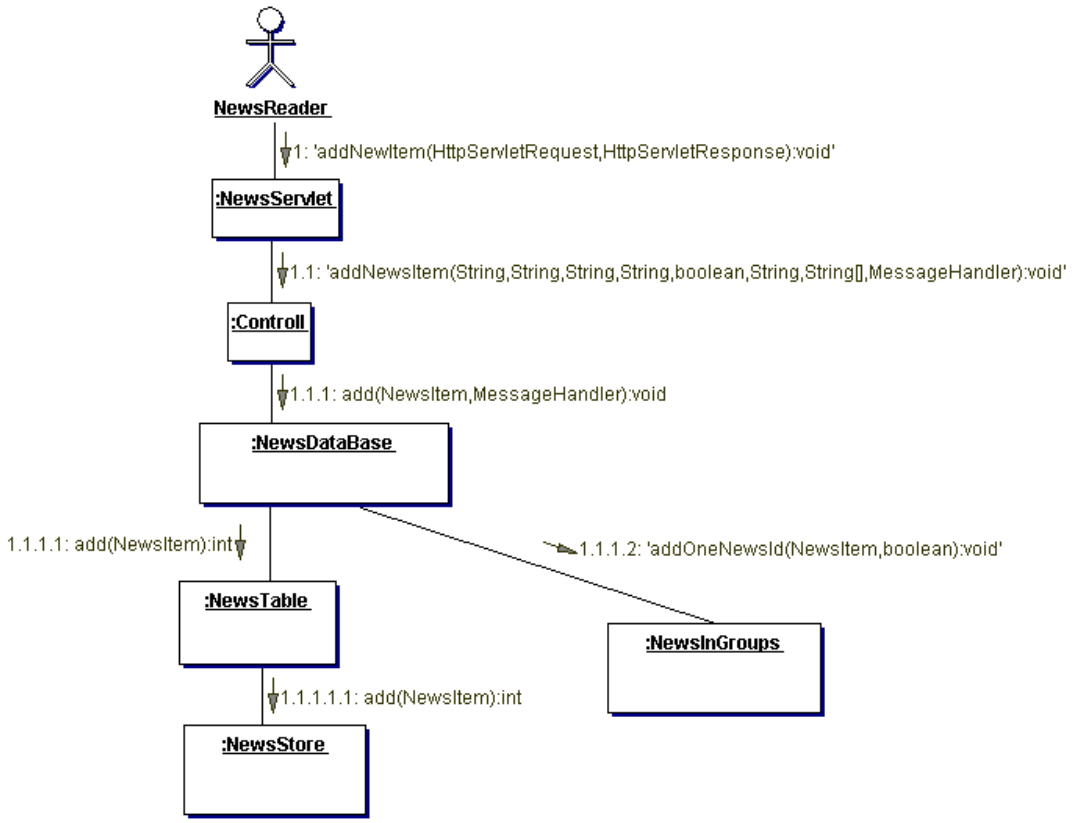
Avläsningen i EJB-designen sköts via en sessionsböna (NewsInfo). Denna sessionsböna anropas av den sessionböna som alla kontakter utifrån sker genom (ControllCykel3). Den som designar systemet behöver inte bestämma på vilket sätt avläsninger sker. Men den som implementerar NewsInfo-bönan måste avgöra vilket lagringsätt som skall användas (relationsdatabaser, objektdatbaser, filer etc.). Lagringen i designen av cykel 2 är gjord med avsikten att flera objektet skall läsas upp från en fil.

5.2.1.3 AddNewsItem – framtida transaktion

När en användare vill skapa en nyhet anropas först funktionen *RequestAddNewsItem*. Då hämtas information som behövs för att skapa relevanta rullmenyer för grupper etc. på klientsidan. Sedan när användaren har skrivit in nyheten och valt revanta data väljer han att spara nyheten och då anropas funktionen *AddNewsItem*.



Figur 5.5 RequestAddNewsItem, cykel 2 och cykel 3



Figur 5.6 AddNewsItem, cykel 2

Figur 5.7 AddNewsItem, cykel 3

5.2.1.3.1 Beständighet

Lagringen i EJB-designen sköts via en entitetsböna (NewItem) och den som designar systemet behöver inte bestämma på vilket sätt det sker (relationsdatabas, objektdatabas etc.). Inte heller den som implementera bönan behöver bestämma lagringsättet om man låter bönan var CMP, vilket vi även senare gjorde i det här fallet. Lagringsättet bestäms då vid driftsättningen av bönan.

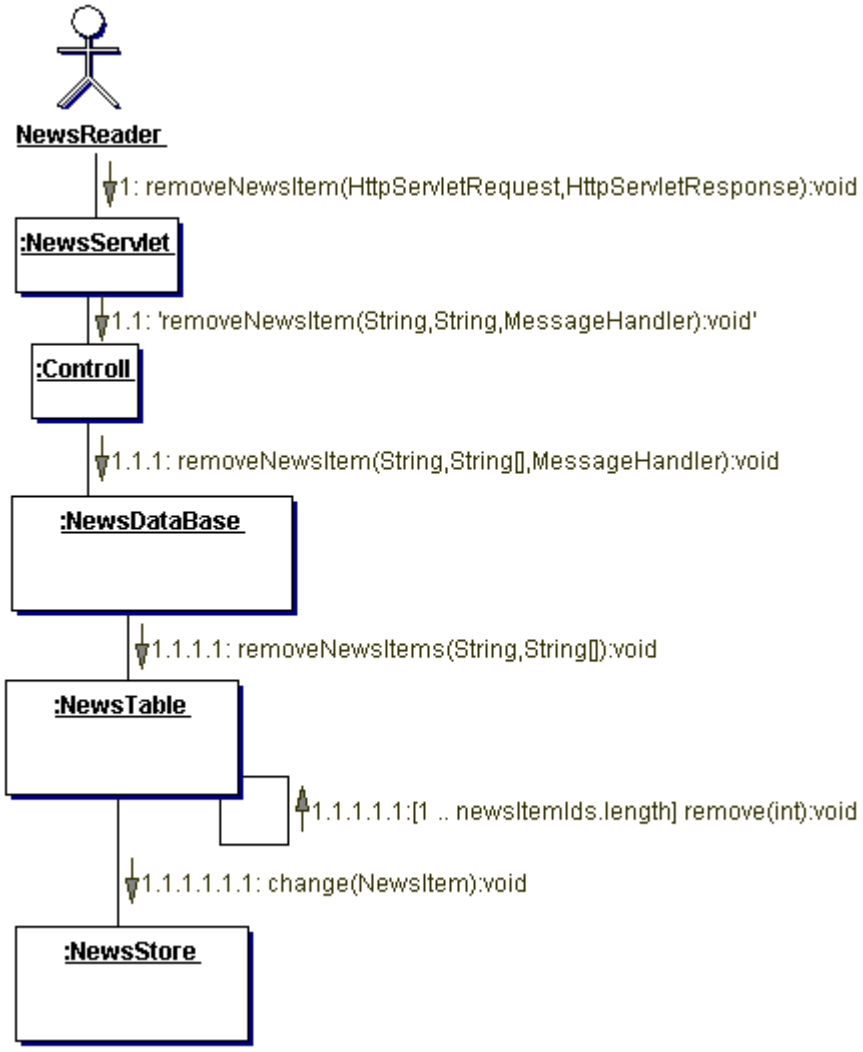
Lagringen i designen av cykel 2 är gjord med intentionen att objektet skall serialiseras (sparas som en binär objektrepresentation) ner till fil.

5.2.1.3.2 Transaktionhantering

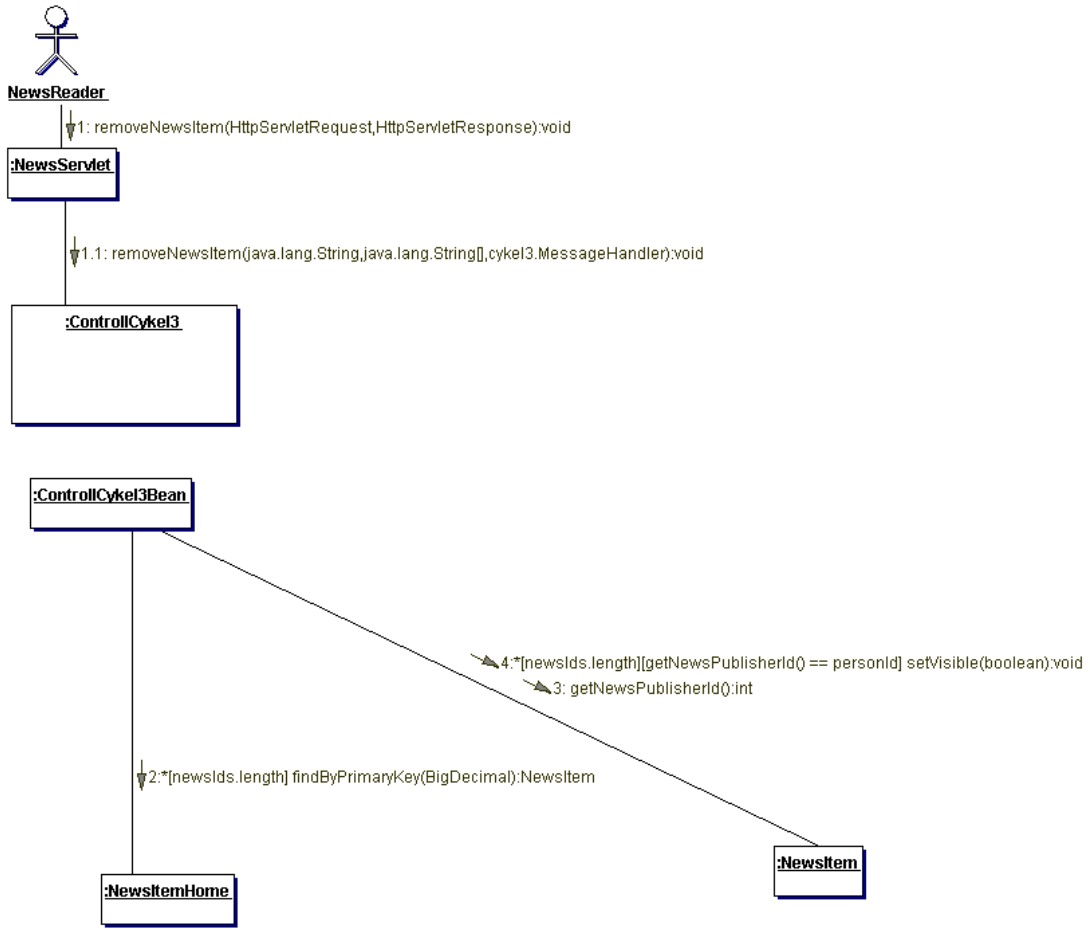
I cykel 2 skulle det behövts en enklare form av transaktionshantering, eftersom funktionen kräver en uppdatering av två olika objekt. I EJB-fallet behövs det ingen transaktionshantering med nuvarande design. Men i en framtida utökning (vilket Xdin har bestämt skall ske) kommer en Nyhet kunna vara knuten till flera olika grupper. Då kommer man att dra nytta av EJB:s tjänster för transaktionshantering.

5.2.1.4 RemoveNewsItem – en ändring av ett objekt

RemoveNewsItem är en funktion som ändrar en nyhet så att den inte längre kan nås av användarna..



Figur 5.8 RemoveNewsItem, cykel 2



Figur 5.9 RemoveNewsItem, cykel 3

5.2.1.4.1 Beständighet

Lagringen i EJB-designen sköts via en entitetsböna (NewItem) och den som designar systemet behöver inte bestämma på vilket sätt det sker (relationsdatabas, objektdatabas etc.). Vid implementationen lät vi entitetsbönan använda CMP. Det var mycket enkelt att ändra ett attribut i objektet jämfört med att vara tvungen att skriva instruktionerna för detta själva vid bönutecklingen.

På den applikationsevern som vi använde var vi dock tvungna att göra det arbetet vid installationen. Lagringen i designen av cykel 2 är gjord med avsikten att objektet skall serialiseras ner till fil.

5.2.2 Utvecklingsverktyg för prototypen

Vi gjorde användarmönsterbeskrivningarna i en vanlig ordbehandlare. De konceptuella klassdiagrammen gjorde vi i *PowerDesigner*[2] från Sybase. Vi gjorde designen i *Together Control Center*[3] från TogetherSoft.

Implementationen av cykel 1 och cykel 2 skedde i Borland *JBuilder 4 Enterprise Edition*[4]. Detta är en så kallad IDE (Integrated Development Environment) för programmering, kompilering och avlusning.

---

[2] http://www.sybase.com/products/internetappdevttools/powerdesigner/

[3] http://www.togethersoft.com/us/products/index.html

Samma verktyg användes även för implementationen av klasserna och interagerandet mellan dessa i cykel 3. Men driftsättningen till applikationsservern *Allaire JRun*[5] gjorde vi delvis via verktyg som fanns i JRun, men mestadels genom att skriva dritftsättningsfiler i en texteditor. Driftsättningen via Allaires enkla webbgränssnitt var ofta väldigt instabil, men när vi gjorde driftsättningen manuellt via en kommandotolk och väl hade listat ut hur man skulle göra fungerade det tillfredställande.

Vi underskattade kraftigt den tid de skulle ta att göra driftsättningen. Kanske för att det i den allmänna EJB-litteraturen beskrivs som en mycket enkel procedur, vilket det antagligen var tänkt att vara.

Ingen av oss hade använt något av de nämnda verktygen tidigare. Detta ledde givetvis till att en viss tid gick åt för att lära sig verktygen.

5.3 Andras erfarenheter

Här beskriver vi resultaten från intervjuerna med två erfarna EJB-utvecklare. Respondent E:s intervju finns även det i digitalt format. Länkar till detta material återfinns i bilaga 5 där även intervjufrågorna finns redovisade. Vi fick dock ej möjlighet att återge respondent F:s intervju i digitalt format.

Respondenternas åsikter återges under rubriker som knyter an till de som användes vid den första intervjuomgången.

5.3.1 Respondent E

Denna respondent är en erfaren mjukvaruutvecklare med fem års erfarenhet av Java-programmering. Han har jobbat med EJB ungefär så länge som det har funnits.

Respondenten utvecklar för tillfället så kallade portal-system. I detta ingår bland annat en spelplattform för bredbandsanvändare som de kallar för games-on-demand. Detta system är utvecklat med hjälp av EJB i ett fyrskiktat system. De valde att enbart använda sig av tillståndslösa sessionsbönor i sin installation. Delvis på grund av att de anser att prestanda blir bättre, men också för att systemet inte är så transaktionsorienterat.

5.3.1.1 Prestanda

Prestandan på deras EJB-utvecklade system ansåg respondenten vara mycket tillfredsställande. Eftersom de använder sig av EJB och applikationsservrar är det bara att öka på antalet datorer i det kluster som systemet är byggt med för att öka prestandan. Han tyckte att det var en stor styrka för EJB att systemen körs i en kontrollerad omgivning i ett standardiserat ramverk.

5.3.1.2 Utveckling och återanvändning

Respondenten ansåg sig lite besviken på EJB, då han inte tyckte sig ha fått ta del av de utlovade produktivitetsökningarna eller återanvändning av komponenter. Han hänvisade också till en undersökning som hävdar att det inte är någon idé att ens försöka återanvända kod rent generellt.

---

[4] http://www.borland.com/jbuilder/

[5] http://www.allaire.com/Products/JRun/

Han medgav dock att de i sina projekt inte använt EJB till det som det är verkligt bra på: transaktions- och resurshantering.

”Fast i det projektet som jag är nu har det [EJB] inte använts till det som det är bra på. I min värld är EJB bra på att hushålla med resurer och underlätta för systemutvecklare när det gäller transationshantering. Och även till viss del om man kör EJB-bönor med CMP.” – Respondent E.

5.3.1.3 Komplexa datastrukturer

Eftersom respondenten inte använt sig av entitetsbönor nämnvärt så kunde han inte uttala sig om EJB:s förmåga att hantera komplexa datastrukturer.

5.3.1.4 Övrigt

Det som respondenten främst framhöll i sina svar var att han tycker om att EJB använder sig av en applikationsserver. Han tyckte att fördelarna i form av en säker, konfigurerbar, stabil och skalbar miljö som EJB ger är den största orsaken till att de använt sig av ramverket istället för att utveckla systemen helt med egen kod.

5.3.2 Respondent F

Den andra respondenten är systemutvecklare sedan tio år tillbaka. Han har arbetat med javaprogrammering i tre år och med EJB sedan i september 2000.

Anledningen till att hans företag använder sig av en viss teknik eller ramverk beror till största delen på krav från deras kunder, då dessa ofta anger vilken teknik som skall användas i sina systemspecifikationer.

5.3.2.1 Prestanda

Respondenten tyckte att EJB-system uppnår minst lika bra prestanda som distribuerade system utvecklade med andra teknologier. Deras kunder har i vilket fall varit nöjda med den prestanda deras system har levererat.

Deras kunder har enligt respondenten inte heller så stora krav på prestanda i början av ett projekt. Det är istället viktigt med en snabb utvecklingstid för att få ut produkten eller tjänsten på marknaden. Prestandakrav skärps däremot ofta vid vidareutveckling av systemen.

5.3.2.2 Utveckling och återanvändning

Respondenten tyckte att EJB lämpar sig bäst för projekt där systemet har komplexa verksamhetsregler och där man behöver använda sig av en transaktionskontext. Det är också viktigt att det är ett system som kunden vill utveckla under en längre tid och kanske bygga hela sin verksamhet på.

Mjukvaruutvecklingsprocessen anser han inte ser särskilt annorlunda ut för dem när de utvecklar med hjälp av EJB. Som systemutvecklare anser han dock att man uppnår en hel del tidsvinster på grund av EJB:s tjänster i form av transaktionshantering, den välutvecklade komponentmodellen och andra, ibland applikationsserverspecifika, tjänster som exempelvis databaspooler. Dessa tjänster anser också respondenten vara EJB:s största styrka.

Respondenten upplevde EJB som ett lättanvänt ramverk för utveckling av flerskiktade system inom ramen för J2EE. Han fokuserade också på de *design patterns* som utvecklaren kan använda sig av ihop med EJB.

Någon ökning av återanvändandet av kod har respondenten inte kunnat uppnå mellan sina projekt. Han ansåg att det kräver en för stor arbetsinsats för att göra tillräckligt generella komponenter och att detta då blir en oönskad kostnad för den första kund som får betala för detta i form av ökad utvecklingstid.

5.3.2.3 Komplexa datastrukturer

Att använda sig av komplexa datastrukturer med EJB tyckte respondenten var lite svårt i nuläget. Särskilt svårt ansåg han det vara att representera associationer mellan olika tabeller. Därför har de inte använt sig av entitetsbönor särskilt mycket utan använder istället sessionsbönor med direkt åtkomst till datalagret.

5.3.2.4 Övrigt

Respondenten är positiv till EJB i allmänhet och tycker att framtiden för ramverket ser bra ut. De nackdelar han har märkt med EJB är att applikationsservrarna är dyra och att man lätt kan låsa sig till en viss servertillverkare om man använder sig av deras proprietära API:er.

5.3.3 Kommentar

Vi upplever det som en svaghet att de två respondenterna inte har utvecklat system som har varit transaktionstunga och de har inte använt sig av entitetsbönor i någon större utsträckning. Deras system har heller inte, enligt respondenterna, modellerat en verksamhetsmässigt avancerad omgivning. Möjligtvis skulle man ha fått en mer övergripande bild av EJB:s fördelar och nackdelar om man även intervjuat individer med bakgrund inom exempelvis bankvärlden. Tyvärr fick vi inte denna möjlighet.

5.4 FAQ-analysen

Syftet med FAQ-studien var att hitta kvantitativt material som visar på vilka svårigheter som är vanliga vid utveckling med EJB.

De observerade frågorna täckte ett stort område. Allt ifrån elementära designfrågor till teoretiska frågor rörande detaljer i olika versioner av EJB-specifikationen. Vi beslutade oss för att dela in frågorna i följande kategorier:

- EJB:s tjänster
- Frågor om EJB-specifikationen, samt grundläggande frågor om EJB
- EJB och andra Java-teknologier
- Frågor som berör produkter från olika tillverkare (till exempel applikationsservrar)
- Övrigt

Eftersom vi anser att det är EJB:s tjänster och arkitekturfrågor som är avgörande för att utvärdera EJB utifrån vår frågeställning valde vi att främst analysera frågor som rörde dessa båda områden.

5.4.1 EJB:s tjänster

Den överlägset största kategorin (cirka 40%) innehöll frågor om de olika tjänster som EJB-ramverket tillhandahåller. Som vi visat i teroriavsnittet tillhandahåller EJB tjänster inom huvudsakligen fem områden: samtidighet, transaktioner, beständighet,

distribuerade objekt, namngivning och säkerhet. Över hälften av frågorna (20% av samtliga frågor) om dessa tjänser berörde ämnen relaterade till beständighet.

Intressant att notera är att nästan inga frågor alls handlade om tillståndslösa sessionsbönor och bara ett fåtal berörde sessionsbönor med tillstånd. Merparten av frågorna berörde alltså entitetsbönor och av dessa frågor finns det fler frågor som specifikt handlade om CMP-entitetsbönor än sådana som berörde BMP-entitetsbönor.

En fjärdedel av frågorna inom kategorin tjänster berörde transaktioner (10% av samtliga frågor). Av dessa är de flesta antingen ganska självklara för den som har förstått grunderna i EJBs transaktionshantering eller så berörde de olika specialfall. Av dessa handlar ganska många om när man inte använder tjänsten fullt ut utan vill kontrolla transaktionerna genom *BMT* (bean managed transactions) eller liknande.

Den resterande fjärdedelen av frågorna om tjänster fördelades på säkerhet, samtidighet och distribuerade system. Av dessa handlade bara några få om samtidighet. Även vissa av frågorna inom kategorin EJB och andra Java-teknologier (nedan) berörde distribuerade system.

5.4.2 Frågor om EJB-specifikationen, samt grundläggande frågor om EJB

Den näst största kategorin handlade om EJB-specifikationen och grundläggande frågor om EJB (17%). Av dessa var de flesta frågorna om grundläggande EJB. Många av dem berörde också den grundläggande EJB-arkitekturen. Andra (ungefär en fjärdedel) var mycket simpla frågor som ofta började: “Vad är...?”. Till exempel “vad är en enterprise bean?”.

5.4.3 EJB och andra Java-teknologier

Var tionde fråga berörde EJB och andra Java-teknologier. Med andra Java-teknologier menar vi till exempel JSP (Java ServerPages), Java Servlets, JMS (Java Messaging Service), JNDI (Java Naming and Directory Interface), JDO (Java Data Objects) och JavaMail. En del av dessa var arkitekturrelaterade. De flesta hade anknytning till distribuerade system.

5.4.4 Frågor som berör produkter från olika tillverkare

En fråga av tio berörde produkter från olika tillverkare. Exempel på produkter som det frågades om var applikationsservrar som exempelvis BEA WebLogic, IBM VisualAge, Allaire JRUN och Inprise VisiBroker. Andra produkter som diskuterades i anslutning till EJB var webbservrar som Apache och servletmotorer som Tomcat.

5.4.5 Övriga

Bland de övriga frågorna fanns det bland annat frågor om EJB jämfört med Microsofts COM/DCOM-arkitektur, MTS (Microsoft Transaction Server), Microsoft DNA (Distributed interNet Architecture), samt RMI och CORBA.

5.4.6 Kommentar till kategoriseringen

Vi hade redan från början valt att ha kategorin *tjänster* med underkategorierna: *lagring, transaktioner, säkerhet, samtidighet* och *distribuerade system* eftersom EJB tillhandhåller tjänster för just detta. Vår utgångspunkt var även att vi skulle göra de andra kategorierna så relevanta för frågeställningen som möjligt.

Kategorin “frågor om EJB-specifikationen” var en tydlig kategori. Kategorin “grundläggande frågor om EJB” valdes utifrån att det fanns en mängd frågor som inte hade behövt ställas på en FAQ om de som frågade hade läst en grundläggande introduktion till EJB innan de frågade.

Kategorin “applikationsservrar och andra om produkter från olika tillverkare” var en tydlig kategori, eftersom en av EJB:s så kallade utvecklingsroller är att installera en EJB-applikation på en applikationserver (DeMichiel et al., 2001, s. 34).

Övriga frågor grupperades tillsammans med de som var liknande. De enda av dessa grupper som vi ansåg var tillräckligt stor för att skapa en egen kategori för var “EJB och andra Java-teknologier”.

5.4.7 Kommentar till resultatet

Att det bara vara var tionde fråga som handlade om applikationsservrar och liknande kan nog inte anses vara representativt. FAQ:n var öppen för alla typer av frågor. Därför kan det antas att de flesta som undrar om något som berör en specifik applikationserver inte vänder sig hit utan till en FAQ (eller forum) som är knuten till tillverkaren av den specifika applikationserver.

Då vi använde JRun som applikationserver försökte vi att använda ett forum knutet till denna produkt. Där handlade många fler frågor om just applikationservern. Tyvärr ansåg vi inte forumet vara av lika hög kvalitet när det gäller moderering och svaren på frågorna, varför vi inte valde att inkludera det i undersökningen.

Resultatets validitet kan eventuellt komma att ifrågasättas då det är svårt att veta något om den population som observerats och det blir då svårt att till fullo motivera att resultaten är generella för en större population av systemutvecklare.

6 Diskussion

För att utröna på vilka sätt det är bättre att använda Enterprise JavaBeans för att bygga ett flerskiktat system än genom att utveckla samma funktionalitet i Java utan att använda något ramverk kommer vi i vår diskussion att knyta samman våra forskningsresultat med tidigare forskning och teori.

Det framgår av litteraturen, de intervjuer vi har utfört, samt våra egna erfarenheter av utveckling att de utvecklare som skapar flerskiktade system har många uppgifter att lösa och att all hjälp de kan få med att lösa dessa är av godo. Vi fokuserar här på de olika tjänster som EJB-ramverket erbjuder. Vi anser dessa tjänster vara grunden i ett CTM-ramverk som EJB och att det är med hjälp av dessa tjänster som de stora vinsterna kan göras i systemutvecklingsprocessen. Det är på grundval av hur väl dessa tjänster fungerar i EJB, som ramverket bör utvärderas.

Diskussionen behandlar först de olika tjänsterna i EJB och hur väl dessa fungerar enligt våra resultat. Därefter diskuterar vi hur systemutvecklingsprocessen påverkas av användandet av EJB. Efter detta uttalar vi oss om några aspekter på framtiden för EJB. Vi skriver också lite om hur det var att använda de olika verktygen under utvecklingen av prototypen. Vi avslutar diskussionen med att presentera uppsatsens slutsats.

Distribuerade objekt

Det förekom relativt få frågor om EJB:s tjänster för distribution i FAQ-resultatet och det fanns inget negtivt om dem i intevjuerna. Resultaten från prototyputvecklingen visade att EJB:s hantering av distribuerade objekt fungerar utan problem.

Detta tyder på att distribuerade objekt en lättanvänd tjänst i EJB, som underlättar för utvecklaren. Namngivning och tjänsterna för att hantera distribuerade objekt visade sig i prototyputvecklingen vara en förutsättning för att kunna utveckla just en *distribuerad*, lättanvänd fyrskiktsarkitektur (där webbserver och applikationserver befinner sig på olika platser). När klienter och applikationsserver är på samma maskin är det inte lika relevant, men då kan man ju knappas tala om ett system som distribuerat.

Namngivning

Inget i vårt resultat tyder på att namngivning skulle vara en svåranvänd tjänst. Den är en förutsättning för att ett distribuerat objektsystem skall kunna fungera och är i den form den förekommer i EJB mycket lättanvänd.

Samtidighet

I FAQ-analysen fanns det få frågor om samtidighet. Det är inte heller en fråga som tas upp av respondenterna i intervjuomgång ett i någon större omfattning. Av prototyputvecklingen framgår det att samtidigheten fungerar mycket bra i EJB, eftersom man inte behöver hantera den själv som utvecklare. Respondenterna i intervjuomgång två angav inte heller att de hade haft några problem med att använda sig av denna tjänst. Av detta kan vi dra slutsatsen att EJB:s tjänst för samtidighet är lätt att implementera och verkar fungera mycket bra.

Säkerhet

Det fanns få frågor om EJB:s tjänst för säkerhet i FAQ-analysen. Den har dessutom inte använts alls av respondenterna i intervjuomgång två. Vi använde dem under

prototyputvecklingen, men det krävde relativt mycket arbete vid driftsättningfasen på applikationsservern och inte mycket av detta arbete kan sägas vara generellt och portabelt mellan olika applikationsservrar.

Utifrån detta kan man anta EJB:s tjänster för säkerhet inte tillför så mycket till systemutvecklingsprocessen, eftersom den hjälper till med en så pass begränsad del av det totala säkerhetsbehovet. Säkerhet var däremot inte den mest efterfrågade funktionaliteten i ett ramverk för att utveckla distribuerade, flerskiktade system, vilket återspeglas i de intervjuer vi utförde i intervjuomgång ett. De flesta utvecklare verkar nöjda med att hantera säkerhetsfrågor själva för varje unik installation.

Beständighet

Det förekom många frågor om EJB:s tjänster för beständighet i FAQ-analysen. Mycket få av dessa handlade om sessionsbönor. Respondenterna hade nästan bara använt sessionsbönor i faktiska systemutvecklingsprojekt och skött lagringen helt själva utan entitetsbönor.

Den stora skillnaden i att hantera beständighet uppnår man i EJB genom att användanda sig av CMP-entitetsbönor. Dessa bönor var mycket enkla att skapa, men krävde däremot mycket mer arbete vid driftsättningen till applikationservern. Eftersom vi i vår prototyputveckling inte hanterade särskilt komplexa datastrukturer, så kan vi inte uttala oss om hur väl EJB hanterar dessa.

Utifrån våra resultat från prototyputvecklingen anser vi att CMP-bönor är mest lämpliga vid bönutveckling av generella bönor som man vill kunna installera på flera företag med olika typer av lagringsinfrastruktur (olika tabellstrukturer, databaser etc.).

Utifrån detta och resultaten av FAQ-analysen och intervjuerna insåg vi att de flesta verkar uppfatta entitetsbönor som svårhanterliga i dagsläget, eftersom det inte är tillräckligt enkelt att flytta entitetsbönor från en installation till en annan. Även svårigheten att representera komplexa datastrukturer i vissa applikationsservrar har lett till en liten användning av CMP-bönor.

Transaktioner

Det förekom många frågor om EJB:s tjänster för transaktioner i FAQ-analysen. Av dessa handlar ganska många om specialfall där man inte använder tjänsten fullt ut utan vill kontrolla transaktionerna genom *BMT* (bean managed transactions) eller liknande. Respondenterna har heller inte använt EJB:s tjänster för transaktioner i någon större omfattning.

Vid prototyputvecklingen har vi nyttjat oss av tjänsten men det har egentligen inte tillfört något för det aktuella systemet eftersom alla metoder i prototypen är så små att de inte behöver använda sig av transaktioner.

Slutsatsen vi drar av detta är att EJB:s tjänster för transaktioner en lättanvänd tjänst via CMT (container managed transactions). För att den skall vara av värde krävs dock en mer komplex verksamhetslogik än i den prototyp vi utvecklat och i de system som våra respondenter har implementerat.

6.1 Systemutvecklingsprocessen med EJB

Vår studie visar att utvecklare lägger generellt sett ganska stor tid på att hantera problem som kan uppstå på grund av samtidighetskrockar eller brist på transaktionshantering. De utvecklare som vi har studerat har kunnat prestera en

mycket mer tillförlitlig produkt i slutändan genom att använda EJB. Dessutom går utvecklingen snabbare eftersom transaktionshanteringen sköts automatiskt av EJB-servern. Ingen kod behöver skrivas för detta.

Utveckling av distribuerade system visade sig också vara enkelt med hjälp av EJB. Komponentmodellen är relativt lättförståelig för en erfaren systemutvecklare.

Vid utvecklandet av flerskiktade system är EJB också en hjälp. Affärslogik är enkel att implementera som sessionsbönor och beständighet sköts med hjälp av entitetsbönor. EJB passar dessutom in i flerskiktstänkandet på ett mycket naturligt sätt.

EJB är däremot en krävande teknik på andra sätt. Att sätta sig in i handhavandet av en applikationsserver tar tid och kvaliteten på dessa servrar varierar. Den största delen av utvecklingstiden har för vår del inte varit att skriva själva bönorna utan att få dem att fungera i en viss applikationsserver. Det är därför viktigt att noga studera vilken applikationsserver man bör använda sig av innan man påbörjar ett systemutvecklingsprojekt med EJB.

En applikationsserver är också en dyr affär. Dessa kostar mellan några tiotusental och hundratusentals kronor, detta är bara för mjukvaran. Fördelen med applikationsservrarna är däremot att de ofta är mycket skalbara. Att koppla in mer datakraft i form av fler maskiner är en relativt enkel procedur. Egenutveckling av sådana klustrade system är en mycket svår konst, som tar mycket utvecklingstid i anspråk. För det lilla företaget finns också gratis EJB-servrar utvecklade i enlighet med Open Source-konceptet[6] till exempel JBoss[7].

Vi har i vårt arbete sett att de system där EJB lämpar sig bäst är där systemet uppvisar en komplex affärslogik med många steg i varje transaktion. Exempel på sådana verksamheter är till exempel system inom bank- och e-handelsvärlden. Det är då fördelarna med en effektiv transaktionshantering är som störst. En del system är inte heller så känsliga för samtidighetsproblem. Det är inte alltid som delad data är känslig.

EJB är ett försök att göra för affärsvärlden vad Java gjorde för datorvärlden. Att kunna skriva komponenter som är helt flyttbara mellan olika miljöer. Detta har dock inte helt uppnåtts ännu. De flesta EJB:er blir inte helt generella eller applikationsserveroberoende på grund av de stora skillnader som de facto existerar mellan de olika servrarna i dag. Vår studie visar att detta har lett till en del besvikelse, men det är väl sällan som en utopi uppnås. ”Write once, run everywhere” var det motto varmed Java lanserades som programmeringsspråk. Även det var en sanning med modifikation. Men EJB är ändå en lovande teknik för att uveckla avancerade system och den har ännu inte hunnit mogna än.

6.2 EJB:s framtid

In om en snar framtid kommer EJB-servrar baserade på den nya EJB-specifikationen (DeMichiel et al., 2001) att komma ut på marknaden. Då finns det stora möjligheter att enterprise-bönornas flyttbarhet mellan olika miljöer kommer att öka.

---

[6] http://www.opensource.org/docs/definition.html Definition på Open Source.

[7] http://www.jboss.org/

Den nya specifikationen introducerar nämligen vad som kallas för *containterhanterade relationer* för entitetsbönornas datafält. Denna teknik skall göra det lättare att representera komplexa datastrukturer med CMP-bönor. (DeMichiel et. al., 2001, s. 217). Detta kombineras med ett nytt standardiserat frågespråk, som kommer att kallas för EJB QL (*eng.* EJB Query Language). Detta språk skall göra det enklare att göra komplicerade utsökningar av entitetsbönor.

Förhoppningsvis leder detta till att komponenter kan bli mer portabla mellan olika applikationsservrar, vilket ju är en av de stora anledningarna till att EJB skapades. Användningen av containterhanterade relationer kommer dock antagligen att kräva ganska avancerade verktyg. Baserat på vår erfarenhet av de existerande verktygen för driftsättning, så tror vi att det kommer att dröja ett bra tag innan dessa verktyg blir tillräckligt mogna för att kunna användas på ett produktivt sätt i verkliga systemutvecklingsprojekt.

6.3 Utvärdering av de olika verktygen

PowerDesigner från Sybase var inriktat mot design av relationsdatabaser. Därför använde vi det inte till några fler moment än till det konceptuella klassdiagrammen. För detta ändamål fungerade verktyget bra.

Together Control Center, som vi använde till vår systemdesign var ett bra verktyg för att göra klassdiagram och collaboration-diagram, men har även mycket fler möjligheter för att stödja en helt UML-baserad analys och design.

Det största problemet med Together var att det krävde väldigt mycket arbetsminne. Detta gjorde att den av maskinerna som vi använde som enbart hade en begränsad mängd arbetsminnde arbetade mycket långsamt. Annars fungerade verktyget bra och man kunde göra diagrammen på flera alternativa sätt.

Något som vi saknade, men som inte var avgörande, var en bättre integration mellan Together och vår IDE JBuilder. Rational Rose[8] är ett annat verktyg av samma typ som har denna integrationsmöjlighet.

JBuilder var däremot ett utmärkt verktyg för programmeringen. Det fanns många väl genomtänkta funktioner, som man kunde använda om man gjort en genomtänkt design.

Applikationsservern Allaire JRun var ett verktyg som inte riktigt höll den kvalitet man kunde ha önskat. Kanske beror det på att den är en relativt billig applikationsserver. Dokumentationen om hur man skulle använda JRun var av ganska dålig kvalitet och det grafiska gränssnittet för driftsättning var ofta buggigt, vilket ledde till en del missförstånd om huruvida vissa fel berodde på utvecklarna (oss) eller själva servern.

6.4 Slutsats

Vår problemställning löd:

På vilka sätt är det bättre att använda ramverket Enterprise JavaBeans för att bygga ett distribuerat, flerskiktat system än att utveckla samma funktionalitet i Java utan att använda något ramverk?

[8] http:// www.rational.com/products/rose/index.jsp

Vår slutsats är att EJB-ramverket är bättre för att utveckla distribuerade, flerskiktade system på många sätt än traditionell systemutveckling i Java. Detta är det främst på grund av de tjänster som ramverket erbjuder utvecklaren. Ramverket tillhandahåller utvecklaren med en kontrollerad och lätt konfiguerbar miljö. Användandet av en applikationsserver gör också att systemen blir mycket skalbara.

För att kunna implementera en effektiv flerskiktad arkitektur krävs främst tjänsterna för distribuerade objekt och namngivning. Dessa tjänster återfinns, som vi visat, även i äldre teknologier som CORBA och Java RMI, men dessa teknologier kräver att utvecklaren själv skapar ett ramverk för att hantera samtidighet och beständighet i applikationen.

Den kanske absolut största fördelen med att använda EJB som ramverk för ett distribuerat objektsystem är att det är så mycket lättare att använda sig av än de tidigare nämnda teknologierna på grund av de övriga CTM-tjänster som ramverket erbjuder. Både teorin, samt vår och andras erfarenhet visar att det är mycket enklare att skapa flerskiktade system med distribuerade objekt med EJB än att använda sig av traditionella Java-tekniker samt RMI eller CORBA. Det har visat sig vara både lättare att skriva koden för systemet, samt att driftsätta det.

I system med en komplex verksamhetslogik och en transaktionstät miljö anser vi att EJB visar att sig från sin bästa sida. Då transaktioner och samtidighet är besvärligt att hantera i systemutveckling leder denna automatiska hantering av dessa problem till en avsevärt kortare utvecklingstid och ett mer driftsäkert system.

När det gäller säkerhet tillför EJB inte så mycket i sig. Även beständighetstjänsterna är fortfarande en svaghet om man vill kunna göra portabla system. Detta kommer radikalt att förändras i det nya specifikationen då strukturen för containterhanterad beständighet gjorts om.

EJB lämpar sig dock inte för alla typer av projekt och systemutvecklaren bör förvissa sig om att så är fallet innan denne väljer att implementera ett system med hjälp av tekniken. I fallet med vår egen prototyp hade det antagligen inte varit nödvändigt egentligen. Detta beror på att många av de tjänster som EJB ger i form av säkerhet, beständighet, samtidighet och transaktioner inte är ett nödvändigt behov för många system, eftersom problem inte kommer att uppstå på grund av brist på dessa tjänster. Men alla system som utvecklas med EJB kan dra fördel av minst några av de tjänster som ramverket erbjuder.

7 Referenser

DeMichiel L G, Yalçinalp L Ü, Krishnan S (2001), *Sun Microsystems Enterprise JavaBeans ™ Specification*, Version 2.0 Proposed Final Draft 2.

Duplancic A, Lindberg K (1998), *Technologies in Self Provisioning Applications*, Institutionen för för Informatik, Handelshögskolan Göteborgs Universitet.

Easterby-Smith et al. (1991), *Management Research: An Introduction*, Sage Publications.

Fowler M, Scott K (1999), *UML Distilled, Second Edition: A Brief Guide to the Standard Object Modeling Language*, Addison Wesley Longman.

Gamma E, Helm R, Johnson R, Vlissides J (1994), *Design Patterns - Elements of Reusable Object-Oriented Software,* Addison Wesley Longman.

Gosling J et al. 2000, *The Java Language Specification, 2nd Edition*, Sun Microsystems, Addison-Wesley Publishing.

Hansson M (1999) *Komponentbaserad systemutveckling - En jämförande introduktion till dagens komponenttekniker*, Institutionen för Data- och Systemvetenskap, Kungliga Tekniska Högskolan.

Hemrajani, A. (1999) *The state of Java middleware, Part 2: Enterprise JavaBeans*. ITworld.com, Inc, URL: http://www.javaworld.com/javaworld/jw-04-1999/jw-04-middleware.html

Jakobsson Olars, H (2000) *Färdig Windows Net om tidigast två år*. Computer Sweden 2000-09-15. URL: http://nyheter.idg.se/display.pl?ID=000915-CS1

Janson F, Zetterquist M (2000) *CORBA vs. DCOM*, Kungliga Tekniska Högskolan.

Kjellgren, P (1998), *Jämförelse av standarder för interaktivitet mellan distribuerade objekt* (HS-IDA-EA-98-114), Institutionen för datavetenskap, Högskolan i Skövde.

Krishnan Sanjeev (1999), *Sun Microsystems Enterprise JavaBeans to CORBA Mapping,* Sun Microsystems. URL: http://java.sun.com/products/ejb/docs.html

Larman, C. (1998), *Applying UML and Patterns*. Upper Saddle River: Prentice-Hall, Inc.

Lundberg C (1998), *Domain Specific Software Architectures: Object-Oriented Frameworks*, ARCS research group, University College of Kalmar, Department of Technology (en del av *Research Report 8/98, Software Architecture - An Overview of the State-of-the-Art* by Jan Bosch (Editor), Department of Computer Science and Business Administration, University of Karlskrona/Ronneby ISSN 1103-1581, *ISRN HK/R-RES-98/8-SE*)

Matena V, Hapner M (1999), *Enterprise JavaBeans™ Specification*, v1.1 Final Release.

Monson-Haefel R, *Enterprise JavaBeans*, 2nd Edition, O'Reilly & Associates.

Sousa J P, Garlan D (2000), *Formal Modeling of the Enterprise JavaBeans TM Component Integration Framework*, CMU-CS-00-162, School of Computer Science, Carnegie Mellon University Pittsburgh USA.

Svensson, D. (1999) *Kloka "mellanskiktsprogram" ska göra nätet smartare*. Datateknik 3.0 1999-02-25. URL: http://www.datateknik30.se/mallar/11lookutskriftsutseende.asp?art_id=2744

Trost, J. (1997) *Kvalitativa intervjuer*. Studentlitteratur.

Tångring, J. (2000) *Java är Rock'n'Roll*. Datateknik 3.0. 2000-05-25. URL: http://www.datateknik30.se/mallar/11lookutskriftsutseende.asp?art_id=4297

Zona Research, Inc. (1999) *Enterprise JavaBeansTM Technology — A Business Benefits Analysis*. URL: http://www.java.sun.com/products/ejb/pdf/zona.pdf

8 Bilagor

8.1 Bilaga 1 Systemutvecklingsmetoden

- Utveckling av prototypen, en utförligare beskrivning

Litteratur

Den UML-notation som jag använder mig av finns beskriven i UML Distilled av M Fowler och K Scott. metoden följer i stort den metoden som beskrivs i Applying UML and Patterns av C Larman. I denna finns det även beskrivet hur man skall driva mer omfattande utveckling under flera år med flera parallella projekt. Dessa delar annat och som inte är naturligt att tillämpa på en mindre underökning som vår kommer vi inte att använda (och de finns därför inte beskrivna).

Översikt

Först utförs en planerings- och beredningsfas. Sedan utförs ett antal utvecklingsfaser.

Planerings- och beredningsfasen

Planerings- och beredningsfasen, resultat

Planerings- och beredningsfasen ger ett antal resultat: kravspecifikation användarmönsterspecifikationer samt en skiss av den konceptuella modellen. I vanliga fall ingår även glosor vilket vi valde att inte göra.

Att få och formalisera den grundläggande informationen från beställaren

Den grundläggande beskrivningen av systemet är beroende på problemformuleringen och syftet med examensarbetet. Genom samtal med beställaren (Xdin AB) får vi fram en översiktlig beskrivning av systemet som kompletterar och innefattar detta. Genom samtalen får vi även fram vilka överordnade mål han har som kompletterar och innefattar de mål som är härledbara ur problemformuleringen. Genom ytterligare samtal med beställaren formaliserar vi hans önskningar i form av systemfunktioner och systemattribut.

Identifieringen av användarmönsterspecifikation

Det finns två metoder att identifiera användarmönsterspecifikationer enligt Larman (1998, s 53):

1. Identifiera aktörerna först. För varje aktör, identifiera vilka processer han initierar eller deltager i.
2. Identifierar yttre händelser som systemet måste reagera på. Relatera händelserna till aktörer och användarmönsterspecifikationer.

När vi inte redogör för något annat har vi använt oss den första metoden.
Alla systemets funktioner som identifieras i kravspecifikation tilldelas användarmönsterspecifikationer.

Skissandet av en Konceptuell modell

Under planerings- och beredningsfasen görs en skiss av den konceptuella modellen. Skissen görs med syftet att skapa en grundläggande förståelse av vokabulären och koncepten som används i problemområdet. Vi går igenom följande faser (som finns beskriva under rubriken Analysen nedan):

- Identifiera konceptuella klasser (typer)

- Rita klasserna i en konceptuell modell
- Addera associationer mellan de konceptuella klasserna (typerna)
- Ge associationens roller dess egenskaper
- Namnge associationer
- Addera attribut
- Skapa subtyp/supertyp

Denna genomgång görs selektivt med syfte att bygga en konceptuell modell som enbart ger grundläggande förståelse av vokabulären och koncepten.

En konceptuell modell illustreras i UML av ett statiskt strukturdiagram utan några operationer. Vi kommer i fortsättningen att kalla det klass diagram. (Senare när vi skriver om klassdiagram under designfasen kommer det att innefatta även operationerna.)

Fördelningen av användningsscenarierna mellan cyklerna

Varje cykel tilldelas ett antal användarscenarier Larman (1998, s 75).

Prioritering av användarscenarier

Innan man fördelar användningsscenarierna mellan cyklerna gör man först en prioritering mellan dem. De prioriteras beroende på:

- Stort inflyttande på arkitekturell design
- Mycket insikter i designen kan nås med liten insats

Systemutvecklingsscenarier

Det finns krav som inte hör hemma i ett användarscenario och de kan riskera att falla mellan stolarna när man bygger ett system. Ett sätt att hantera dem är att använda systemutvecklingsscenarier. Dessa kan t. ex. innefatta saker som att ”designa arkitekturella lager” eller ”utveckla ett system för datalagring” och då fördelas mellan cyklerna precis som användarscenarier.

Tidsuppskattning av användarscenarier

För varje användarscenario och systemutvecklingsscenario görs en tidsuppskattning. Sedan görs en bedömning av hur lång cykeln skall vara. Vi har valt att ha tvåveckorscykler. Slutligen fördelas användarscenarierna mellan cyklerna.

Utvecklingscykler

Övergripande

En utvecklings cykel består av följande faser: förbättra planerna, synkronisera dokumentationen, analys, design, konstruktion och test.

Förbättra planer

Enlig metoden gör man i varje utvecklingscykel en revidering av tidsplanen och fördelning av användarscenarier mellan utvecklingscyklerna. Om man vill ha tiden för en utvecklingscykel fast och man märker att man inte hinner med allt man planerat finns det två val: fler arbetare i projektet eller att man skjuter till något/några användarscenarier. I ”Mythical Man-Month” (Brooks, 1975) gör man en liknelse: ”nio

kvinnor föder inte ett barn på en månad". Det finns en likhet här d.v.s att flytta användarscenarier till en senare cykel är mera realistiskt. Vi har valt att inte göra på det här sättet utan att hålla på vår ursprungliga fördelninga av arbetet mellan cyklerna. Eftersom cyklerna också är basen för vår jämförelse mellan EJB och referensalternativ.

Synkronisera dokument

De dokument så som användarscenario, klassdiagram e t c som bearbetats under föregående cykel gör vi så att de överensstämmer med varandra vi med varandra.

Analys

Övergripande om analys

Under analysfasen utarbetar vi essentiella användarscenarier, förbättrande statiska strukturdiagram av konceptuella klasser,"systemsekvensdiagram", kontrakt för systemoperationer, tillståndsdiagram för konceptuella klasser.

Definiera essentiella användarmönsterbeskrivningar.

Expanderade essentiella användarmönsterbeskrivningar görs av de användarmönsterbeskrivningar som skall behandlas i den aktuella utvecklingscykel. (Under planerings och bearbetnings fasen skrevs de som högnivåbeskrivningar.)

Förbättra den konceptuella modellen

Den konceptuella modelleringen som gjorts under planerings- och beredningsfasen förbättras.

Identifiera konceptuella klasser (typer)

Genom att ta de ord som används i det gamla systemet som vi skall ersätta och genom samtal med beställaren får vi fram ett antal substantiv. Vi använder de namn som används i problemområdet (kundens namn). Sen tar vi bort det som inte relevant. Vi lägger till specifikationsklasser om:

- Att man tar bort sista instans av en klass leder till att man förlorar information som skulle ha behövts
- Det minskar redundansen i informationen.

Observera att vi inte tar bort klasser bara för att de kanske inte innehåller information som behöver sparas, eller för att de inte har några attribut. (Det är bättre att ha en modell med många klasser än att underspecificera.)

Rita klasserna i en konceptuell modell

Vi ritar in klasserna i en konceptuell modell där bara klassernas namn framgår.

Addera associationer mellan de konceptuella klasserna

Enligt (Larman, 1998, s 108) fram associationer genom att använda vanliga associationslistor (Larman, 1998, s 108) och genom att tänka igenom vilken kunskap om relationen måste bevaras för en längre tid. Vi har inte använt oss av associationslistor. Det är mycket viktigare att hitta klasser än att hitta associationer (Larman, 1998, s 110).

Ge associationens roller dess multipliciteten

Vi ger associationens roller dess egenskaper när det gäller multipliciteten (Larman, 1998, s 110).

Namnge associationer

Vi ger associationen ett namn om det hjälper att förklara relationen (Larman, 1998, s111).

Addera attribut

Vi adderar de attribut som användarscenarierna ”föreslår” eller implicit antar att objektet behöver komma ihåg. Attribut låter vi bara rena datavärden bli. Data som behöver innehålla ett identifierande attribut får bli egna klasser. Om vi är tveksamma för vi in ett attribut till en separat klass. Vi låter attributen ha en icke primitiv datatyp om (Larman, 1998, s124):

- det består av flera sektioner
- det finns operationer knutna till det
- det har egna attribut
- det är en kvantitet med en enhet

Lägg till delade aggregat eller komposit-aggregat

Vi överväger att visa aggregation:

- när livstiden hos delen är bunden till livstiden av helheten
- när det är en uppenbar del-helhet relation, fysisk eller logisk
- när några egenskaper hos helheten (composite) används av alla delarna
- när några operationer då de appliceras på helheten automatisk appliceras på delarna

Lägg till härledda element

Vi lägger till härledda element när de är framstående i den terminologi som används i problemområdet (domänen) och när det inte minskar förståelsen att ha med dem. I övrigt undviker vi att visa härledda element eftersom de ökar beskrivningens komplexitet utan att tillföra någon ny information.

Systemsekvensdiagram

Definiera systemsekvensdiagram

Vi gör ett systemsekvensdiagram för varje vanlig sekvens av händelser för alla användarscenarier som hanteras i den aktuella utvecklingscykeln. Vi gör systemsekvensdiagrammet på följande sätt (Larman , 1998, s140):

- Rita systemet som en ”black box”
- Identifiera alla aktörer som opererar på systemet rita och rita en linje för varje aktör
- Identifierar alla yttre systemhändelser som aktören genererar. Visa dem i diagrammet som händelser

- Vid behov av förtydligande: skriv in användarscenarioetexten till vänster om diagrammet. Namnet på en händelse skall utrycka syftet/målet med händelsen. Det skall inte säga något om (det grafiska) gränssnittet.

Definiera systemoperationskontrakt

Vi skapar ett systemoperationskontrakt för varje systemoperation i systemsekvensdiagrammen som gjort tidigare under samma utvecklingscykel. För varje systemoperationskontrakt skriver vi (Larman, 1998, s149):

- Först ansvarssektionen.
- Sen post-condition sektionen. Det är viktigt att post-conditions är skrivna på ett kungörande och tillståndsorienterat sätt (inte på ett händelseorienterat sätt)
- Sist pre-conditions.

Definierar tillståndsdiagram

Ett objekt som alltid regerar på samma sätt på en händelse kan man kalla tillståndsoberoende. Ett tillståndsdiagram kan reagera på tre typer av händelser:

- yttre händelse == system händelser
- inre händelser == orsakade av något innanför systemgränsen
- temporala händelser == orsakade av att en viss tidpunkt inträffat eller en viss tid har förflutit.

Vi gör tillståndsdiagram för alla tillståndsberoende klasser som har ett komplext beteende enbart baserat på yttre händelser och temporala händelser. Klasser som kan vara tillståndsberoende när det gäller yttre och temporala händelser är:

- fönster
- applikationskoordinerare (t. ex. ”Applets” i Java)
- controllers == klasser som är ansvariga för att hantera systemhändelser
- transaktioner (t ex försäljning).

I vårt system fanns det inga tillståndberoende klasser.

Design

Definiera verkliga användarmönsterspecifikationer

Gör verkliga användarmönsterspecifikationer för alla användarscenarier i den aktuella utvecklingscykeln.

Definiera användargränssnitt.

Gör bilder av hur användargränssnitten skall se ut.

Förbättra systemarkitekturen

Systemarkitekturen bör utvärderas iterativt för varje ny utvecklingscykel. Man bör bestäma vilken övergripande typ av arkitektur som passar bäst: tvåskikts, treskikts, fyrskikts etc. Klasserna bör grupperas i paket utifrån följande riktlinjer:

- Gemensam tjänst, med relativ hög coupling[9] och collaboration[10]
- På en mera abstrakt nivå kan man se paketet som cohesive – det har relaterade ansvar.
- Coupling och collaboration mellan klasser i olika paket är låg.

Sedan bör man rita ut synligheten mellan lagren i form av dependency-pilar[11].

Vi har valt att inte följa Larman(1998) i detta utan utan i bestämt systemarkitekturen tidigare utifrån de önskemål som Xdin har haft (fyrskiktsarkitektur för cykel 3). Men vi har trots detta grupperat klasserna enligt ovanstående kriterier. Men på grundval av våra riktlinjerna från Xdin ligger alla klasser i samma paket. (En klass flyttades ut vid implementationen).

Definiera collaboration-diagram

För varje systemoperationskontrakt skall det skapas ett collaboration-diagram (Larman, 1998, s218). Börja med den yttre systemhändelsen. Ta utgångspunkten i vilket ansvar kontraktet har, kontraktets post-condition samt i användarsceneriet. Designa sedan ett system av interagerande objekt som uppfyller uppgiften.

Definiera design-klassdiagram

Skapa ett design-klassdiagram genom att:

- Identifiera alla klasser som skall ingå genom att analysera collaboration-diagrammet.
- Rita in klasserna i ett klassdiagram.
- Skriv över attributen från motsvarande klasser i det konceptuella klassdiagrammet.
- Addera metodnamn till klasserna genom att analysera collaboration-diagrammet.
- Addera typer till attribut och metoder.
- Addera dependency-relations-linjer för att indikera icke-attributsynlighet.

---

[9] Coupling (koppling) är hur när två klasser hänger samman. Det finns beskrivet i Mathiassen e t c (1998). Hög koppling byter att om vi ändra i den ena klassen måste vi också ändra i den andra. Den starkaste kopplingen är när en klass som inte är en subklass av en annan klass refererar till privata egenskaper i denna. Denna konstuktion går att göra i C++ (med friends) men inte i java. Den näst starkaste kopplingen är när en klass som ärver en annan klass refererar direkt till dess privata egenskaper. (Vilket är fallet om man deklarera attributen som protected i C++). Den tredje starkaste kopplingen är om en metod refererar direkt till privata egenskaper i ett objekt i samma klass (eller i klassen). Det är vanligt i java och C++. (Men i Smalltake går det inte gör om variablerna är privata.) Ytterligare svagare är kopplingen när en klass refererar till den offentliga operationer i en annan klass.

[10] Collaboration (samhörighet) finns beskrivet i finns beskrivet i Mathiassen e t c (1998). Det är hur väl en komponent (eller klass) hänger samman. De egenskaper som vis ar på hög samhörighet i en komponent är: a)komponetens klasser är begreppsmässigt besläktade b)de strukturella förbindelserna mellan klasserna är generalisering och aggregat c) centrala operationer kan utföras inom komponeten.

[11] Dependency (beroende) mellan två paket existera om någon förändring i det ena paketet gör att något måste ändra i det andra. Dependecypilen visa är en streckad linje med en öppen pil (V-formad).

8.2 Bilaga 2 Notationsbeskrivning

Denna bilaga beskriver övergripande den notationen vi använt oss av.

Kravspecifikation

Översiktligt beskrivning av systemet

En översiktligt beskrivning av systemet i en till tre meningar.

Kunder

Kundens namn och kort beskrivning av kunden.

Systemfunktioner

Allt systemet skall göra. Ingen förklaring på vems initiativ systemet skall göra det. Ingen förklaring hur systemet skall göra det. Inte någon utförlig förklaring vad systemet skall göra bara ett antal satser typ:

Systemet skall göra <X1>

Systemet skall göra <X2>

Systemattribut

Med systemattribut menar vi karakteriseringar av systemet som inte är funktioner t. ex.: responstid, interface-metafor, feltolerans, begräsning av vilka operativsystem det skall gå att köra på. Attribut som hör samman med en viss systemfunktion skall skrivas tillsammans med den.

Beskrivning av olika typer användarmönsterspecifikationer

Högnivå-användarmönsterspecifikation

En högnivå-användarmönsterspecifikation består av:

Namn: Namnet på användarmönstret

Aktör: Namnet på aktörerna

Typ: Primär/sekundär/ej obligatoriskt

Kors ref: Referenser till bland annat systemfunktioner som behandlas i kravspecifikation

Beskrivning: En kortfattad beskrivning på två eller tre meningar.

Expanderad användarmönsterspecifikation

En utökad användarmönsterspecifikation består av:

Namn: Namnet på användarmönstret

Aktör: Namnet på aktörerna

Syfte: Syftet med användarmönstret.

Beskrivning: En kortfattad beskrivning på tre till fyra meningar.

Typ: Primär/sekundär/ ej obligatoriskt och essentiell/verklig

Kors ref: Referenser till bland annat systemfunktioner som behandlas i kravspecifikation

Typiskt händelseförlopp:

Beskrivning av ett typiskt händelseförlopp där aktörens agerande beskrivs omväxlande med systemets respons. Agerande att respons är numrerade. Första händelsen börjar med: Det här användarscenariot börjar när <aktör> <start händelse>.

Alternativa förlopp:

Beskrivning av alternativa händelseförlopp.

Primära, sekundära och ej obligatoriska användarmönsterspecifikationer

Primära användarmönsterspecifikationer representerar de huvudsakliga processerna. Sekundära användarmönsterspecifikationer representerar de mindre eller ovanliga processerna. En ej obligatorisk användarmönsterspecifikation representerar de processer som kanske inte behöver behandlas.

Essentiella respektive verkliga användarmönsterspecifikation

En essentiell användarmönsterspecifikation är fri från teknologi- och implementationsdetaljer. Designbeslut har undvikits. En verklig användarmönsterspecifikation beskriver processen i termer av den beslutade designen t. ex. valda input och output-teknologier. Expanderade användarmönsterspecifikation kan vara essentiella eller verkliga. Högnivå- användarmönsterspecifikationer är alltid essentiella.

Sektioner i användarmönsterspecifikationen

Ibland är det så att vi vid en punkt i användarmönsterspecifikationen finns flera alternativ som är ganska normala och inträffar ungefär lika ofta. Då används följande notationsstruktur:

I huvudsektionen “typiskt händelse förlopp“ hänvisas till olika subsektioner. Skriv en subsektion för varje alternativ där “typiskt händelse förlopp“ ingår i varje. Varje subsektion börjar numreras från ett. Om en subsektion har alternativ skriv dem i “alternativa förlopp“ för respektive subsektion. Ex:

Sektion: Huvudsektion

Typiskt händelse förlopp:

… 5

Om G se sektion X

Om H se sektion Y

6 …

Alternativa förlopp

Sektion: X

Typiskt händelse förlopp:

Alternativa förlopp

Sektion: Y

Typiskt händelse förlopp:

Alternativa förlopp

Beskrivning av systemsekvensdiagram

Systemsekvensdiagram är en form av sekvensdiagram där den anropande agenten är en aktör inte ett objekt och den anropade agenten är systemet sett som en "black box". Systemsekvensdiagrammet visar för ett visst användarscenario:

a) händelserna som det externa systemet genererar i form av meddelande från aktör till systemet

b) ordningen som händelserna inträffar i, uppifrån och ner

c) ev. händelser mellan system.

Meddelanden inuti systemet visas inte. Aktören visas med en streckgubbe. Systemet symboliseras av en rektangulär box med :System understryket (D v s en "instans" av systemet"). Om vi har flera system är de numrerade. Istället för livslinje finns det en prickad linje under aktören för de respektive systemen.

Namnet på meddelandet är även namnet på händelsen som gör att metoden anropas. Meddelandet kallas systemoperation eftersom det reagerar på en systemhändelse. Till vänster om linjen som symboliserar meddelandet skrivs text som förtydligar logik, upprepning o s v från användarscenariot. Ex. på text från användarsceneriet kan vara: "upprepa tills dess att kunden inte vill köpa mera varor".

Design by contract

Centralt i tekniken "Design by contract" är begreppet assertion (ung. *bekräftelse*). Assertion är ett villkor som kan utvärderas till sant eller falskt. I "Design by contract" finns det tre olika sorters villkor: post-condition, pre-condition och invariants.

Post-condition ett är en redogörelse för hur världen skall se ut efter en operation har utförts.

Pre-condition är hur operationen förutsätter att värden skall se ut innan vi den utförs. Invariants är assertion om en klass. Det kan vara ett villkor som alltid skall stämma för alla instanser för en klass. D v s det skall alltid stämma om en instans är tillgänglig för metodanrop. De innebär att det skall gälla både som pre- och post-condition. Det är viktigt att man alltid förstärker eller bibehåller de assertion som finns i den klass som man specialiserar. Man skall aldrig försvaga assertion när man specialiserar en klass.

Beskrivning av operationskontrakt

Ett operationskontrakt beskriver förändringar i tillståndet av ett system, en klass eller ett interface när en händelse inträffar. Ett operationskontrakt har följande beståndsdelar:

- Namn
- Ansvar
- Typ
- Korsreferenser
- Noteringar
- Undantag
- Pre-conditions
- Post-conditions

Namnet är namnet på operationen och dess parametrar (med typer).

Ansvar beskrivs informellt.

Typen är namnet på typens klassnamn, interfacenamn eller "Systemet".

Korsreferenser är referenser till systemfunktioner, användarscenarier e t c.

Noteringar kan vara algoritm som skall användas, design-noteringar e t c.

Undantag är tillstånd då det skall visas att det inträffat någon form av fel eller undantagstillstånd.

För allmän beskrivning av pre- och post-condition se "Design by contract" ovan.

Användbara post-condition-kategorier är skapande och borttagandet av objekt, modifierandet av attribut, skapandet och borttagandet av associationer.

8.3 Bilaga 3 Planering- och beredningsfasen

Övergripande mål

Göra en ny “news”-funktion för xdins intranät.

Systemfunktioner

Kunna lägga in nyheter.

Kunna titta på nyheter.

Systemattribut

GUI med hjälp av JPS och Servlet. (JSP: view, Servlet: controller).

Samtidighet: flera samtidiga användare. (T e x att någon lägger in nyheter samtidigt som det visas nyheter).

Aktörer:

- Nyhetsläsare
- Nyhetsinläggare
- Administratör

Användarscenarier

Titta på aktuella nyheter

Aktör: nyhetsläsare

Användaren väljer “visa nyheter”.

Alla nyheter som är aktuella tidsmässigt och avdelningsmässigt visas.

En länk till författarens "personprofilsida" visas för varje nyhet.

Titta på andra gruppers/avdelningars nyheter

Aktör: nyhetsläsare

Användaren väljer att se på nyheter från en viss grupp eller avdelning.

Markera nyhet som läst

Aktör: nyhetsläsare

Användaren väljer en eller flera nyheter och väljer att markera dem som lästa. De kommer då inte upp vid nästa nyhetslistning.

Titta på privata nyheter

Aktör: nyhetsläsare

Användaren väljer “visa privata nyheter”.

Alla privata nyheter som är aktuella visas.

Skapa nyhet

Aktör: nyhetsinläggare

Användaren väljer “skapa nyhet”.

Fyller i rubrik, text, privat/ej privat, avdelningsintern/icke avdelningsintern, avdelning/grupp, www-länk, slutdatum.

Man skall kunna välja att göra en nyhet synlig för en eller flera grupper/avdelningar.

Ta bort egen nyhet

Användaren väljer en av sina egna nyheter och väljer att ta bort den.

Aktör: nyhetsinläggare

Ta bort andras nyheter

Aktör: administratör

Administratören väljer att radera en nyhet ur systemet.

Lägg till/ta bort/ändra person

Aktör: administratör

Administratören lägger in en ny person eller väljer att ta bort en existerande.

Administratören kan också ändra uppgifter för en person.

(Den här funktionaliteten finns redan i xdins intra. Vi kommer att använda den befintliga funktionaliteten när vi jobbar med EJB.)

Lägg till/ta bort/ändra avdelning/grupp

Aktör: administratör

Administratören lägger in en ny avdelning/grupp eller väljer att ta bort en existerande.

Administratören kan också ändra uppgifter för en avd/grupp.

(Den här funktionaliteten finns redan i xdins intra. Vi kommer att använda den befintliga funktionaliteten när vi jobbar med EJB.)

Fritextsökning

Aktör: nyhetsläsare

Användaren fyller i et ord och söker på det. Alla nyheter som användaren har behörighet att läsa visas.

Skapa nyhet expanderad: koppla dokument

Aktör: nyhetsläsare

Som ”skapa nyhet” men även möjlighet att koppla dokument till nyheten.

Systemutvecklingsscenarier

Fleranvändarapplikation utan EJB

Göra så att applikationen fungerar för flera användare utan att den byggs med hjälp av Enterprise JavaBeans.

EJB

Göra så att applikationen fungerar för flera användare genom att den byggs med hjälp av Enterprise JavaBeans i en fyrskiktsarkitetur.

8.4 Bilaga 4 Analys- och design-dokument

Analys

Essentiella Användarscenarier

Grund

Logga in

Aktör: Nyhetsläsare

Syfte: Titta på de nyheter som är aktuella.

Beskrivning: Användaren loggar in och tittar på alla nyheter som är aktuella tidsmässigt och gruppmässigt.

Typ: expanderat essentiellt.

Kors ref:

Typiskt händelseförlopp:

Aktörens *handling*	Systemets gensvar
1. Användaren loggar in.	2 Alla nyheter som är aktuella tidsmässigt och för de *grupper* som användaren tillhör visas.. En länk till författarens "personprofilsida" visas för varje nyhet.

Titta på aktuella nyheter

Aktör: Nyhetsläsare

Syfte: Titta på de nyheter som är aktuella.

Beskrivning: Användaren tittar på alla nyheter som är aktuella tidsmässigt och gruppmässigt.

Typ: expanderat essentiellt.

Kors ref:

Typiskt händelseförlopp:

Aktörens *handling*	Systemets gensvar
1. Användaren väljer ”visa alla nyheter”..	2 Alla nyheter som är aktuella tidsmässigt och för de *grupper* som användaren tillhör visas.. En länk till författarens "personprofilsida" visas för varje nyhet.

Titta på privata nyheter

Aktör: Nyhetsläsare

Syfte: Titta på de privata nyheter som är aktuella.

Beskrivning: Användaren tittar på alla privata nyheter som är aktuella tidsmässigt och gruppmässigt.

Typ: expanderat essentiellt.

Kors ref:

Typiskt händelseförlopp:

Aktörens handling	Systemets gensvar
1. Användaren väljer privat	2. Alla nyheter som är akutella tidmässigt och gruppmässigt visas med den senast inlagda nyheten högst upp. En länk till författarens "personprofilsida" visas för varje nyhet.

Skapa nyhet

Aktör: Nyhetsinläggare

Syfte: Lägga till en nyhet till systemet.

Beskrivning: Användaren skapar en nyhet som skall kunna ses av de specificerarde grupperna.

Typ: expanderat essentiellt.

Kors ref:

Typiskt händelseförlopp:

Aktörens handling	Systemets gensvar
1. Användren väljer lägga in nyheter	2.Systemet presenterar användaren de uppgifter som skall fyllas i: • rubrik • text • *evtuell kategori (privat)* • *klassificerad/ icke klassificerad* • www-länk • slutdatum Användaren skall kunna välja att göra en nyhet synlig för en eller flera grupper/avdelningar.
3. Användaren väljer/fyller i uppgifterna som systet kräver.	
4. Användaren väljer att valda alternativ skall sparas	5 Systemet sparar valda alternativ

	6 Systemet presentera en bekräftelse

Alternativa händelseförlopp

Rad 4: Användaren väljer att avbryta inläggandet av nyhet

Rad 6 Om något alternativ inte kan kan presenteras ett felmedelande.

Ta bort nyhet

Aktör: Nyhetsinläggare

Syfte: Ta bort sin egen nyhet från systemet

Beskrivning: Användaren väljer en av sina egna nyheter och väljer att ta bort den.

Typ: expanderat essentiellt.

Kors ref:

Användarscenarie: Nyhetsinläggare måste utfört användarscenariet ”Titta på aktuella nyheter” eller ”Fritextsökning”

Typiskt händelseförlopp:

Aktörens handling	Systemets gensvar
1. Användaren markerar en eller flera nyheter (endast de egna nyheterna är markerbara)	
2. Användaren väljer att de nyheter som är markerade skall tas bort.	3. Systemet gör så att den nyheten inte lägre kan ses i NEWS. (Men den finns fortfarande lagrad)
	4. Systemet presenterar en bekräftelse

Ta bort andras nyheter

Aktör: Administratör

Syfte: Administratör skall kunna ta bort stötande nyheter

Beskrivning: Administartören tar bort en nyhet från systemet

Typ: expanderat essentiellt.

Kors ref:

Användarscenarie: Administratör måste utfört användarscenariet”Fritextsökning”

Typiskt händelseförlopp:

Aktörens handling	Systemets gensvar
1. Användaren markerar en eller flera nyheter (alla nyheterna är markerbara)	
2. Användaren väljer att de nyheter som är markerade skall tas bort.	3. Systemet gör så att den nyheten inte lägre kan ses i NEWS. (Men den finns fortfarande lagrad)

fortfarande lagrad)

4. Systemet presenterar en bekräftelse

Funktionalitet som bara används i referensversionen

Lägg till/ta bort/ändra person

Aktör: administratör

Syfte: att lägga till eller ta bort en Person ur systemet. Att ändra en persons uppgifter, såsom namn, avdelningstillhörighet etc.

Beskrivning:

Administratören lägger in en ny person eller väljer att ta bort en existerande.

Administratören kan också ändra uppgifter för en person.

(Den här funktionaliteten finns redan i xdins intra. Vi kommer att använda den befintliga funktionaliteten när vi jobbar med EJB. Men när vi gör vår trådningsversion kommer vi att skapa funktionaliteten för det här användarsceneriet själva.).

Typiskt händelseförlopp:

Sektion: Huvudsektion

Aktörens handling	Systemets gensvar
1. Detta händelseförlopp börjar när en Administratör väljer att administrera personalinformationen i systemet.	
2. Administratören väljer att lägga till, ta bort eller ändra en Person.	3. Om ”lägga till, se sektion ”lägga till”. Om ”ta bort”, se sektion ”ta bort”. Om ”ändra”, se sektion ”ändra”.

Alternativa händelseförlopp: -

Sektion: Lägga till

Aktörens handling	Systemets gensvar
1. Börjar när administratören har valt att lägga till en Person i systemet.	
	2. Systemet presenterar Administratören vilka personuppgifter som skall fyllas i: • Förnamn • Efternamn • E-post • Avdelning(Grupp)
3. Administratören fyller i uppgifterna för den nya Personen och anger därefter att han vill spara Personen. Som Avdelning kan endast en existerande	4. Systemet presenterar Administratören med information om det gick bra att lägga till Personen.

Avdelning (Grupp) väljas.	

Alternativa händelseförlopp:

- Rad 3. Administratören väljer att inte lägga till en Person. Scenariot avbryts.

Sektion: Ta bort

Aktörens handling	Systemets gensvar
1. Börjar när administratören har valt att ta bort en Person ur systemet.	
	2. Systemet presenterar Administratören med en listning av de Personer som finns i systemet.
3. Administratören väljer en eller flera Personer att ta bort.	
4. Administratören väljer att ta bort de markerade.	5. Systemet visar om det gick bra att ta bort Personerna.

Alternativa händelseförlopp:

- Rad 4. Administratören ångrar sig och tar inte bort. Scenariot avbryts.

Sektion: Ändra

Aktörens handling	Systemets gensvar
1. Börjar när Administratören har valt att ändra en Persons uppgifter.	
	2. Systemet presenterar Administratören med en listning av de Personer som finns i systemet.
3. Administratören väljer en Person.	4. Systemet presenterar Administratören med Personens uppgifter i editerbart format.
5. Administratören ändrar de aktuella uppgifterna och väljer att uppdatera informationen till systemet.	6. Systemet presenterar Administratören med en bekräftelse på att de nya uppgifterna har införts.

Alternativa händelseförlopp:

- Rad 5. Administratören väljer att återställa uppgifterna. Scenariot börjar på nytt vid rad 4.
- Rad 6. Administratören väljer att avbryta ändringen. Scenariot avslutas.

Lägg till/ta bort/ändra avdelning/grupp (Grupp)

Aktör: Administratör

Syfte: Att administrera de avdelningar och grupper som finns i systemet.

Beskrivning:

Administratören lägger in en ny avdelning/grupp eller väljer att ta bort en existerande.

Administratören kan också ändra uppgifter för en avd/grupp.

(Den här funktionaliteten finns redan i xdins intra. Vi kommer att använda den befintliga funktionaliteten när vi jobbar med EJB. Men när vi gör vår trådningsversion kommer vi att skapa funktionaliteten för det här användarsceneriet själva.)

Typiskt händelseförlopp:

Sektion: Huvudsektion

Aktörens handling	Systemets gensvar
1. Detta händelseförlopp börjar när en Administratör väljer att administrera avdelningar/grupper (hädanefter kallat Grupp) i systemet.	
2. Administratören väljer att lägga till, ta bort eller ändra en Grupp.	3. Om ”lägga till, se sektion ”lägga till”. Om ”ta bort”, se sektion ”ta bort”. Om ”ändra”, se sektion ”ändra”.

Alternativa händelseförlopp: -

Sektion: Lägga till

Aktörens handling	Systemets gensvar
1. Börjar när administratören har valt att lägga till en Grupp i systemet.	
	2. Systemet presenterar Administratören vilka Gruppuppgifter som skall fyllas i. • Gruppnamn • Gruppchef (kan endast välja en i systemet befintlig Person).
3. Administratören fyller i uppgifterna för den nya Gruppen och anger därefter att han vill spara Gruppen.	4. Systemet presenterar Administratören med information om det gick bra att lägga till Gruppen.

Alternativa händelseförlopp:

- Rad 3. Administratören väljer att inte lägga till en Grupp. Scenariot avbryts.

Sektion: Ta bort

Aktörens handling	Systemets gensvar

1. Börjar när administratören har valt att ta bort en Grupp ur systemet.	
	2. Systemet presenterar Administratören med en listning av de Grupper som finns i systemet.
3. Administratören väljer en eller flera Grupper att ta bort.	
4. Administratören väljer att ta bort de markerade.	5. Systemet visar om det gick bra att ta bort Grupperna.

Alternativa händelseförlopp:

- Rad 4. Administratören ångrar sig och tar inte bort. Scenariot avbryts.

Sektion: Ändra

Aktörens handling	Systemets gensvar
1. Börjar när Administratören har valt att ändra en Grupps uppgifter.	
	2. Systemet presenterar Administratören med en listning av de Grupper som finns i systemet.
3. Administratören väljer en Grupp.	4. Systemet presenterar Administratören med Gruppens uppgifter i editerbart format.
5. Administratören ändrar de aktuella uppgifterna och väljer att uppdatera informationen till systemet.	6. Systemet presenterar Administratören med en bekräftelse på att de nya uppgifterna har införts.

Alternativa händelseförlopp:

- Rad 5. Administratören väljer att återställa uppgifterna. Scenariot börjar på nytt vid rad 4.
- Rad 6. Administratören väljer att avbryta ändringen. Scenariot avslutas.

Utökad funktionalitet

Markera nyhet som läst

Aktör: Nyhetsläsare

Syfte: Att markera en Nyhet som läst för att inte behöva se på den igen, förutom i Arkivet.

Beskrivning:

Användaren väljer en eller flera nyheter och väljer att markera dem som lästa. De kommer då inte upp vid nästa nyhetslistning.

Typiskt händelseförlopp:

Sektion: Huvudsektion

Aktörens handling	Systemets gensvar
1. Detta händelseförlopp börjar när en Nyhetsläsare har valt att se på en nyhet eller då han ser på en nyhetslistning med rubriker.	
2. Användaren väljer att markera nyheten som läst.	3. Systemet visar inte nyheten nästa gång användaren listar Nyheter.

Titta på andra gruppers/avdelningars nyheter

Aktör: Nyhetsläsare

Syfte: Att en Person skall kunna läsa Nyheter från andra Grupper än den han själv tillhör.

Beskrivning:

Användaren väljer att se på nyheter från en eller flera grupper. De som är aktuella tidsmässigt visas och behörighetsmässigt (markerade som icke *icke klassificerad* när nyheten skapades) visas.

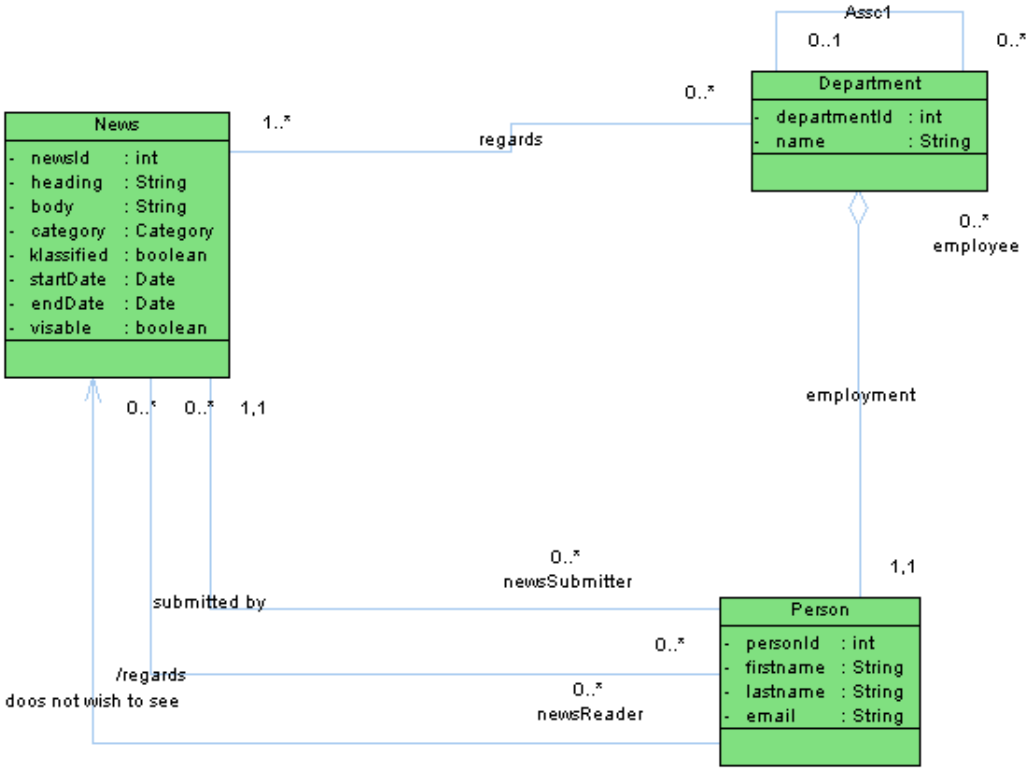
Typiskt händelseförlopp:

Sektion: Huvudsektion

Aktörens handling	Systemets gensvar
1. Detta händelseförlopp börjar när en Nyhetsläsare har valt att titta på nyheter som är avsedda för andra Grupper än den han själv tillhör.	

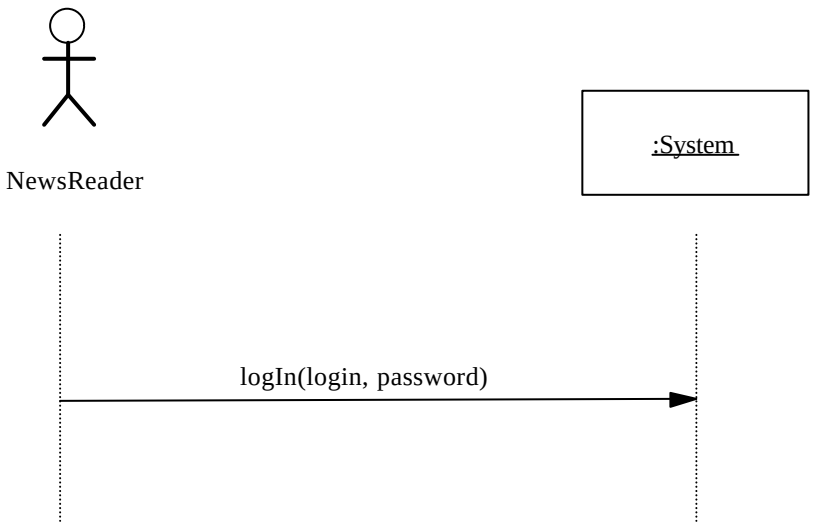
2. Nyhetsläsaren väljer den Grupp vars nyheter han vill titta på.	3. Systemet presenterar en listning av nyheter från den valda gruppen. Endast de nyheter som är markerade som *"ej avdelningsintern"* visas.

Konceptuella klasser

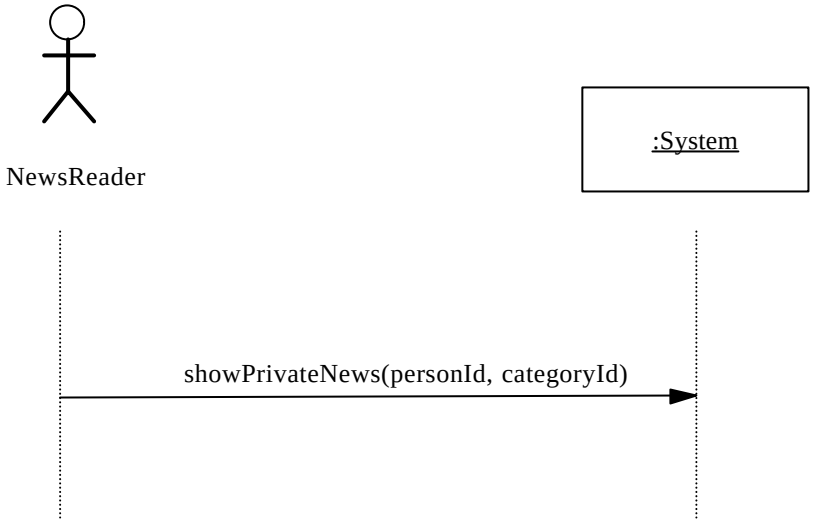


Systemsekvensdiagram

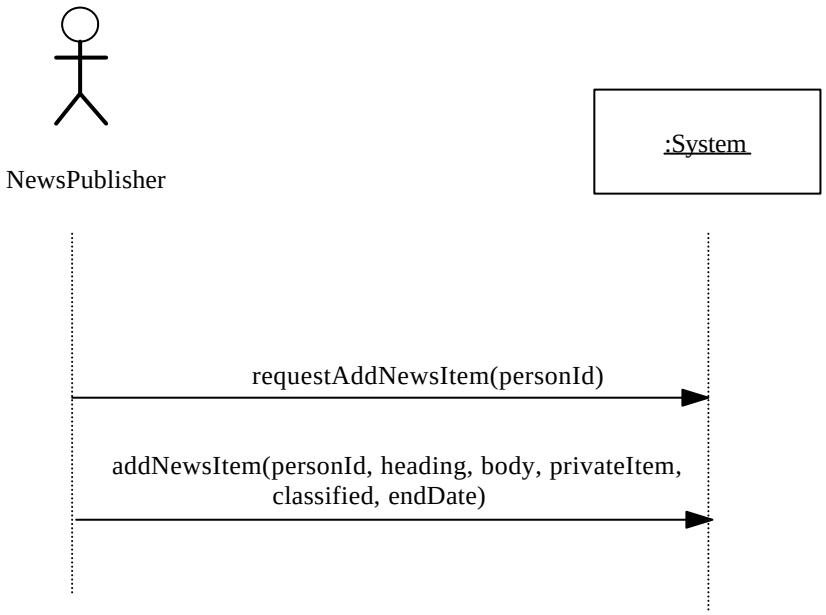
Logga in



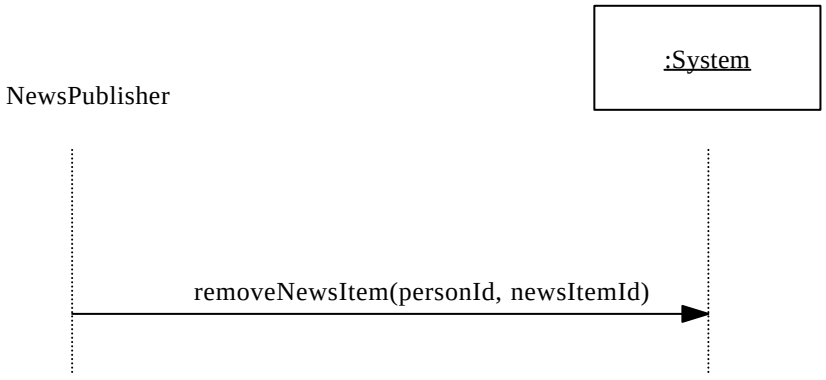
Titta på aktuella nyheter



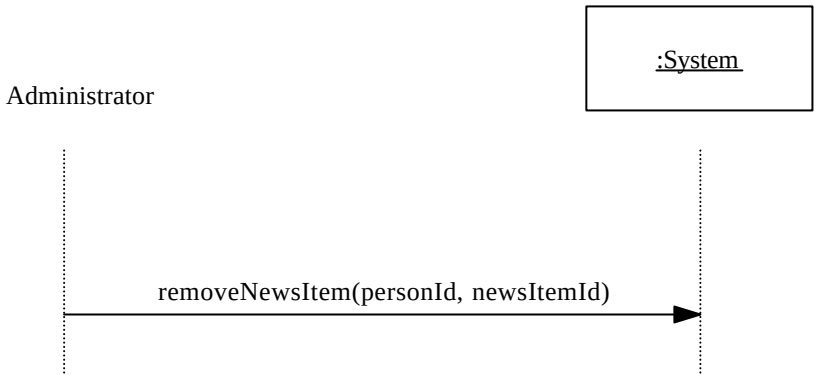
Skapa nyhet



Ta bort egen nyhet

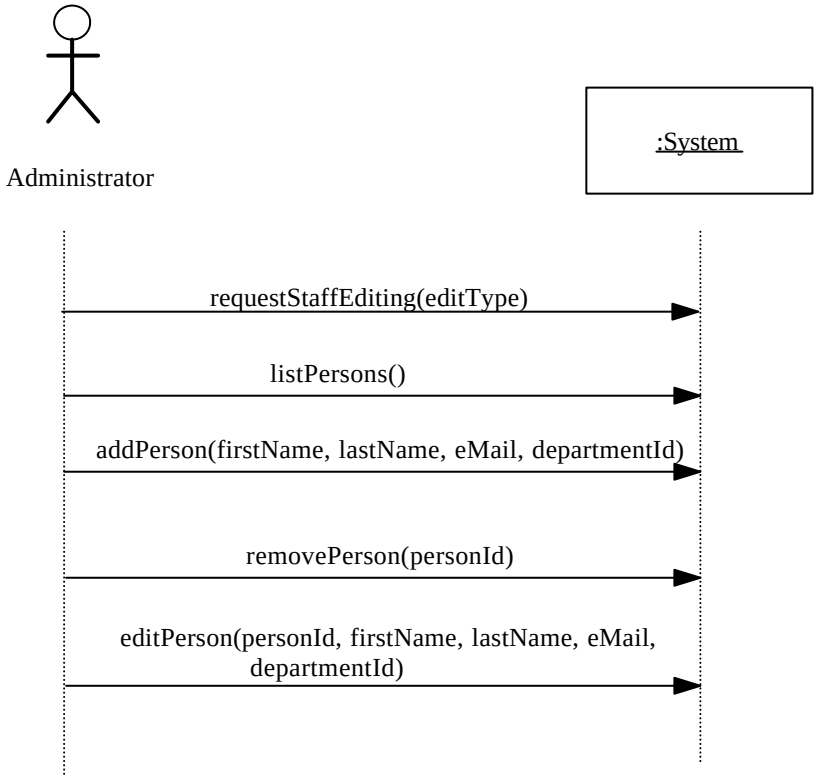


Ta bort nyhet

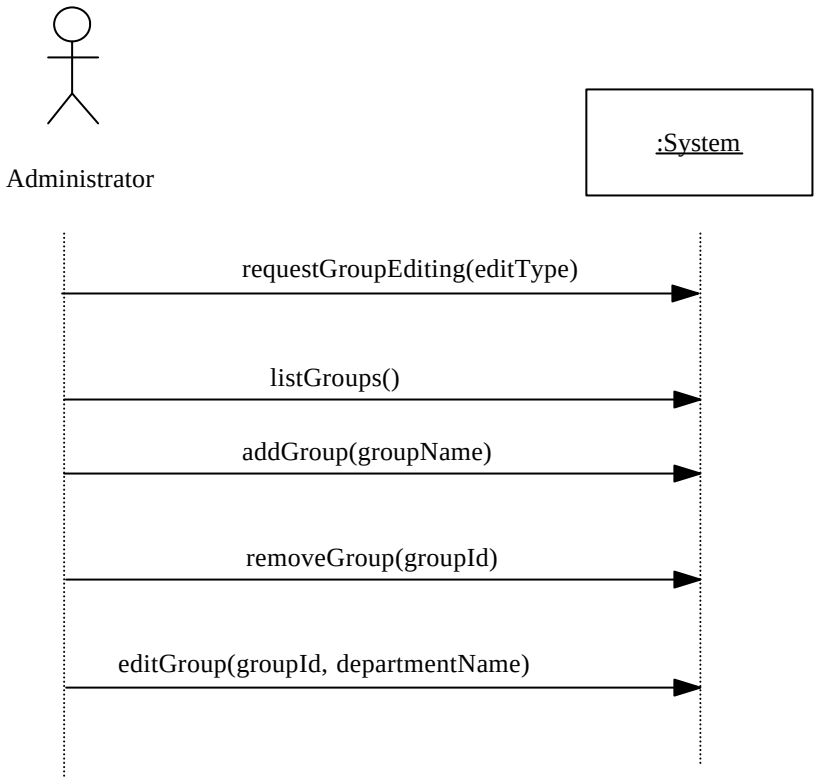


Endast i referensversionen

Lägga till, ta bort eller ändra en Person

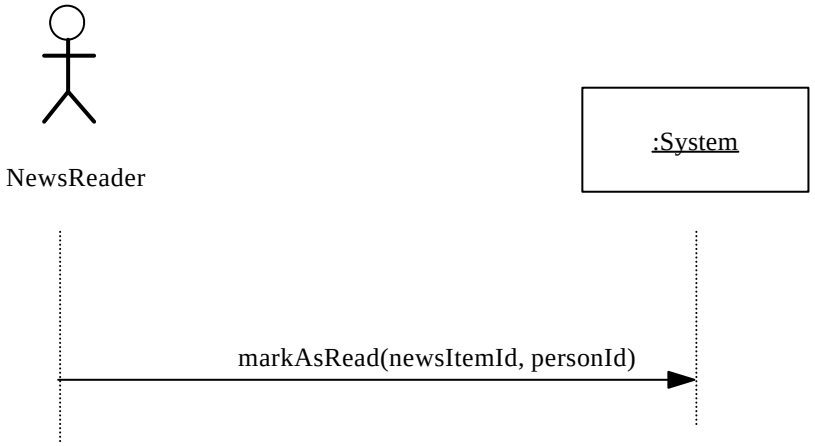


Lägga till, ta bort eller ändra en Grupp

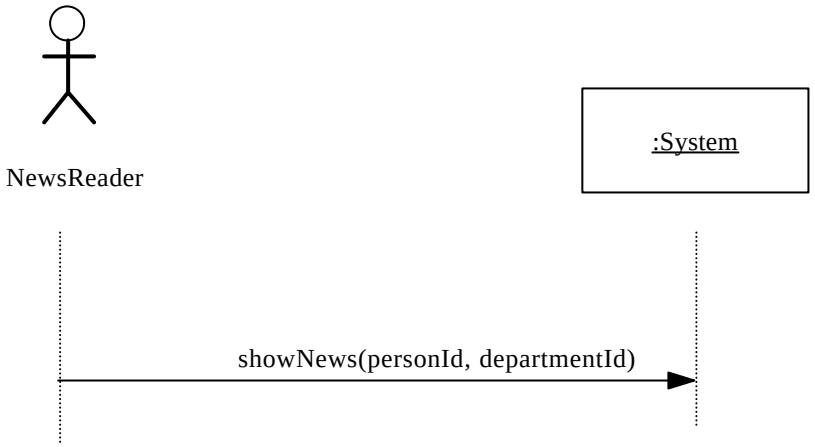


Utökad funktionalitet

Markera nyhet som läst



Se på nyheter från andra avdelningar



Systemoperationskontrakt

Log in

Namn:	logIn (login: String, password:String)
Ansvar:	Loggar in användaren i systemet och anropar automatiskt ShowNews(personId:int)
Typ:	System
Korsreferens:	Systemfunktioner:R1.2
Noter:	
Undantag:	Visar ett felmedelande om lösenordet och/eller loginId är felaktig.
Output:	-
Pre-conditions	Login-password är godkänt av systemet.
Post -conditions	-

View recent news

Namn:	showNews(personId:int, categoryId:int)
Ansvar:	Visa de nyheter som är relevanta för användren och den kategori han har valt..
Typ:	System
Korsreferens:	Systemfunktioner:R1.2
Noter:	-
Undantag:	-

Output:	-
Pre-conditions	-
Post -conditions	-
Create NewsItem	
Namn:	requestAddNewsItem(personId:int)
Ansvar:	Visa sida där användaren kan välja att relevanta data.
Typ:	System
Korsreferens:	Systemfunktioner:R1.1
Noter:	-
Undantag:	-
Output:	-
Pre-conditions	-
Post –conditions	-

Namn:	addNewItem(personId:int, heading:String, body:String, categoryId:int, classified:Boolean, endDate:Date, groupId:int[])
Ansvar:	Skapa en newsItem och lagra den
Typ:	System
Korsreferens:	Systemfunktioner:R1.1
Noter:	Undataget skall inte kunna inträffa om man inte hackar html-sidan
Undantag:	Visar ett felmedelande om body eller head innhåller för stor textmassa.
Output:	-
Pre-conditions	-
Post –conditions	En instans av News nws skapades. nws.heading = heading nws.body = body nws.privateItem = privateItem nws.endDate = endDate nws.startDate = den tidpunkten metoden anropas.

	nws.newsId = ett unikt id från en löpande sekvens. nwn associeras med de avdelningar vars departmentId har skickats med. nwn associerars med den person som har skapat nyheten.
Delete own NewsItem	
Delete any NewsItem	
Namn:	removeNewsItem(personId:int, newsItemId:int)
Ansvar:	Göra ett en nyhet inte mera kan ses från NEWS sidan.
Typ:	System
Korsreferens:	Systemfunktioner:R1.3
Noter:	Undataget skall inte kunna inträffa om man inte hackar html-sidan
Undantag:	Visar ett felmedelande om newsItemId är felaktig eller om användaren inte har behörighet att ta bort aktuellnyhet.
Output:	-
Pre-conditions	newsItemId existerar i systemet.
Post –conditions	News.visable = false

Systemoperationskontrakt för referensversionen

Add, remove or edit Person

Namn:	requestPersonEditing (editType: int {0,1,2})
Ansvar:	Presentera en dialog för att antingen lägga till, ta bort, eller ändra en person.
Typ:	System
Korsreferenser:	Systemfunktioner: R3.1, R3.2, R.3.3 Användardiagram: "Add, remove or edit Person"
Kommentar:	-
Undantag:	Om inte editeringstypen är en integer eller inte ligger inom intervallet så indikera felet.
Output:	-
Pre-conditions:	Editeringstypen är känt för systemet.
Post-conditions:	-

Namn:	listPersons
Ansvar:	Visa användaren vilka personer som finns i systemet.
Typ:	System
Korsreferenser:	Användardiagram: "Add, remove or edit Person"
Kommentar:	-
Undantag:	-
Output:	-
Pre-conditions:	-
Post-conditions:	-

Namn:	addPerson (firstName : String, lastName : String , eMail : String, departmentId : int)
Ansvar:	Lägga till en person med de specifierade uppgifterna till systemets

	databank.
Typ:	System
Korsreferenser:	Systemfunktioner: R3.1
	Användardiagram: "Add, remove or edit Person"
Kommentar:	-
Undantag:	Om inte departmentId refererar till en existerande avdelning, så indikera felet.
Output:	-
Pre-conditions:	Systemet måste känna till departmentId.
Post-conditions:	En *Person* skapades *(en instans skapas)*.
	Den nya *Personen* associerades med en *Department (en association skapas)*.
	Person.firstName sattes till firstName *(attributmodifiering)*.
	Person.lastName sattes till lastName *(attributmodifiering)*.
	Person.eMail sattes till eMail *(attributmodifiering)*.
	Person.personId sattes till ett unikt nummer från en löpande sekvens eMail *(attributmodifiering)*.

Namn:	removePerson
	(personId : int)
Ansvar:	Att ta bort en Person ur systemet, samt alla associationer till denna.
Typ:	System
Korsreferenser:	Systemfunktioner: R.3.3
	Användardiagram: "Add, remove or edit Person"
Kommentar:	-
Undantag:	Om inte personId refererar till en existerande Person, så indikera felet.
Output:	-
Pre-conditions:	Systemet måste känna till personId.
Post-conditions:	*Personen* togs bort *(instansborttagning)*.
	Personen avassocierades med *Department*.
	Personen avassocierades med *NewsItem*.

Namn:	editPerson

	(personId : int, firstName : String, lastName : String , eMail : String, departmentId : int)
Ansvar:	Att editera en Persons attribut.
Typ:	System
Korsreferenser:	Systemfunktioner: R.3.2 Användardiagram: "Add, remove or edit Person"
Kommentar:	-
Undantag:	Om inte personId refererar till en existerande Person, så indikera felet. Om inte departmentId refererar till en existerande avdelning, så indikera felet.
Output:	-
Pre-conditions:	Systemet måste känna till personId. Systemet måste känna till departmentId.
Post-conditions:	*Person* med personId associerades med en *Department (en association skapas).* *Person.firstName* sattes till firstName *(attributmodifiering).* *Person.lastName* sattes till lastName *(attributmodifiering).* *Person.eMail* sattes till eMail *(attributmodifiering).*

Namn:	requestDepartmentEditing (editType : integer)
Ansvar:	Att editera en Department:s attribut.
Typ:	System
Korsreferenser:	Systemfunktioner: R2.1, R2.2, R2.3 Användardiagram: "Add, remove or edit Departments"
Kommentar:	-
Undantag:	Om inte editType är inom definierat intervall eller är av integer-typ så indikera felet.
Output:	-
Pre-conditions:	Systemet måste känna till editType.
Post-conditions:	-

Add, remove or edit Departments

Namn:	addDepartment (departmentName : String)
Ansvar:	Lägga till en Department med de specifierade uppgifterna till systemets databank.
Typ:	System
Korsreferenser:	Systemfunktioner: R2.1 Användardiagram: "Add, remove or edit Department"
Kommentar:	-
Undantag:	Om departmentName existerar så indikera felet.
Output:	-
Pre-conditions:	-
Post-conditions:	En *Department dpt* skapades *(en instans skapas)*. *dpt.departmentName* sattes till *departmentName (attributmodifiering)*.

Namn:	listGroups
Ansvar:	Visa användaren vilka grupper/avdelningar som finns i systemet.
Typ:	System
Korsreferenser:	Användardiagram: "Add, remove or edit Department"
Kommentar:	-
Undantag:	-
Output:	-
Pre-conditions:	-
Post-conditions:	-

Namn:	removeDepartment (departmentId : int)
Ansvar:	Att ta bort en Department ur systemet, samt alla associationer till denna.
Typ:	System
Korsreferenser:	Systemfunktioner: R.2.3 Användardiagram: "Add, remove or edit Department"
Kommentar:	-
Undantag:	Om inte DepartmentId refererar till en existerande Department, så indikera felet. Om det existerar några referenser till Personer för den aktuella

	Department, så indikera felet (alla sådana associationer måste tas bort innan).
Output:	-
Pre-conditions:	Systemet måste känna till DepartmentId.
Post-conditions:	Om *Department* inte har några associationer till *Person*, samt om den inte har några associationer till *NewsItem*:s vilka inte har associationer till andra *Department*: *Department* togs bort *(instansborttagning).* *Departmenten* avassocierades med *NewsItem.* Om *Department* inte har några associationer till *Person*, samt om den **har** några associationer till *NewsItem* som inte har associationer till andra *Department*: *Department* togs bort *(instansborttagning).* *Departmenten* avassocierades med *NewsItem.* *NewsItem* togs bort *(instansborttagning).* Om *Department* har några associationer till *Person:* Otillåtet tillstånd, se undantag ovan.

Namn:	editDepartment (departmentId : int, departmentName : String)
Ansvar:	Att editera en Departments attribut.
Typ:	System
Korsreferenser:	Systemfunktioner: R.2.3 Användardiagram: "Add, remove or edit Department"
Kommentar:	-
Undantag:	Om inte departmentId refererar till en existerande Department, så indikera felet.
Output:	-
Pre-conditions:	Systemet måste känna till departmentId.
Post-conditions:	*Department.departmentName* sattes till departmentName *(attributmodifiering).*

Utökad funktionalitet

Mark NewsItem as read

Namn:	markAsRead (personId : int, newsItemId : int[])
Ansvar:	
Typ:	System
Korsreferenser:	Systemfunktioner: - Användardiagram: " Mark NewsItem as read"
Kommentar:	-
Undantag:	Om inte newsItemId refererar till en existerande NewsItem så indikera felet..
Output:	-
Pre-conditions:	Systemet måste känna till newsItemId.
Post-conditions:	*Person* med personId associerades med *NewsItem* med newsItemId *(en association skapas).*

View NewsItems from other Departments

Namn:	searchNews (personId : int, groupId : int[])
Ansvar:	
Typ:	System
Korsreferenser:	Systemfunktioner: - Användardiagram: "View NewsItems from other Departments"
Kommentar:	-
Undantag:	Om inte DepartmentId refererar till en existerande Department, så indikera felet.
Output:	-
Pre-conditions:	Systemet måste känna till departmentId.
Post-conditions:	-

View NewsItems from other Departments

Namn:	searchNews (personId : int, groupId : int[])

Ansvar:	
Typ:	System
Korsreferenser:	Systemfunktioner: - Användardiagram: "View NewsItems from other Departments"
Kommentar:	-
Undantag:	Om inte DepartmentId refererar till en existerande Department, så indikera felet.
Output:	-
Pre-conditions:	Systemet måste känna till departmentId.
Post-conditions:	-

Design

Verkliga användarmönsterspecifikationer

Grund

Logga in

Aktör: Nyhetsläsare

Syfte: Titta på de nyheter som är aktuella.

Beskrivning: Användaren loggar in och tittar på alla nyheter som är aktuella tidsmässigt och gruppmässigt.

Typ: expanderat verkligt.

Korsref: -

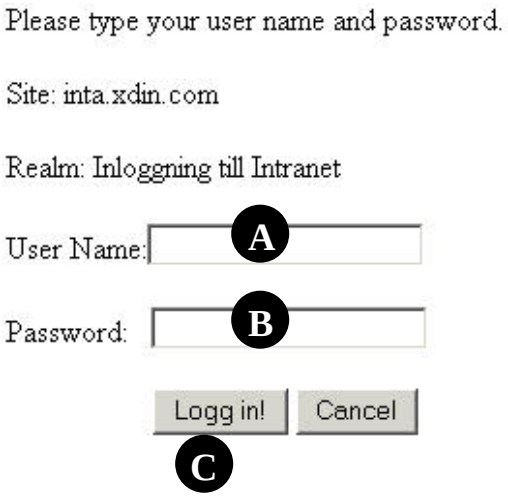


Bild 1. Fönster-1.

Typiskt händelseförlopp:

Aktörens *handling*	Systemets gensvar
1. Användaren skriver in sitt användarnamn i A lösen ord i B efter och trycker på C.	2. Alla nyheter som är aktuella tidsmässigt och för de *grupper* som användaren tillhör visas.. En länk till författarens "personprofilsida" visas för varje nyhet.

Titta på aktuella nyheter

Aktör: Nyhetsläsare

Syfte: Titta på de nyheter som är aktuella.

Beskrivning: Användaren tittar på alla nyheter som är aktuella tidsmässigt och gruppmässigt.

Typ: expanderat verkligt.

Korsref: -

Bild 2. Fönster-2.

Typiskt händelseförlopp:

Aktörens *handling*	Systemets gensvar
1. Användaren väljer A	2. Alla nyheter som är akutella tidmässigt och avdelsningsmässigt visas med den senast inlagda nyheten högst upp. En länk till författarens "personprofilsida" visas för varje nyhet.

Titta på privata nyheter (ev ta bort, eftersom ”privat” kommer att ligga i Shop-delen)

Aktör: Nyhetsläsare

Syfte: Titta på de privata nyheter som är aktuella.

Beskrivning: Användaren tittar på alla privata nyheter som är aktuella tidsmässigt och gruppmässigt.

Typ: expanderat verkligt.

Korsref:-

Bild 3. Fönster-3.

Typiskt händelseförlopp:

Aktörens handling	Systemets gensvar
1. Användaren väljer A.	2. Alla nyheter som är akutella tidmässigt och gruppmässigt visas med den senast inlagda nyheten högst upp. En länk till författarens "personprofilsida" visas för varje nyhet.

Skapa nyhet

Aktör: Nyhetsinläggare

Syfte: Lägga till en nyhet till systemet.

Beskrivning: Användaren skapar en nyhet som skall kunna ses av de specificerarde grupperna.

Typ: expanderat verkligt.

Korsref: -

Bild 4. Fönster-4.

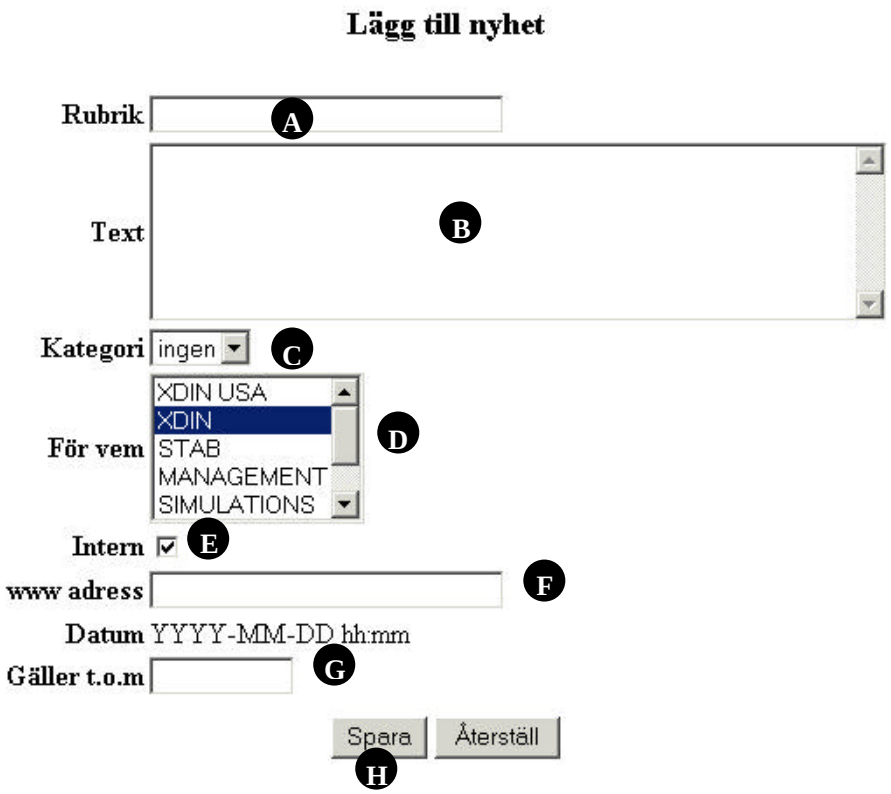


Bild 5. Fönster-5.

Typiskt händelseförlopp:

Aktörens handling	Systemets gensvar
1. Användren väljer A i fönster 4.	2.Systemet presenterar användaren de uppgifter som skall fyllas i. Default för A är ingen. Default för D är XDIN. Default för E är Intern. Default G är 10 dagar.
3. Användaren väljer/fyller i relventa uppgifter i A – G (F kan lämnas tom) i fönster 5.	
4. Användaren trycker på H	5 Systemet sparar valda alternativ
	6 Systemet presentera en bekräftelse

Alternativa händelseförlopp

Rad 4: Användaren väljer att avbryta inläggandet av nyhet

Rad 6 Om något alternativ inte kan kan presenteras ett felmedelande.

Ta bort nyhet

Aktör: Nyhetsinläggare

Syfte: Ta bort sin egen nyhet från systemet

Beskrivning: Användaren väljer en av sina egna nyheter och väljer att ta bort den.

Typ: expanderat verkligt.

Korsref: Användarscenario: Nyhetsinläggare måste utfört användarscenariet ”Titta på aktuella nyheter” eller ”Fritextsökning”

Bild 6. Fönster-6.

Typiskt händelseförlopp:

Aktörens handling Systemets gensvar

1. Användaren markerar en eller flera nyheter vid A (endast de egna nyheterna är markerbara)

2. Användaren trycker på B.

3. Systemet gör så att den nyheten inte lägre kan ses i NEWS. (Men den finns fortfarande lagrad)

4. Systemet presenterar en bekräftelse

Ta bort andras nyheter

Aktör: Administratör

Syfte: Administratör skall kunna ta bort stötande nyheter

Beskrivning: Administartören tar bort en nyhet från systemet

Typ: expanderat verkligt.

Korsref: Användarscenarie: Administratör måste utfört användarscenariet”Fritextsökning”.

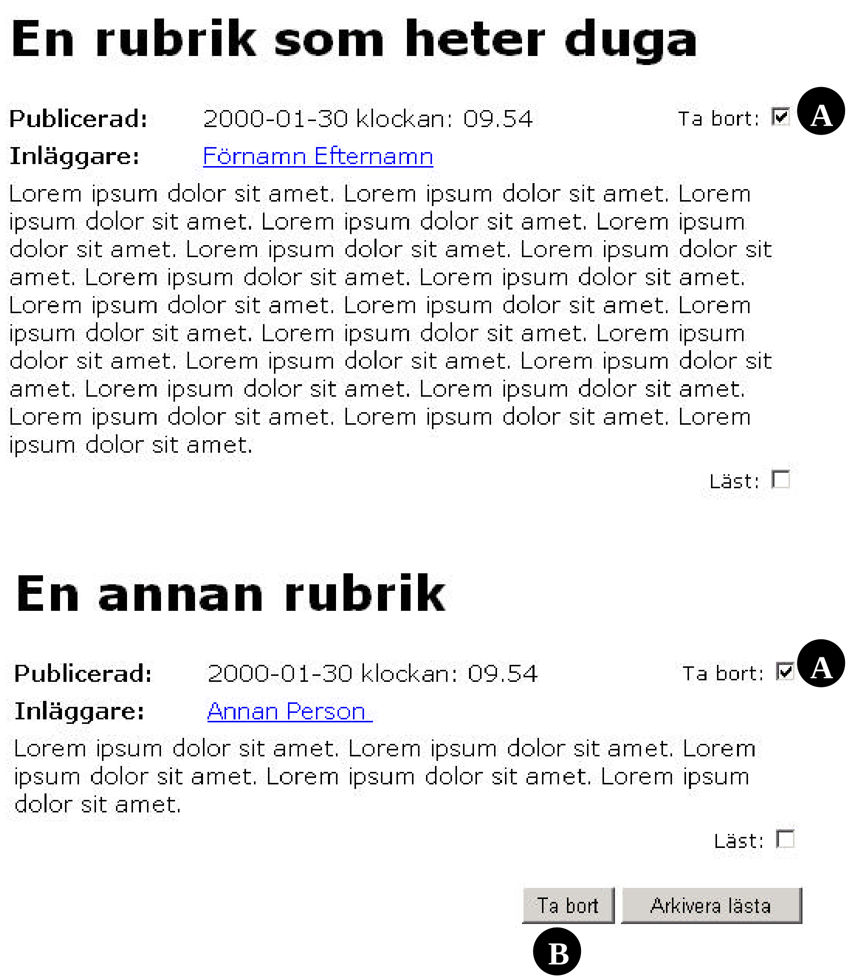


Bild 7. Fönster-7.

Typiskt händelseförlopp:

Aktörens handling	Systemets gensvar
1. Användaren markerar en eller flera nyheter vid A (alla nyheterna är markerbara)	
2. Användaren trycker på B	3. Systemet gör så att den nyheten inte lägre kan ses i NEWS. (Men den finns fortfarande lagrad)
	4. Systemet presenterar en bekräftelse

Lägg till/ta bort/ändra Person

Aktör: Administratör

Syfte: att lägga till eller ta bort en Person ur systemet. Att ändra en persons uppgifter, såsom namn, avdelningstillhörighet etc.

Beskrivning:

Administratören lägger in en ny person eller väljer att ta bort en existerande.

Administratören kan också ändra uppgifter för en person.

(Den här funktionaliteten finns redan i xdins intra. Vi kommer att använda den befintliga funktionaliteten när vi jobbar med EJB. Men när vi gör vår trådningsversion kommer vi att skapa funktionaliteten för det här användarsceneriet själva.).

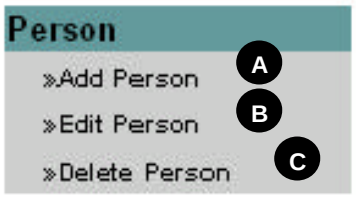


Bild 8. Fönster-8.

Typiskt händelseförlopp:

Sektion: Huvudsektion

Aktörens handling	Systemets gensvar
1. Detta händelseförlopp börjar när en Administratör väljer att administrera personalinformationen i systemet.	
2. Administratören väljer att lägga till, ta bort eller ändra en Person (A, B eller C i bild 8).	3. Om ”lägga till, se sektion ”lägga till”. Om ”ta bort”, se sektion ”ta bort”. Om ”ändra”, se sektion ”ändra”.

Alternativa händelseförlopp: -

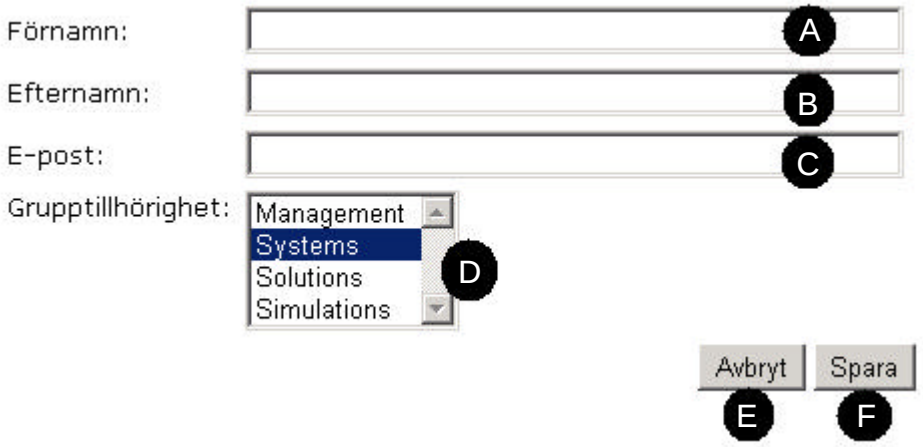


Bild 9 Fönster-9.

Sektion: Lägga till

Aktörens handling	Systemets gensvar
1. Administratören har tryckt på A i fönster 8.	
2. Administratören fyller namn och e-post för den nya Personen (A-C i fönster 9). Som Avdelning kan endast en eller flera existerande Avdelningar (Grupper) väljas.	3. I B i fönster y visas de aktuella grupper som finns. Flera grupper kan väljas.
4. Administratören trycker på spara (F).	4. Systemet presenterar Administratören med information om det gick bra att lägga till Personen.

Alternativa händelseförlopp:

Rad 4. Administratören väljer att inte lägga till en Person utan trycker på E. Scenariot avbryts.

Bild 10. Fönster-10.

Sektion: Ta bort

Aktörens handling	Systemets gensvar
1. Administratören har tryckt på B i fönster 8.	
2. Administratören väljer en eller flera Personer att ta bort (A i Fönster 10).	
3. Administratören trycker på B för att ta bort de markerade.	4. Om inga personer är markerade och administratören trycker på B så presenteras han med ett meddelande om detta. Om några är markerade så tar systemet bort dessa och meddelar administratören om resultatet.

Alternativa händelseförlopp:

- Rad 3. Administratören ångrar sig och tar inte bort, trycker istället på C. Scenariot avbryts.

Bild 11a, fönster 11a

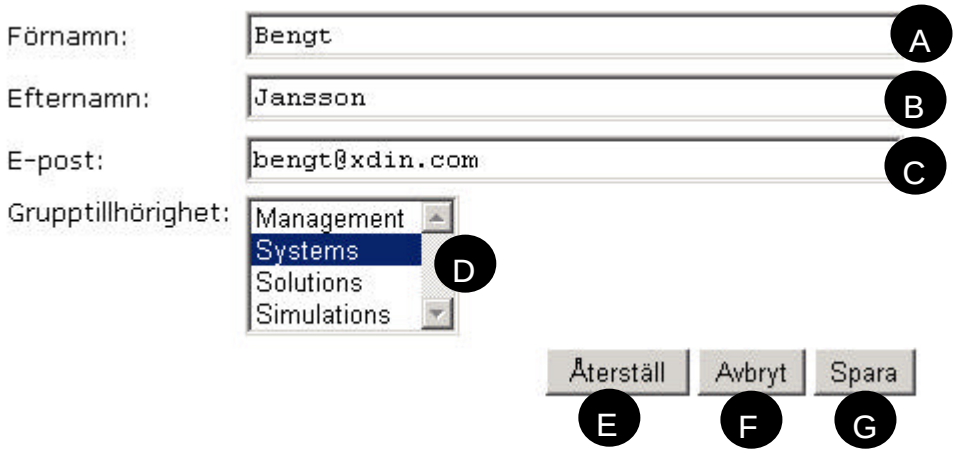


Bild 11b. Fönster-11b.

Sektion: Ändra

Aktörens handling	Systemets gensvar
1. Administratören har tryckt på C i fönster 8.	
	2. Systemet presenterar Administratören med en listning av de Personer som finns i systemet i form av fönster n.
3. Administratören väljer att editera en persons uppgifter genom att trycka på en av Edit-knapparna (A i fönster 11a).	
4. Administratören ändrar de aktuella uppgifterna (A-D i fönster 11b).	
5. Administratören trycker på Spara (G).	6. Systemet presenterar Administratören med en bekräftelse på att de nya

uppgifterna har införts.

Alternativa händelseförlopp:

- Rad 5. Administratören väljer att återställa uppgifterna och trycker på E. Scenariot börjar på nytt vid rad 4.
- Rad 5. Administratören väljer att avbryta ändringen och trycker på F. Scenariot avslutas.

Lägg till/ta bort/ändra Grupp (avdelning)

Aktör: Administratör

Syfte: Att administrera de avdelningar och grupper som finns i systemet.

Beskrivning:

Administratören lägger in en ny avdelning/grupp eller väljer att ta bort en existerande.

Administratören kan också ändra uppgifter för en avd/grupp.

(Den här funktionaliteten finns redan i xdins intra. Vi kommer att använda den befintliga funktionaliteten när vi jobbar med EJB. Men när vi gör vår trådningsversion kommer vi att skapa funktionaliteten för det här användarsceneriet själva.)

Typiskt händelseförlopp:

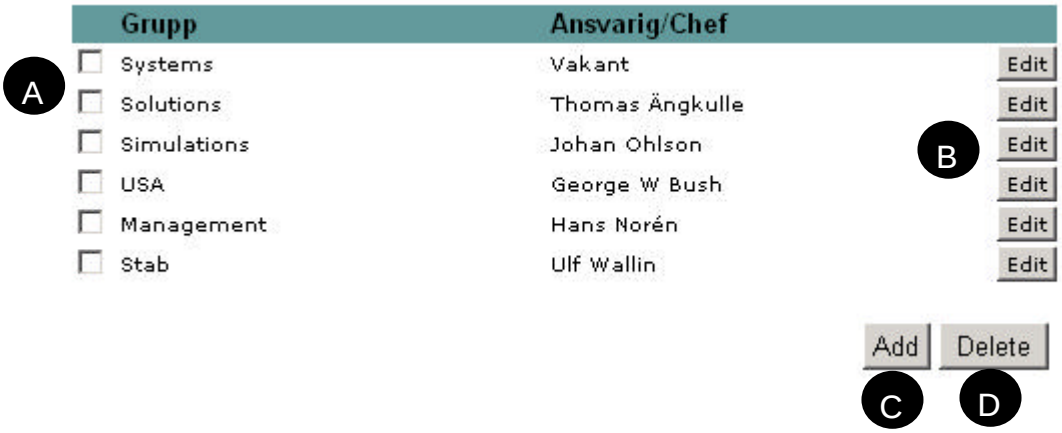


Bild 12. Fönster-12.

Sektion: Huvudsektion

Aktörens handling	Systemets gensvar
1. Detta händelseförlopp börjar när en Administratör väljer att administrera avdelningar/grupper (hädanefter kallat	

Grupp) i systemet.

2. Administratören väljer i fönster 12 att lägga till(C), ta bort(A och D) eller ändra (B) en Grupp.	3. Om ”lägga till, se sektion ”lägga till”. Om ”ta bort”, se sektion ”ta bort”. Om ”ändra”, se sektion ”ändra”.

Alternativa händelseförlopp: -

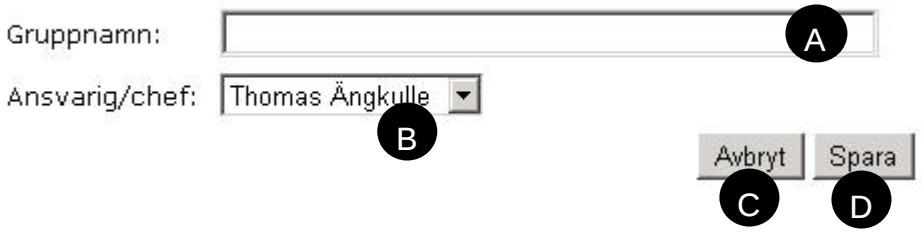


Bild 13. Fönster-13.

Sektion: Lägga till

Aktörens handling	Systemets gensvar
1. Administratören har tryckt på C i fönster 12**.**	
2. Administratören fyller i gruppens namn (A), samt vem som är ansvarig för gruppen (B) i fönster 13.	3. Värdena i B är ifyllda med alla personer som finns i systemet.
4. Administratören lägger till gruppen genom att trycka på spara (D).	5. Systemet presenterar Administratören med information om det gick bra att lägga till Gruppen.

Alternativa händelseförlopp:

- Rad 4. Administratören väljer att inte lägga till en Grupp och trycker på avbryt (C). Scenariot avbryts.

Sektion: Ta bort

Aktörens handling	Systemets gensvar
1. Börjar när administratören har valt att ta bort en Grupp ur systemet. Han har då markerat en eller flera grupper (A i fönster 12) och tryckt på D.	2. Systemet visar om det gick bra att ta bort Grupperna. Om ingen grupp valts så meddelas administratören om detta.

Alternativa händelseförlopp:

- Rad 1. Administratören ångrar sig och tar inte bort. Scenariot avbryts.

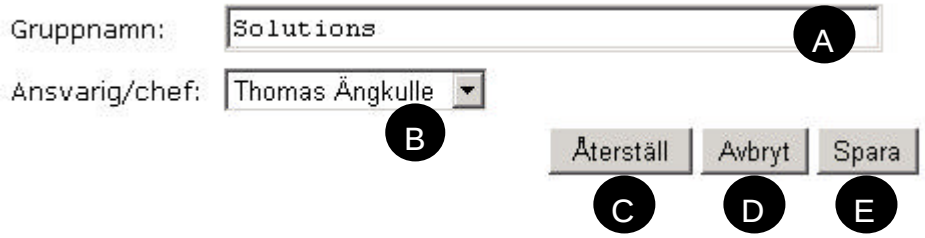


Bild 14. Fönster-14.

Sektion: Ändra

Aktörens handling	Systemets gensvar
1. Administratören har tryckt på B i fönster 12**.**	
2. Administratören ändrar de uppgifter han vill (A och B i fönster 14).	
3. Administratören sparar ändringarna genom att trycka på spara (E).	6. Systemet presenterar Administratören med en bekräftelse på att de nya uppgifterna har införts.

Alternativa händelseförlopp:

- Rad 3. Administratören väljer att återställa uppgifterna. Scenariot börjar på nytt vid rad 2.
- Rad 3. Administratören väljer att avbryta ändringen. Scenariot avslutas.

Utökad funktionalitet

Markera Nyhet som läst

Aktör: Nyhetsläsare

Syfte: Att markera en Nyhet som läst för att inte behöva se på den igen, förutom i Arkivet.

Beskrivning:

Användaren väljer en eller flera nyheter och väljer att markera dem som lästa. De kommer då inte upp vid nästa nyhetslistning.

Typiskt händelseförlopp:

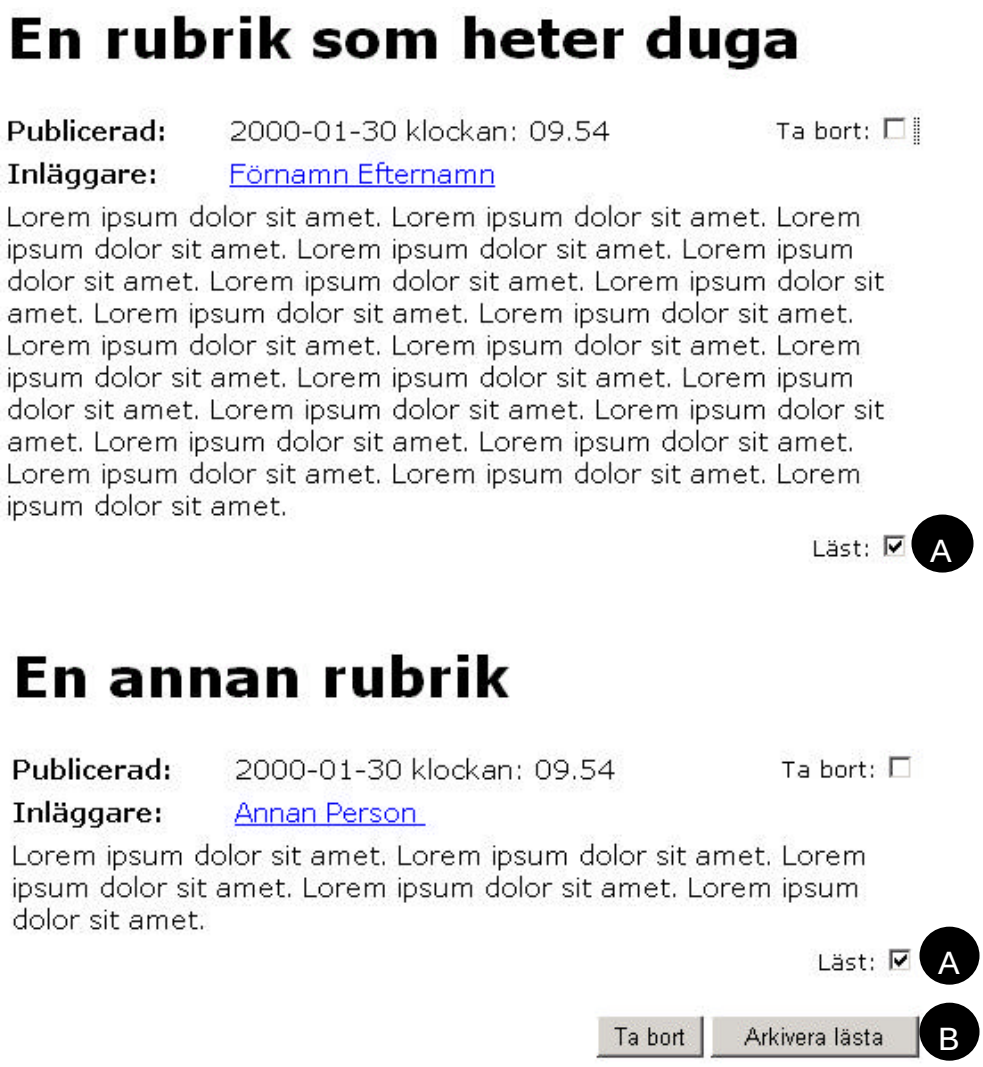


Bild 15. Fönster-15.

Sektion: Huvudsektion

Aktörens handling	Systemets gensvar
1. Detta händelseförlopp börjar när en Nyhetsläsare har valt att se på en nyhet eller då han ser på en nyhetslistning med rubriker.	
2. Användaren väljer att markera nyheten som läst (A i fönster 15).	
3. Användaren trycker på "Arkivera lästa" (B).	3. Systemet visar inte nyheten nästa gång användaren listar Nyheter.

Titta på andra gruppers/avdelningars nyheter

Aktör: Nyhetsläsare

Syfte: Att en Person skall kunna läsa Nyheter från andra Grupper än den han själv tillhör.

Beskrivning:

Användaren väljer att se på nyheter från en eller flera grupper. De som är aktuella tidsmässigt visas och behörighetsmässigt (markerade som icke *icke klassificerad* när nyheten skapades) visas.

Typiskt händelseförlopp:

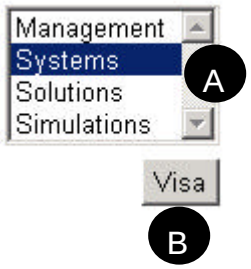


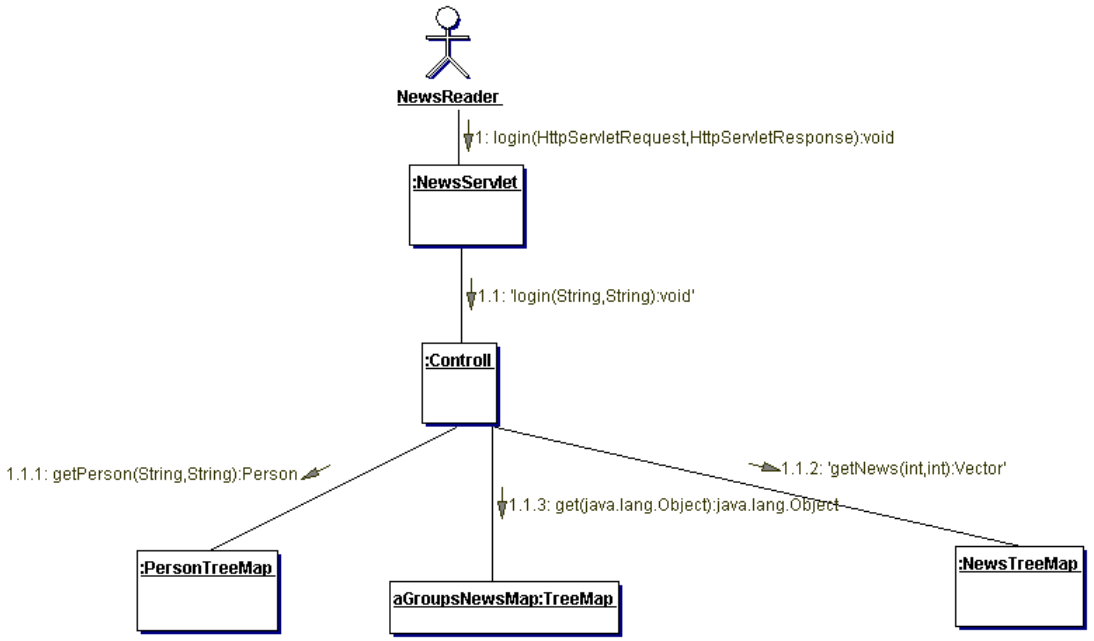
Bild 16. Fönster-16.

Sektion: Huvudsektion

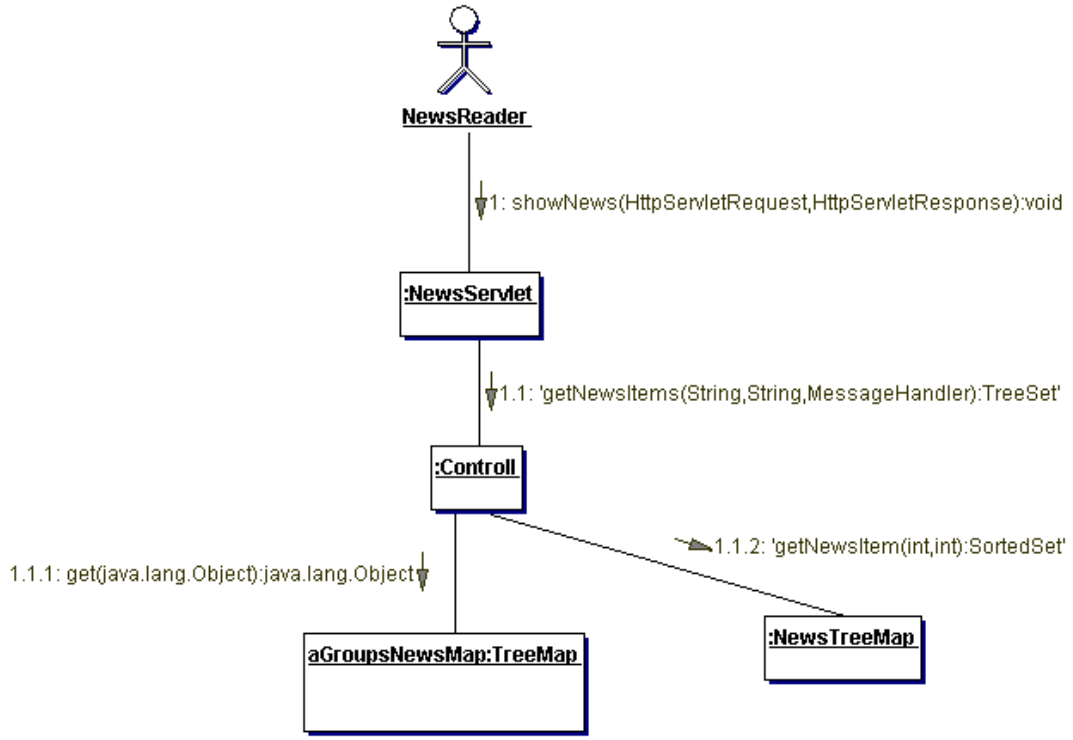
Aktörens handling	Systemets gensvar
1. Detta händelseförlopp börjar när en Nyhetsläsare har valt att titta på nyheter som är avsedda för andra Grupper än den han själv tillhör.	2. I A i fönster 16 visas de aktuella avdelningarna.
3. Nyhetsläsaren väljer den/de Grupp(er) vars nyheter han vill titta på i A i fönster 16 och trycker därefter på visa (B).	4. Systemet presenterar en listning av nyheter från den valda gruppen. Endast de nyheter som är markerade som *"ej avdelningsintern"* visas.

Cykel 1: Collaboration Diagrams

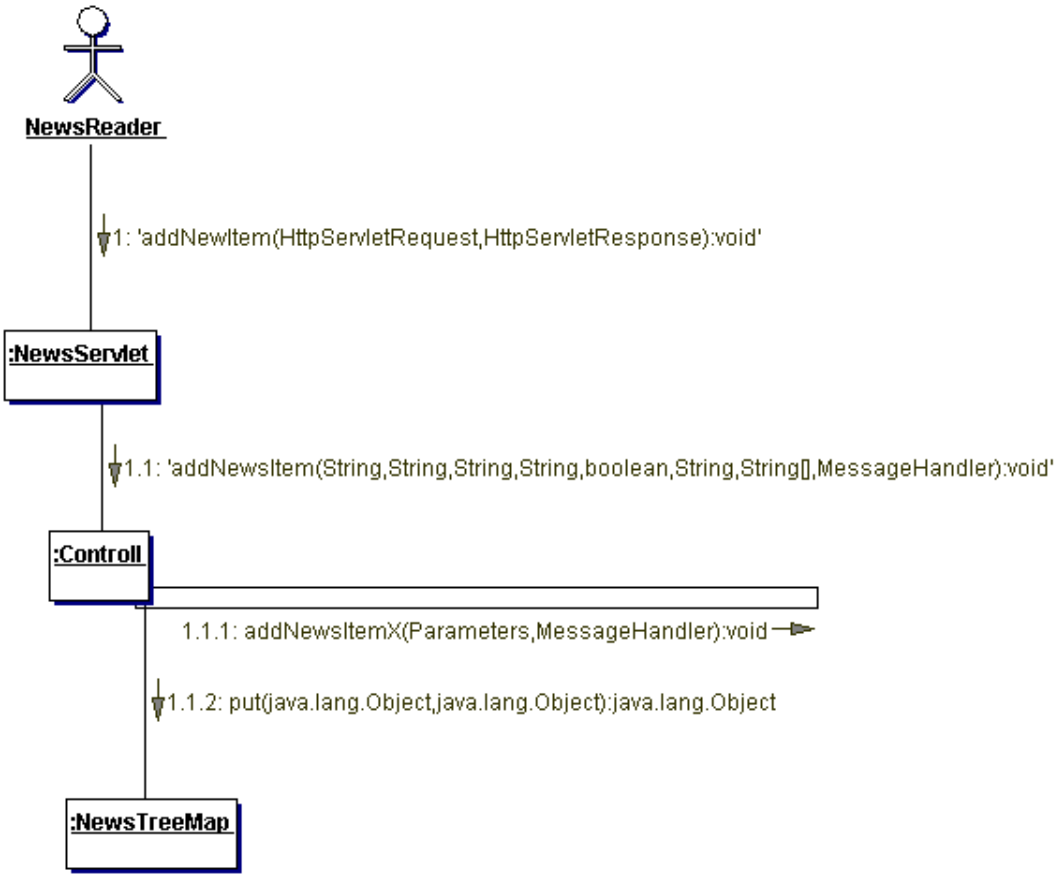
Logga in



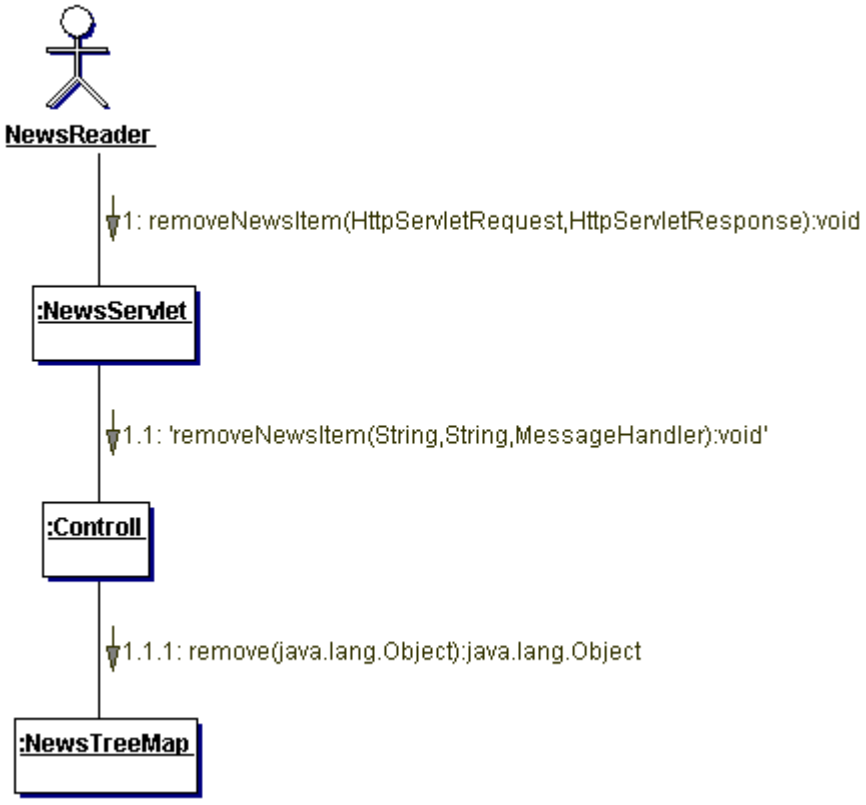
Titta på aktuella nyheter / Titta på privata nyheter



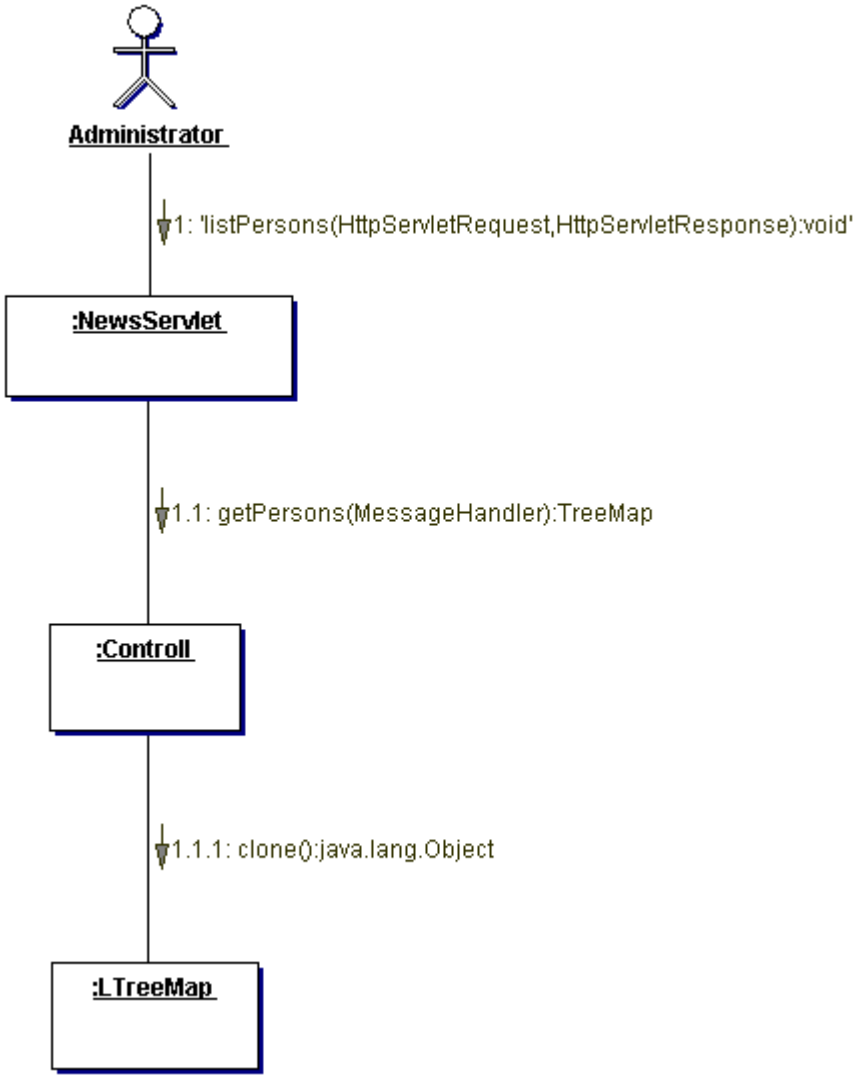
Skapa nyhet



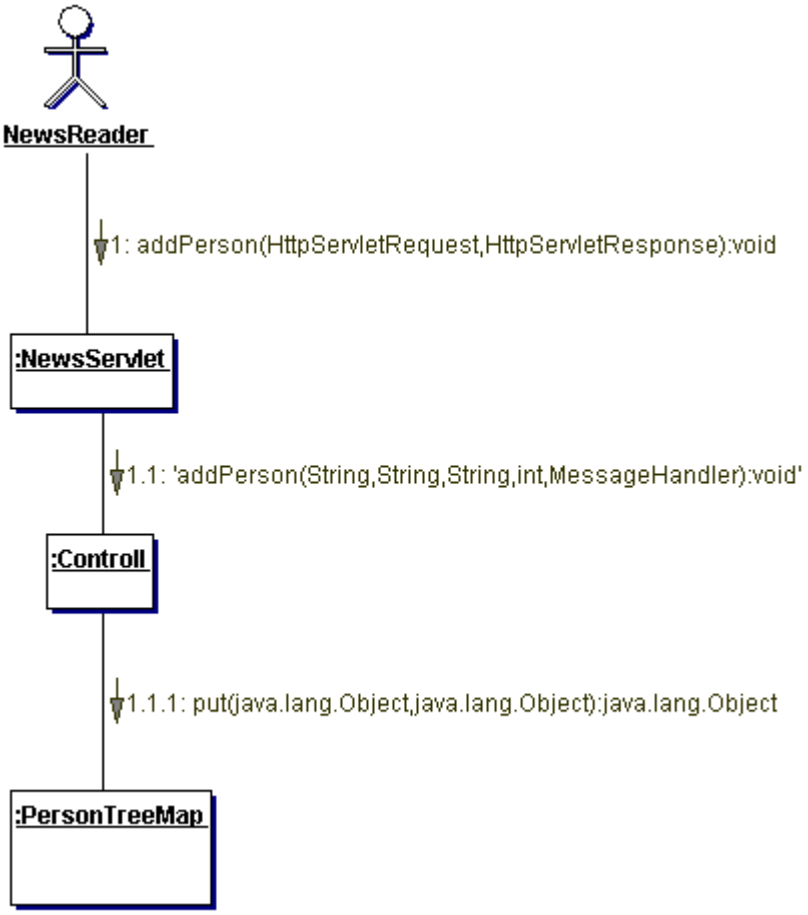
Ta bort nyhet



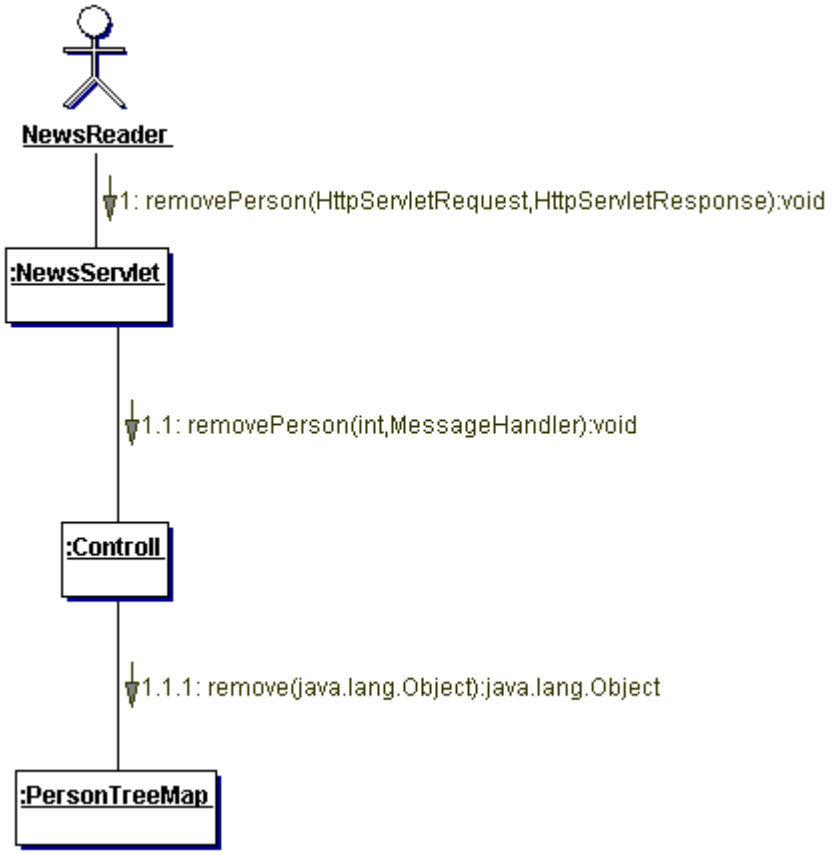
Lista Personer



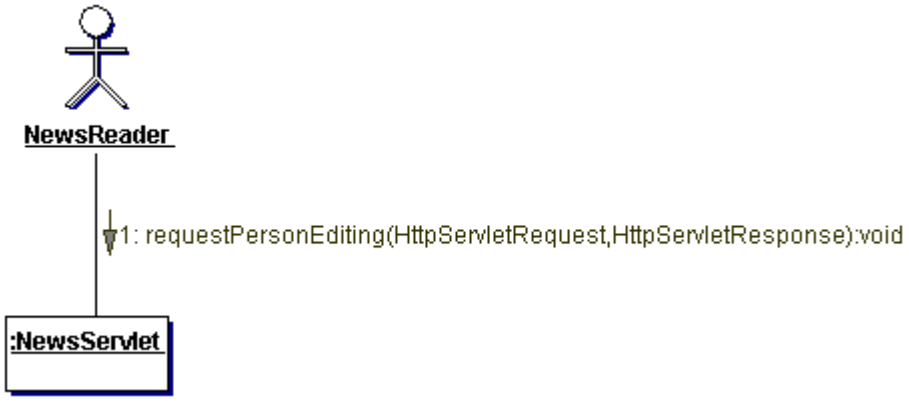
Lägga till Person

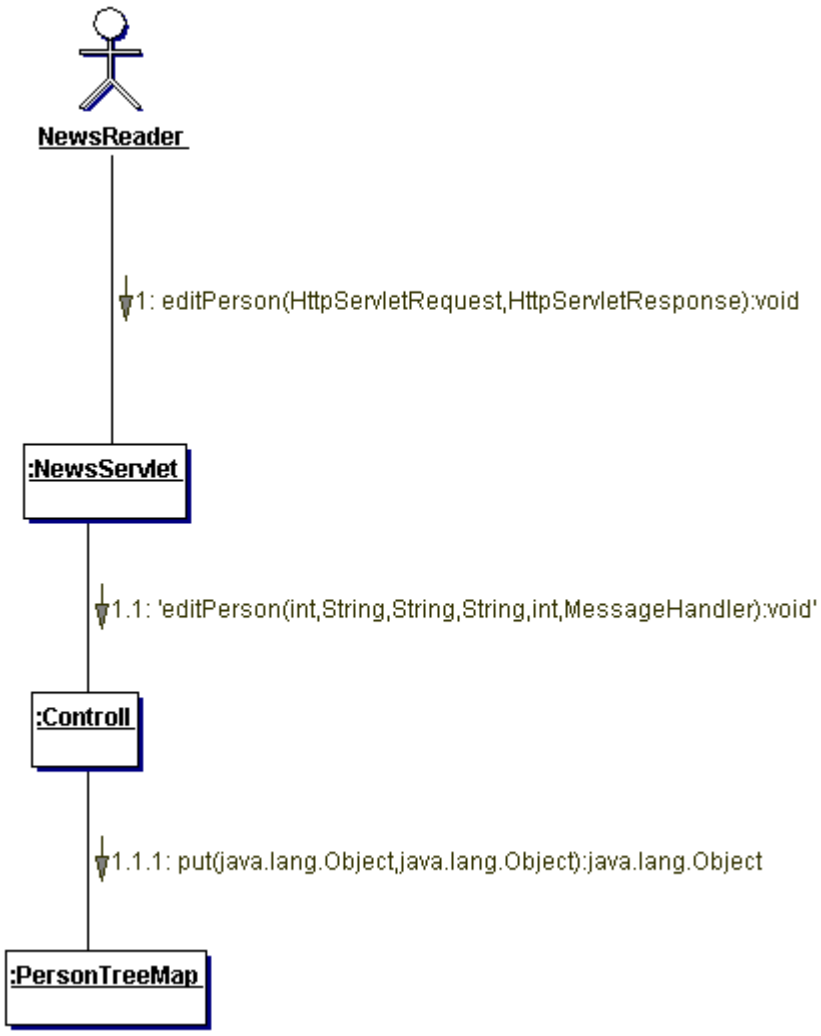


Ta bort Person

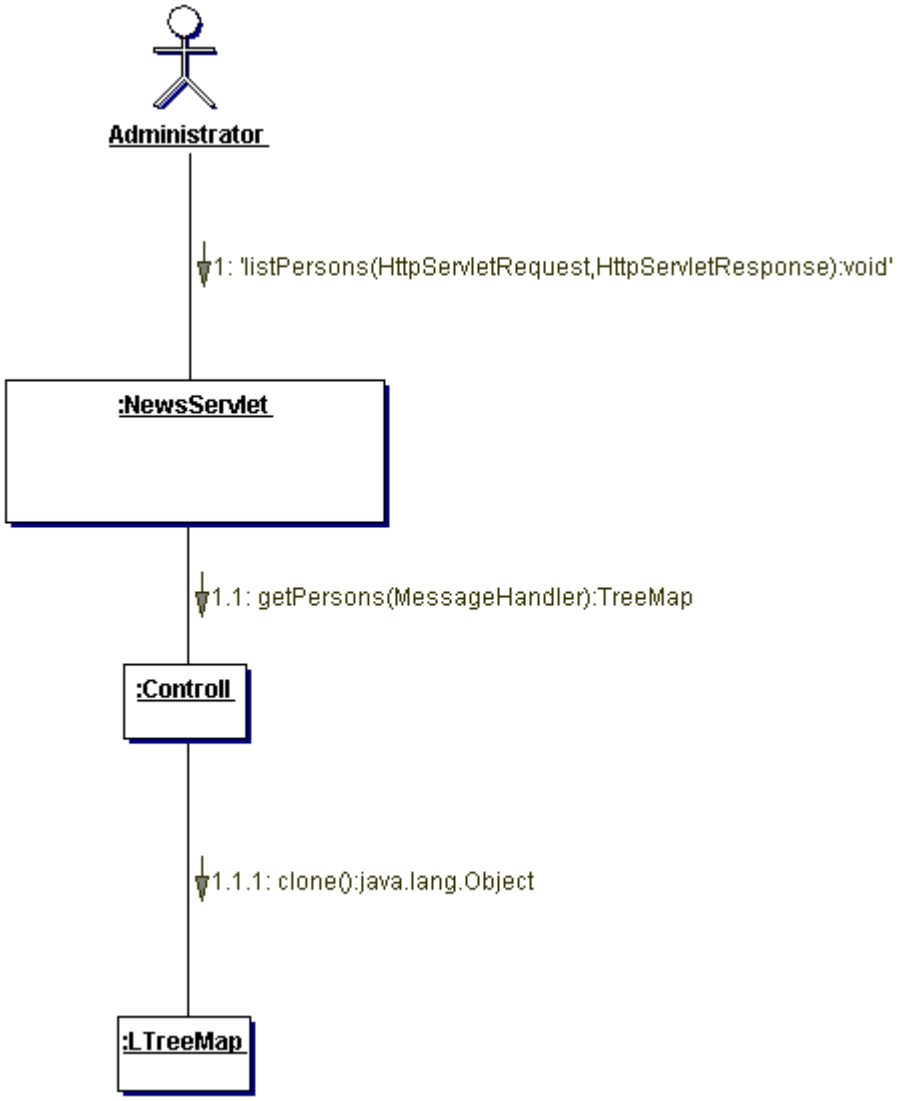


Ändra Person

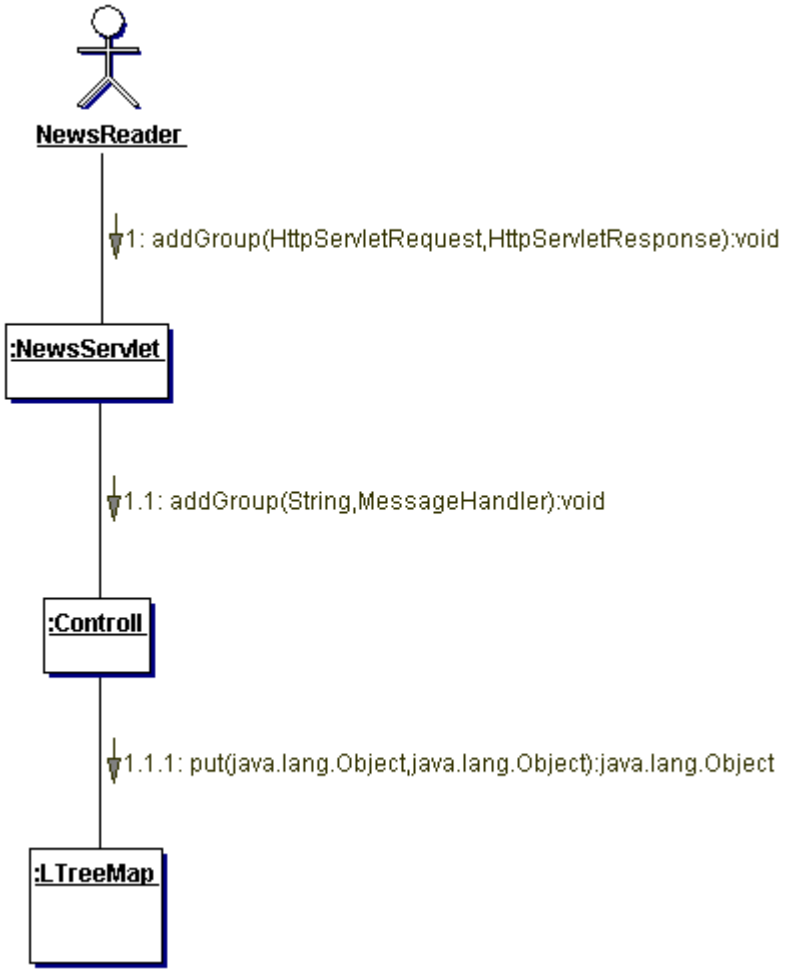



NewsReader
1: editPerson(HttpServletRequest,HttpServletResponse):void
:NewsServlet
1.1: 'editPerson(int,String,String,String,int,MessageHandler):void'
:Controll
1.1.1: put(java.lang.Object,java.lang.Object):java.lang.Object
:PersonTreeMap

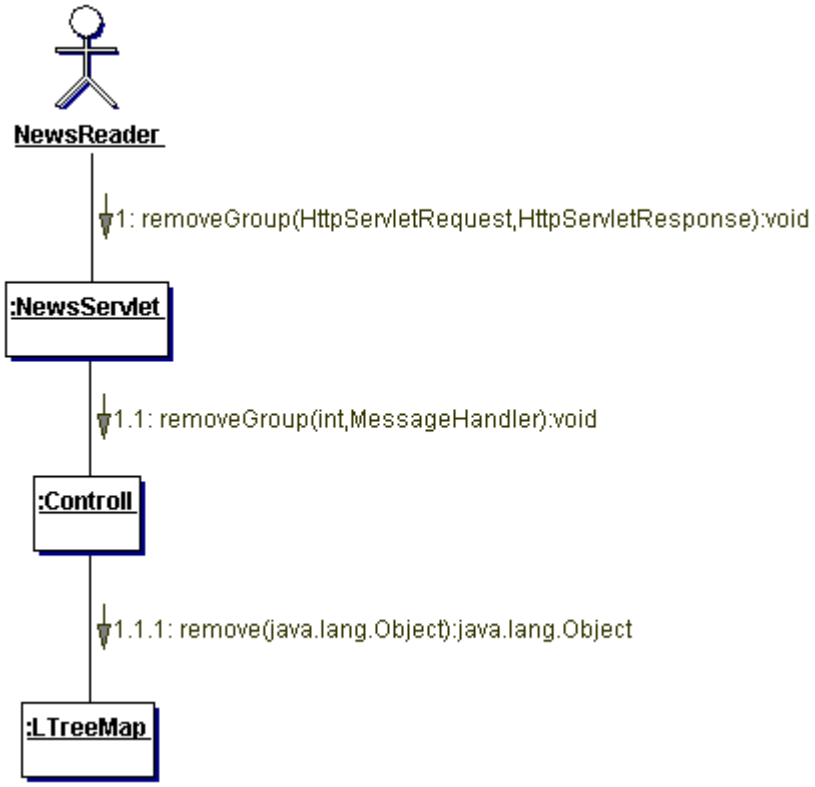
Lista Grupper



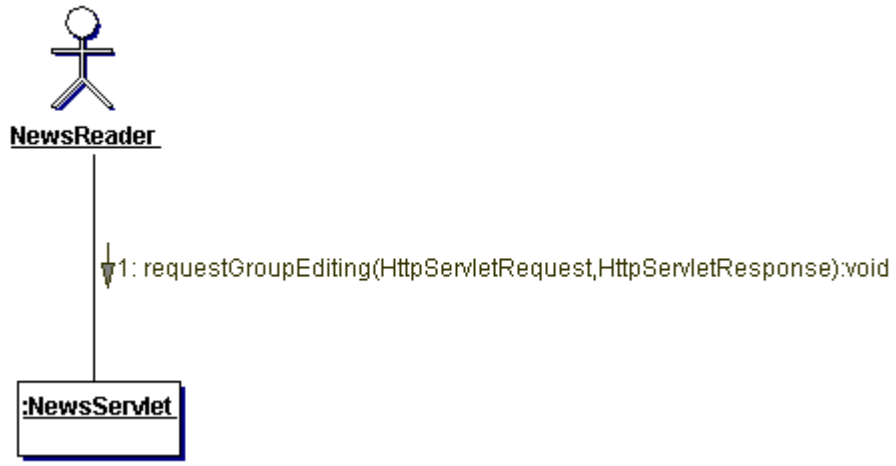
Lägga till Grupp

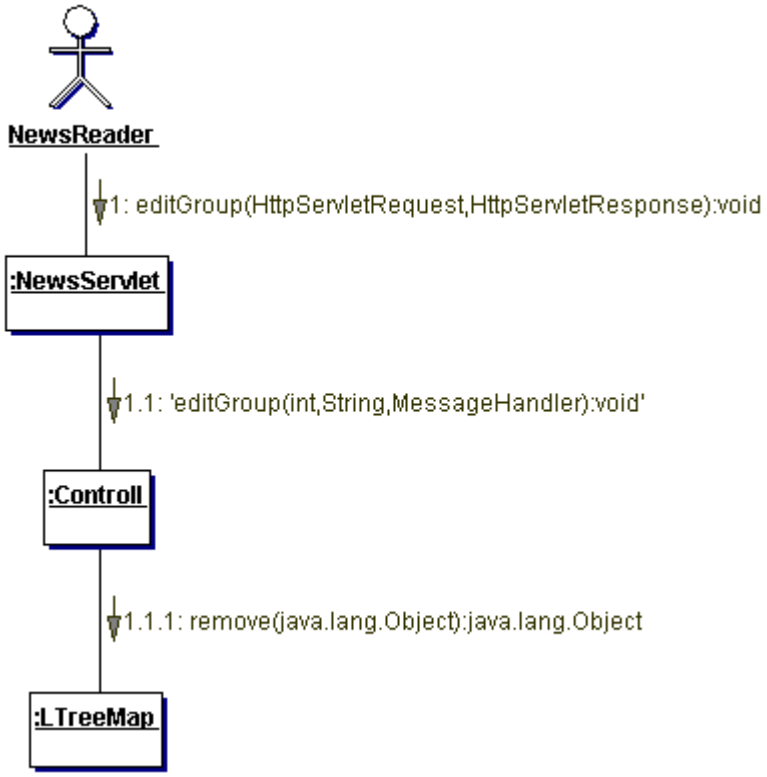


Ta bort Grupp

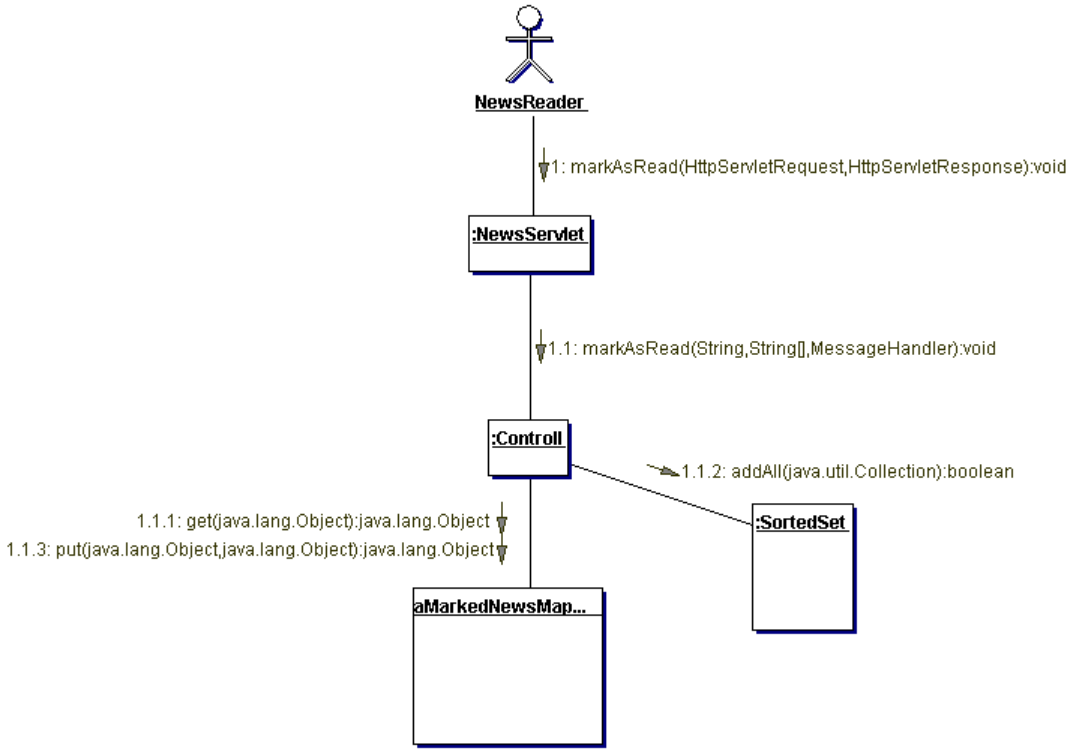


Ändra Grupp

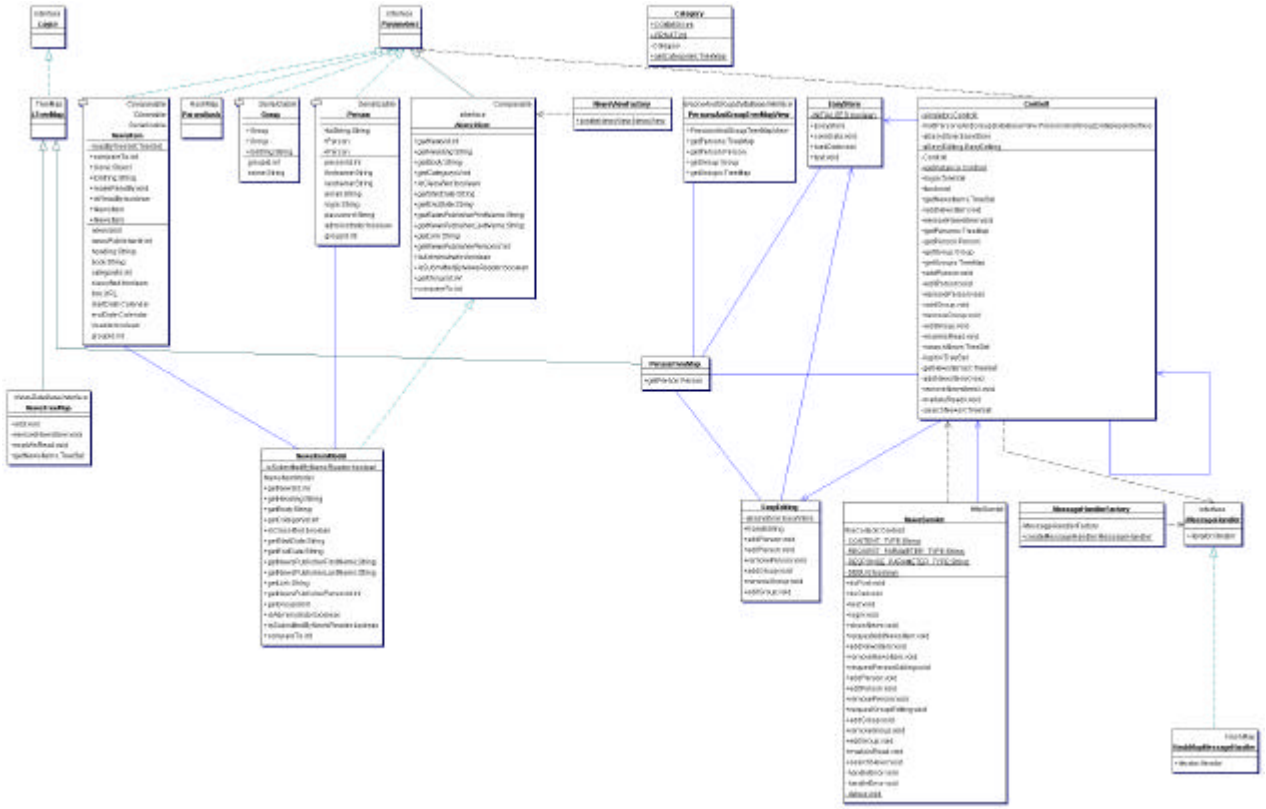




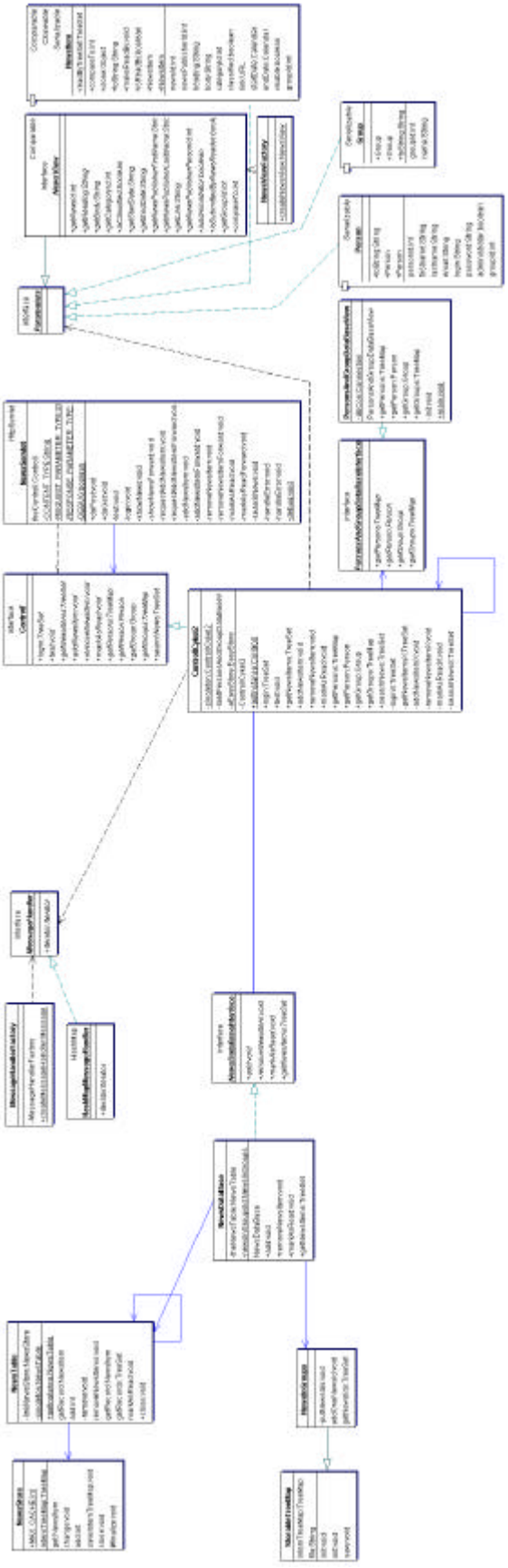
Markera Nyhet som läst



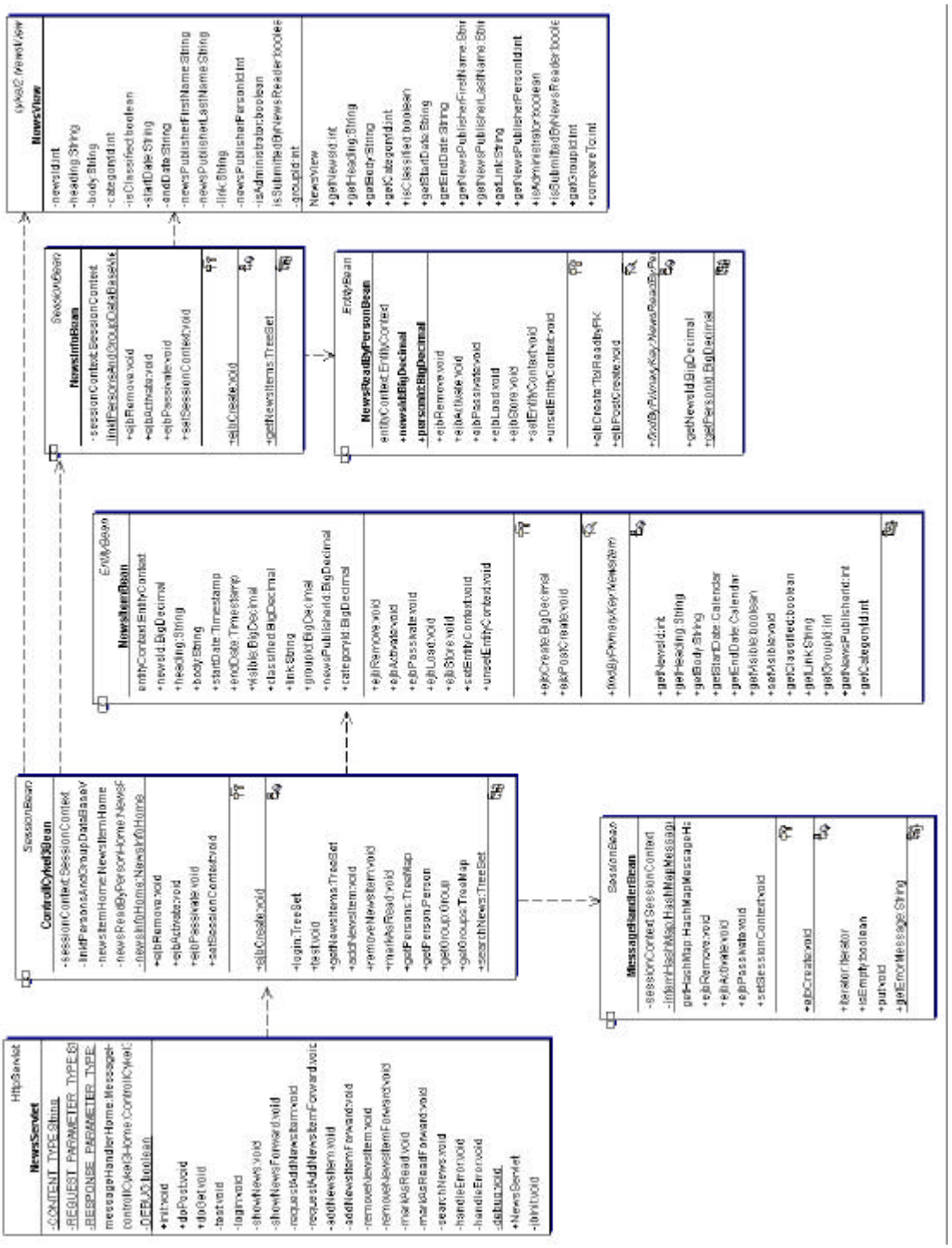
8.4.1 Klassdiagram, cykel 1



8.4.2 Klassdiagram, cykel 2



8.4.3 Klassdiagram, cykel 3



8.5 Bilaga 5 Intervjufrågor och intervjuresultat

Intervjuerna kan nås via internet på följande URL:

http://www.skip.informatik.gu.se/~j/ejb/intervjuer/

Intervjuomgång 1

- Vad heter du?
- Hur länge har du arbetat med programmering?
- Vilka programmeringsspråk kan du?
- Hur länge har du arbetet med Javaprogrammering?
- (Hur länge har du arbetat med C++-programmering?)
- Hur länge har du arbetat med trådade Java-program?
- Hur länge har du arbetat med Servlets/JSP ?
- Hur stor del av tiden i kodningen använder du till att tråda applikationen när du skriver serverprogram med Servlets/JSP?
- Hur väger du kraven på snabb utvecklingstid respektive snabbast möjliga prestanda på den färdig produkten?
- Hur mycket känner du till om Enterprise JavaBeans?
- När du gör en utvärdering av Enterprise JavaBeans - jämfört med att hantera samtidighet etc. i applikationerna själv - vilka kriterier äv viktiga för dig?
- Vilka kriterier vid en utvärdering av Enterprise JavaBeans är viktiga när det gäller utvecklingsstöd i form av API:er etc. (dvs. programmeringsmiljö m.m.)?
- Vilka förväntningar har du på EJB?
- Vilka av dessa rör språkstöd i form av API:er etc.?

Intervjuomgång 2

- Vad heter du?
- Hur länge har du arbetat med programmering?
- Vilka programmeringsspråk kan du?
- Hur länge har du arbetet med Javaprogrammering?
- (Hur länge har du arbetat med C++-programmering?)
- Hur länge har du arbetat med trådade Java-program?
- Hur länge har du arbetat med Servlets/JSP ?
- Hur stor del av tiden i kodningen använder du till att tråda applikationen när du skriver serverprogram med Servlets/JSP?
- Hur väger du kraven på snabb utvecklingstid respektive snabbast möjliga prestanda på den färdig produkten?
- Hur länge har du arbetat med EJB?

- Vilka typer av lösningar har du använt EJB för?
- Vilka var dina kunskaper om och förväntningar på EJB innan du hade jobbat med det?
- Vad anser du vara EJB:s största styrka respektive svaghet?
- Hur bra prestanda anser du att EJB levererar till skillnad från andra (egenutvecklade) lösningar?
- Vilka typer av projekt typer du lämpar sig för att implementeras med hjälp av EJB?
- Hur ser mjukvaruutvecklingsprocessen ut för dig/er när ni utvecklar med EJB?
- Ser du några vinster i form av tid eller annat under mjukvaruutvecklingsprocessen när ni använder er av EJB?
- Upplevde du EJB som ett lättanvänt ramverk för utveckling av flerskiktade system?
- Tycker du att det finns några alternativ till EJB som ramverk för att utveckla distribuerade flerskiktade system? Har du i så fall någon erfarenhet av detta eller dessa?
- Hur tycker du att EJB hanterar komplexa datastrukturer?
- Är det lättare att skriva återanvändbar kod med EJB? Har du återanvänt bönor?

8.6 Bilaga 6 FAQ-frågor

Frågorna som redovisas ned var de som fanns att finna på JGurus EJB-FAQ den tionde maj 2001.

Frågor om EJB specen, samt andra grundläggande frågor om EJB

Frågor om olika versioner av EJB-specifikationen

- When was EJB 1.1 defined?
- How is Version 1.1 better than Version 1.0?
- What's different in Enterprise JavaBeans 1.1?
- Does the new specification impact forward compatibility?
- When was EJB 1.0 defined?
- Is Enterprise JavaBeans 1.0 still viable?
- What's new in the EJB 2.0 specification?
- What are the deprecated EJB 1.0 conventions with respect to EJB 1.1?
- What's the main difference between throwing an javax.ejb.EJBException and throwing an java.rmi.RemoteException from within a EJB? When should each be thrown?
- What is the correct DTD URL (DOCTYPE) for use with a messaging bean?

Övergripande eller grundläggande frågor om EJB

- What is Enterprise JavaBeans?
- What's so special about Enterprise JavaBeans?
- Are Enterprise JavaBeans and JavaBeans the same thing?
- What is a container?
- Why are beans not allowed to create their own threads?
- Is method overloading allowed in EJB?
- What is the difference between a Server, a Container, and a Connector?
- What restrictions are imposed on an EJB? That is, what can't an EJB do?
- Where can I find the API documentation for Enterprise JavaBeans (EJB)?
- Is it possible to use EJB in architecting a multi-user collaborative application? If so, how might it work, and what are some pitfalls to avoid?
- Are enterprise beans allowed to use Thread.sleep()?
- In reference to EJB, what is the meaning of scalable?, how does EJB support scalability?
- What is an enterprise bean?

- How do I introspect a bean at run time to discover its type(s) and available methods?
- What makes a Java class an enterprise bean?
- Does the EJB programming model support inheritance?
- Should synchronization primitives be used on bean methods?
- Can I use Threads in a enterprise bean?
- How does EJB support polymorphism?
- Can a bean act as a singleton to provide logging and other services?
- What's the reason for having two interfaces -- EJBHome for creating, finding & removing and EJBObject for implementing business methods. Why not have an single interface which supports both areas of functionality?
- How should singleton EJBs be implemented in a cluster ?
- What is an EJB Context?
- Why do we have a remove method in both EJBHome and EJBObject?
- Can the bean class which implements either javax.ejb.EntityBean or javax.ejb.SessionBean be inherited from any other class?
- What is the difference between session and entity beans? When should I use one or the other?
- Is it possible to derive my EJB class from a standard Java class?
- Is it legal to have static initializer blocks in EJB?
- Is it possible to write two EJB's that share the same Remote and Home interfaces, and have different bean classes? if so, what are the advantages/disadvantages?
- Can an EJB Remote interface extend other interfaces as well as javax.ejb.EJBObject?
- Why am I able to call a business method using the same Remote Interface reference after I called the remove() method of a Stateless Session Bean??
- What is a session bean?
- What is a stateful session bean?
- How does passivation work in stateful session beans?
- What is a stateless session bean?
- How can one have non-client specific state in stateless session beans?
- What are the benefits of using a Stateless Session Bean over using a class purely consisting of static methods?

Tjänster

Managed persistence

CMP

- What is a CMP bean?
- While deploying CMP entity beans, which fields in the bean are container-managed and how are they identified?
- How to manage fields that can have null values in a container-managed Entity bean?
- Is there a difference between container managed and bean managed persistence in terms of performance?
- How is persistence implemented in enterprise beans?
- How do I map a Date/Time field to an Oracle database with CMP?
- My client's connection times out when trying to retrieve a huge number of records (1000+) from an EJB server that uses CMP to a database. How can I resolve this?
- If there is a method in an Entity EJB that does not modify the bean's data (read only), can I specify that load() and store() methods should not be called before and after the call to this method?
- My database has a column of type DATETIME. What type of Java variable do I use in the EJB?
- Should I use CMP or BMP for an application with complex data manipulation & relations?
- What is EJB QL?
- How can I get the underlying JDBC connection which the CMP bean is using?

BMP

- What is a BMP bean?
- Please show me an example of an Entity Bean that accesses more than one database table, using BMP.
- I am developing a BMP Entity bean. I have noticed that whenever I the create method is invoked, the ejbLoad() and the ejbStore() methods are also invoked. I feel that once my database insert is done, having to do a select and update SQL queries is major overhead. is this behavior typical of all EJB containers? Is there any way to suppress these invocations?
- For BMP, I don't want all the variables to be updated in the database when only one variable has been changed in each of the business methods. Is it a good practice to load and store values from and to database inside the business methods rather than ejbStore() and ejbLoad()?

Primary Keys

- What is an EJB primary key? How is it implemented when the database doesn't have a primary key?
- Can primary keys contain more than one field?
- How do I automatically generate primary keys?

- Can the primary key in the entity bean be a Java primitive type such as int?
- When using Primary Keys, why do I have to implement the hashCode() and equals() method in my bean?
- Why is ejbFindByPrimaryKey mandatory?
- I've read that the Primary Key field can be a wrapper class. But, nowhere has it been mentioned if it can be any Java API class. To be specific, can java.util.Vector be a Primary Key class for my bean?
- What happens when a client calls an entity bean's home interface create() method and an entity bean already exists with that primary key, created previously by another client ? Also what happens when a client calls an entity bean's home interface findByPrimaryKey() method and an entity bean does not already exist with that primary key ?
- Is it possible to use container-managed persistence and take advantage of database triggers and sequences to populate the primary key? If so are there any good examples?
- Are there any examples and best practices for implementing the hashCode() and equals() methods for a compound primary key class?
- Where and when should I use the context.getPrimaryKey() method rather than just accessing the Primary Key field(s) directly?
- Can I re-use the same Primary key class for various Entity Beans?

Entity Beans allmänt

- Which fields in beans should be public?
- Should I use Entity Beans for all database tables, even though they may have complicated relations and joins? Or should I just use them for frequently-used and simple tables such as account info?
- When I want to add/remove a record from database, should I use the methods directly from Entity Bean or I should use a Session Bean to create some methods which will call the methods in the Entity Bean?
- Is there a guarantee of uniqueness for entity beans?
- How can I make a clone or duplicate of an entity bean? Also, can I do so within another EJB which does not have any knowledge about the entity bean it is referring to? I need to build an EJB which will take an(y) entity bean as input, make a copy and return the copy.
- Is there any default cache management system with Entity beans ? In other words whether a cache of the data in database will be maintained in EJB ?
- Can I set the data represented by the BMP Entity Bean to null in ejbStore, once it's updated in the database?
- Is it legal to use JDBC code inside the business methods of an Entity Bean? What are the implications of doing so?
- What is the difference between a "Coarse Grained" Entity Bean and a "Fine Grained" Entity Bean?

- How does Container Managed Persistence work with automatically generated database ID fields? Should I map the ID field explicitly or leave it unspecified?
- Can an Entity Bean call a stored procedure using Container Managed Persistence ? If yes, how?
- Can you control when passivation occurs?
- For Entity Beans, What happens to an instance field not mapped to any persistent storage,when the bean is passivated?

Stateful SessionBean

- When using a stateful session bean with an idle timeout set, how can the bean receive notification from the container that it is being removed due to timeout?
- When using Stateful Session Beans, What will happen if an EJB client program terminates without calling the Stateful Sesson Bean's remove() method?

Övrigt

- What are the constraints or drawbacks of container managed EJB's ?
- How do you model for a JOIN operation in a relational database using entity bean(s)? Do you have to have separate entity beans for each table in the database?
- How is the passivation of Entity beans Managed?
- When does the container call my bean's ejbCreate / ejbPostCreate / ejbStore /ejbPassivate / ejbActivate / ejbLoad / ejbPassivate method? And what should I do inside it?
- What is passivation and activation?
- Is it possible to have a Container Managed Persistent Entity Bean which refers to a view rather than a table.
- Why does an update trigger is activated when I access a method in a CMP EJB mapped to the table with that trigger?
- In EJB 2.0, What is an abstract persistence schema?
- What is the advantage of using Entity bean for database operations, over directly using JDBC API to do database operations? When would I use one over the other?
- The EJB container implements the EJBHome and EJBObject classes. For every request from a unique client, does the container create a separate instance of the generated EJBHome and EJBObject classes?
- When using EJB and making use of connection pools. How can changes to the database be audited per user?
- Can the activation/passivation of beans be tuned so that passivation is NOT done unless absolutely necessary. Which servers support this, and how?

- In EJB 2.0, What is a cmr-field. How can I differentiate between a cmr-field and a cmp-field?

Transaktioner

- How does a client application create a transaction object?
- Do JTS implementations support nested transactions?
- Why would a client application use JTA transactions?
- How does a session bean obtain a JTA UserTransaction object?
- How does an enterprise bean that uses container-managed transactions obtain a JTA UserTransaction object?
- Is an EJB, JMS, or general-purpose Java application server a transaction manager?
- Is entity data persisted at the end of a transaction or at any time?
- When should I use bean-managed transactions instead of specifying transaction information in the deployment descriptor?
- Explain the different Transaction Attributes and Isolation Levels with reference to a scenario.
- Can I use the afterBegin, beforeCompletion, and afterCompletion methods in a BMP bean? (as these are the methods of the SessionSynchronization interface)
- How do I insert or update 2 different Entity Beans within the same transaction?
- How do the six transaction attributes map to isolation levels like "dirty read"? Will an attribute like "Required" lock out other readers until I'm finished updating?
- Is it possible to get notified somehow if a certain amount of time elapses during (within) a transaction or method call? I want to react if e.g. an EJB-method doesn't react after a few seconds.
- For session beans, we can use the SessionSynchronization interface. For entity beans, how do we have control over a transaction?
- What is the default transaction attribute for an EJB?
- How can I find out the cause of a javax.transaction.TransactionRolledbackException?
- Is there any way to read values from an entity bean without locking it for the rest of the transaction (e.g. read-only transactions)? We have a key-value map bean which deadlocks during some concurrent reads. Isolation levels seem to affect the database only, and we need to work within a transaction.
- With regard to Entity Beans, what happens if both my EJB Server and Database crash, what will happen to unsaved changes? Is there any transactional log file used?
- How do you configure a session bean for bean-managed transactions?

- How do you configure the transaction characteristics for a session bean with container-managed transactions?
- My session beans call other bean methods within a transaction. Using bean-managed transactions, how should I take care of commit and rollback ?
- Is it possible to stop the execution of a method before completion in a SessionBean?
- Why would a session bean use bean-managed transactions?
- Is there a way that a Stateful Session Bean can remove itself when a transaction commits? Without this, the remove must be performed by the client that starts the transaction, which can be difficult when there may be non-stateful beans in-between.
- Is it possible for a stateless session bean to employ a JTA UserTransaction object?
- To complete a transaction, which Transaction Attributes or Isolation Level should be used for a stateless session bean?
- How do you configure the transaction characteristics for a session bean with container-managed transactions?
- Is it necessary for an entity bean to protect itself against concurrent access from multiple transactions?
- Is there any way to force an Entity Bean to store itself to the db? I don't wanna wait for the container to update the db, I want to do it NOW! Is it possible?

Säkerhet

- When can we hope to see a comprehensive security spec for EJB?
- Can the security context be passed between a web container (servlets/jsp) and an EJB container ? When the servlet calls an EJB I want its identity to be that of the original web client authenticated with a certificate.
- How to setup access control in an EJB such that different application clients have different rights to invoke different methods in one EJB?
- If Client C calls EJB A, and EJB A calls EJB B, then will the methods in B be invoked with the same security context of Client C?
- Does the EJB specification allow for such products as the Java Cryptography Architecture (JCA) and other security APIs, to be used from within session beans? What are the potential issues that would need to be addressed from doing this?
- The EJB 1.1 spec eliminated "runAs" security delegation. Is there some other way to achieve this effect now? For example, how can I enable session bean method A, which is running as "guest" to call entity bean method B which only allows some role not containing "guest" to execute it? Assume I will do some kind of authentication inmethod A prior to the call to B (and only do the call to B on behalf of authorized callers) so this makes sense.

- Is is possible for an EJB client to marshall an object of class java.lang.Class to an EJB?
- How can EJB's be accessed through Firewalls?

Samtidighet

- What happens when two users access an Entity Bean concurrently?
- OK, so EJB doesn't support user-created threads. So how do I perform tasks asynchronously?
- Why don't Session Beans have to be re-entrant?

Klient Access –Distribuerad access

- What classes does a client application need to access EJB?
- Which, if any, of the automatically generated stub & skeleton class files does one need to include in a JAR file to be used when creating stand-alone clients? (I would like to create a client-side jar file which contains only the class files actually needed by the client.)
- How can I access EJBs from JavaBeans Components?
- How can I call one EJB from inside of another EJB?
- Is there a way to dynamically lookup EJB interfaces, comparable to CORBA's dynamic invocation interface (DII)?
- Is it safe to pass EJBObjects as parameters to remote objects that may be on another machine, or must you use the handle instead?
- When are stubs & skeletons created for my EJB components? is it at development time, deployment time or run time?
- What is the role of serialization in EJB?
- Does EJB 1.1 support mandate the support for RMI-IIOP ? What is the meaning of "the client API must support the Java RMI-IIOP programming model for portability, but the underlying protocol can be anything" ?
- Does RMI-IIOP support dynamic downloading of classes?
- How can an applet talk directly with EJB?

EJB och andra Java-teknologier (JSP/Servlets JNDI etc)

JSP/Servlets, integration

- How can my JSP page communicate with an EJB Session Bean?
- Let's assume I use a JavaBean as a go-between a JSP and an EJB, and have, say, 50 concurrent clients that need to access the EJB functionality. Will the JSP container actually instantiate 50 instances of the bean, or can it reuse a single instance to access the EJB?
- What is the most efficient approach for integrating EJB with JSP? Should the EJBs be invoked directly from within JSP scriptlets? Should the access take place from within Java beans? Or is it best to use custom tags for this purpose?

- What is the best way of implementing a web application that uses JSP, servlet and EJB technologies all together following a Model View Controller (MVC) architecture?
- Can an EJB handle (RMI remote reference) be stored in an HttpSession?
- I have created a remote reference to an EJB in FirstServlet. Can I put the reference in a servlet session and use that in SecondServlet?
- How can I invoke an EJB from a servlet?'
- Why use EJB when we can do the same thing with servlets?
- Is there an example of a shopping cart using Stateful Session Beans and a JSP client?
- Architecturally speaking, under what circumstances must I consider adding EJB into the mix when building web-based applications using servlets and JSP?

JMS/JNDI/ JDO/ JavaMail

- How can JMS be used from EJB 1.1?
- Does the J2EE reference implementation support the JMS?
- Can an EJB send asynchronous notifications to its clients?
- What is a Message Driven Bean, What functions does a message driven bean have and how do they work in collaboration with JMS?

JNDI

- How do you get a JDBC database registered with a JNDI name so that it can be accessed from an EJB?
- Are there any JNDI tutorials?
- What books cover JNDI (Java Naming and Directory Interface)?
- How do you use the JNDI ENC to access JavaMail?
- How do I use JNDI to locate the home interface of an EJB running on a different host? My client's InitialContext doesn't seem to know where the server is.
- When (manually) deploying an EJB where do you specify the JNDI name that the bean is bound to in the JNDI tree?
- When is the best time to use <ejb-link>? Do I have to use it everytime I want to make a reference from an EJB to another EJB?
- From R. Monson-Haefel's book on EJB I understand, that EJB clients that are EJB's themselves, use a somewhat different (namingcontext-)lookup method than 'regular' clients. But why ? Why the <ejb-ref> tag in the deployment descriptor and why the 'java:comp/env' namespace ?
- Is it possible to specify multiple JNDI names when deploying an EJB?
- When calling an EJB from another EJB should they both be packaged in same JAR?. Are the any preferences for Weblogic or any other servers?

JDO

- How can I use JDO with EJB?

JavaMail

- Is there anything special about sending mail with JavaMail when using EJB?

Frågor som berör produkter från olika tillverkare

Olika J2EE-servrar

- Is there a J2EE server for RedHat Linux 6.1?

Olika EJB-servrar

- Is there a EJB Server available for Win98?
- What code changes do we need to make to upgrade our EJBs developed in Weblogic 4.5 to work under Weblogic 5.1?
- Can I deploy a new EJB without restarting my server? (I'm using Weblogic.)
- What is "hot deployment" in WebLogic?
- Tools:AppServer:BEA WebLogic
- Are there any good hands-on EJB programming tutorials for WebLogic?
- Does Tomcat support EJB?
- Can someone explain in detail the different steps involved in adding 'finder' methods other than 'findByPrimaryKey' in IBM's VisualAge for Java?
- Is it better to use JRUN Connection Pooling or native Connection Pooling?
- Which application server is best for EJB deployment?
- I want to access a session EJB deployed on a Websphere 3.02 from the web container of a Weblogic 4.51 Server. Both servers are running on machines behind the firewalls of two different corporate networks. What are the viable solutions here?
- When I try to run J2EE SDK Deploytool.bat it doesn't start and gives: Exception in thread "main" java.lang.NullpointerExcception. How can I fix this?
- What are the free EJB containers available? Which version of the EJB specification do they support? Are they free for development and production?
- Cluster weblogicss editing while promoting?

Produkter att “plugga in” i AppServer

- Is there a profiler application that runs with BEA Weblogic App Server (v4.51) so that I can monitor my EJB's and server-side Java classes?

Olika databaser

- How do I connect to SQL Server 6.5 from J2EE for Windows? What changes should I have to do in the properties file? Can I use the JDBC-ODBC bridge?
- Is it possible to use Microsoft Access as the database while implementing entity beans? (Weblogic refuses to use the JDBC-ODBC bridge, since it needs a thin driver.)
- I would like to bind my EJB object (container managed or bean managed) to Oracle 8 DBMS_ALERT or DBMS_PIPE so that my object is notified whenever there is an update in the database by some batch process. Is it possible to do? If so, how?

IDEs

- Which IDEs are available that support EJB development? And are any free or low-cost?

Debuggers

- How can I debug my EJB applications?

Testing

- How can I use JUnit to test J2EE/EJB applications?

Integration med specifik servlet produkt

- Can I use the latest version of Tomcat inside Sun's J2EE implementation?
- Who offers hosting of EJBs?

VisiBroker

- Is it possible to access a CORBA object from a EJB? I am using VisiBroker 4.0 for my CORBA objects and J2EE 1.2.1 for my EJB.

EJBDoclet

- What is EJBDoclet?

Web Server

- Can I use the Apache Web Server with the Sun J2EE Reference Implementation?

Scheduling Framework

- Where can I find a vendor-independent Job Scheduling Framework for J2EE?

Övrigt

Integration med COM/C/ ILE/RPG/Oracle

Integration med COM

- How can I access my EJBs from COM environments such as VB?

- How can we interact with COM/DCOM components from a EJB component ?
- Are there any performance issues that have to be considered while using EJB Components from COM environments?

Integration med C

- What are the viable alternatives for calling EJBs from non-Java programs? Specifically, I'm interested in calling EJBs (probably indirectly) from plain old C language.

Integration med ILE/RPG

- How can a EJB talk to legacy applications written in ILE/RPG without using the IBM java toolbox?

Integration med LDAP

- How can we work with LDAP using EJB's?

Integration med oracle

- How can I call an EJB via an oracle stored procedure?

Resurser/böcker med information om EJB

What newsgroups and mailing lists are available to learn more about EJBs?

Are there any good books that cover EJB (and related topics like LDAP and so on)?

Are there any online EJB tutorials?

Where I could find good literature on J2EE architecture?

Where can I find "EJB/J2EE coding standards" ?

Jämförelse andra teknologier

Java:API:EJB

Why is EJB better than Microsoft COM/DCOM/MTS/DNA ?

How do I decide between Enterprise Java Beans (EJB), Remote Method Invocations (RMI), and CORBA?

Definition av Servertyper och arkitektur relaterade ord.

What is meant by the term "business logic"?

What is a three-tier architecture?

What are all the different kinds of servers? (Such as Web Servers, Application Servers, etc)

What is the difference between a Component Transaction Monitor (CTM) and an Application Server?

What is the difference between an Application Server and a Web Server?

Find-operationer

How should complex find operations be implemented?

Can I combine two or more finder functions with AND or OR operator to get one result collection?

Can my ejbFind methods have return types like RowSet or Vector, or should it be only of type Enumeration?

In CMP how can I define a finder method equivalent to a 'SELECT * FROM TABLE'?

What optimizations can be made to increase the performance of finder methods, especially when retrieving large result sets?

Can I specify specific WHERE clauses for a find method in a CMP Entity Bean?

Annat

Portabilitet

How can I write EJBs that will run (unmodified) in any EJB-Container? What do I have to pay attention to?

Söker komponenter för ett vist ändamål (eg affärsändamål)

Where can I find EJB components for retail, health care, and utility solutions?

Prestanda

How good is the performance of EJB systems? Are there any benchmarks?

Is there any performance problem with calling one EJB from another EJB running in the same container?

Callbacks

How do you implement callbacks in EJB?

CG

Given that RMI-IIOP does not support distributed garbage collection (unlike RMI-JRMP), do I need to do something explicitly to GC beans, or is it magically handled by the EJB framework?

Grundläggande programmering

How should I set the CLASSPATH to develop EJBs?

Listing Behavior

How do I get a list of records (like from a database query) in EJB?

Templates

Where can I find ejb-jar.xml file templates?

Deployment och jar-filen

Can I deploy two beans in a single jar file? If so, how?

Problem med deployment (som ger felmedelande)

Using Websphere for a small EJB application deployment, why do I

get the message "The jar file you have created does not contain any beans.Please select a jar file with beans in it".

Why not final?

Why is it that business methods should not be declared final?

Init parameters

How can I pass init parameters to enterprise beans?

Kan EJB användas till det här …

Are EJBs the best solution to make a component which have to manage a hardware device (modem, fax...) and provide access to them?

Native libraries

How do enterprise beans access native libraries?

Initial context factory

What is the initial context factory for sun's EJB reference implementation, and how is the initial context url specified? I always see the code fragment "InitialContext ctx = new InitialContext();" but where are the initial properties specified for this to work?

Java.io - logging system

How do I perform I/O operations from a bean?

How do I implement a logging system in my beans?

What is the best way to do a standard logging/tracking mechanism for a distributed architecture? We're using J2EE, EJB, load balancing, and multiple tiers.

EAR

What's an .ear file?

Stored procedures

Can beans use stored procedures in a database?

One-to-many many-to-one

Are there any good examples of how to handle one-to-many, many-to-one, and many-to-many entity bean relationships?

Clustering

What is "clustering" in EJB?

Value objects/ intermediate data access object

Is there any reason why value objects cannot have set methods?

What is an intermediate data access object and what are the advantages and disadvantages of using it?

What is a value object? Why would I use one? What are some problems with them?

Java 1.1

Can I use a Java 1.1 client to access an EJB?

Pattern

What is the Session wraps Entity Pattern? What are the motivations behind it?

Value classes

In EJB 2.0, What are dependent value classes?

Design patterns

What are some standard design patterns that are used in EJB?

Where can I find an example of the command pattern implemented with EJB?

DTD's

Where can I find the DTD's for EJB Deployment Descriptors?

How can I use a local DTD to validate ejb-jar.xml instead of going to SUN's server as specified in DOCTYPE element?

Serialized deployment descriptor

How to create serialized deployment descriptor in EJB 1.0?

ResultSet

How can I pass a ResultSet from an EJB to a client? what are the best classes to use?