Inventeringsmiljö för datasystem

Johan Jonsson

18 april 2006
Master's Thesis in Computing Science, 20 credits
Supervisor at CS-UmU: Lars-Erik Janlert
External supervisor at Land System Hägglunds: Gunno Hamberg
Examiner: Per Lindström

Umeå University
Department of Computing Science
SE-901 87 UMEÅ
SWEDEN

Abstract

In larger companies where computerbased systems are used on daily basis by many users, a lot of applications tend to be needed for each user specific purpose. It is not unusual that these applications depend on other to work. For example, a webserver needs a database server. In order to increase the availability, the different software are often placed on different servers. However, this may cause a problem when the network grows, which applications gets affected when a specific server goes down? This master thesis describes the result of creating a supervising tool showing dependencies between servers and applications. It can also supply the user of the tool information if anything is wrong and how this error is reproduced in the network. The thesis also includes a in-depth study on architectures which is strongly connected to the created application which is described later.

Sammanfattning

I större företag där datorbaserade system används som hjälpmedel på daglig basis av många användare behövs många olika applikationer för olika användare. Det är inte ovanligt att dessa applikationer är beroende av andra för att de ska fungera, till exempel kan en webbserver behöva en databas för att fungera. För att öka tillgängligheten i nätverk placeras ofta applikationer på olika servrar. Detta ställer dock till med ett annat problem när nätverket växer. Vilka applikationer blir drabbade när en server slutar fungera? Detta examensarbete beskriver resultat av implementationen av ett övervakningsverktyg som visar beroenden mellan både applikationer och servrar. Den kan även ge användaren information om något i nätverket är felaktigt samt hur detta fel fortplantar i nätverket. Rapporten innefattar även en fördjupningsstudie om arkitekturer samt hur detta är kopplat till skapandet av applikationen.

Innehåll

Kapitel 1

Introduktion

Allt fler företag har numera ett eget internt nätverk där flera servrar och applikationer körs. Initialt då det enbart finns en handfull servrar blir en felsökning vid eventuella problem inte speciellt problematisk, speciellt då en ansvarig person kan hålla ordning på allt. Om företaget expanderar och datoriseringen växer sig starkare är det inte ovanligt att detta nätverk växer i storlek, fler servrar, fler applikationer och fler beroenden mellan dessa applikationer införs. Detta ökar komplexiteten i nätverket dramatiskt och därigenom även felsökning samt förståelse av nätverket. Dessa problem gör det önskvärt att kunna monitorera sitt nätverk, att kunna se servrar eller applikationer som inte svarar och utifrån detta kunna ställa snabbare prognoser och förhoppningsvis minimera tiden som nätverket inte fungerar korrekt. Det kan dessutom vara önskvärt att kunna visualisera nätverket för att finna relationer som inte är nödvändiga och därigenom kunna rationalisera samt även kunna förutsäga vilka applikationer som kommer att drabbas om en viss server stängs av för underhåll.

Denna rapport är en del i ett examensarbete som är utfört vid instutitionen för datavetenskap, Umeå universitet. Uppdragsgivare för detta examensarbete är Land System Hägglunds vilket är ett företag med 1100 anställda som är stationerat i Örnsköldsvik både då det gäller huvudkontor samt produktion. Land System Hägglunds AB är ledande tillverkare av stridsfordon och terränggående fordon. Numera ägs företaget av BAE Systems, vilket är ett engelskt företag som har specialiserat sig på försvarssystem. För att belysa Hägglunds ledande position kan det nämnas att de levererat till mer än 40 länder och hade 2004 en omsättning på 3 miljarder kronor.

1.1 Rapportens uppbyggnad

Kapitel 2 Behandlar problembeskrivning, mål, avgränsningar samt relaterat arbete.

Kapitel 3 Behandlar arkitekturer, dess termer samt vilken betydelse de kan ha i ett system. I detta kapitel beskrivs även kopplingen mellan prototypen och arkitekturer.

Kapitel 4 Beskriver planering av projektet.

Kapitel 5 Behandlar implementationspecifika delar

Kapitel 6 Innehåller slutsatser

Appendix Användarhandledningen

Kapitel 2

Problembeskrivning

Nedan följer en förklaring hur prototypen ska se ut och vilka delas som förväntas vara med.

2.1 Beskrivning

Applikationen kan delas upp i fyra större delar: gränssnitt, databas, server och klient. Eftersom klienten ska installeras på samtliga servrar i nätverket var ett krav på den att den skulle hållas tunn för att inte tillföra problem på servrarna. För struktur se Bilaga C.

2.1.1 Server

Servern ska stå och ta emot registreringar samt hålla reda på vilka servrar som lever. Den ska vara passiv, det vill säga bara lyssna på inkommande anrop och inte själv aktivt efterfråga något. Vid förfrågan från klienter ska den söka upp den efterfrågade datan från databasen och leverera denna data till klienten. Likt klienten och gränssnittet ska denna vara programmerad enligt tre-skiktmodelllen.

2.1.2 Klient

Klienten ska installeras på samtliga servrar i nätverket. Klient ska köras som en tjänst på servrarna och anropa servern för att initialt registrera sig och sedan för att meddela att den fortfarande är uppe. Tidsintervallet mellan varje inrapportering ska kunna ändras. Klienten ska dessutom kunna skicka data som kan indikera att det är något problem på servern. Sådan data skulle exempelvis kunna vara ledigt ramminne och processoranvändande. Eftersom det existerar olika operativsystem på servrarna i nätverket och samtliga ska kunna monitoreras ska klientlösningen vara helt plattformsoberoende.

2.1.3 Grafisk användargränssnitt

För att kunna övervaka systemet ska det finnas ett gränssnitt. I detta ska intervallet som anger hur ofta klienten ska rapportera till servern kunna ändras. Samtliga förändringar i gränssnittet ska lagras på permanent minne i databasen. I gränssnittet ska det även gå att ställa in vilka applikationer som ska monitoreras på en viss server i nätverket. Det

ska även finnas en möjlighet att fylla i vilka beroenden som finns i systemet samt kunna presentera nätverket i en graf i vilken det ska synas hur nätet ser ut och därmed ge möjligheter till att upptäcka eventuella felaktigheter. Gränssnittet ska vara webbaserat.

2.1.4 Databas

Databasen ska ha mycket av sin intelligens lagrad i sig själv i form av lagrade procedurer samt triggers. Den bör dessutom vara dynamisk för att förenkla eventuella utbyggnationer i framtiden.

2.2 Begränsningar

- Klienten behöver bara implementeras i Windows miljö.
- Språket i gränssnittet behöver enbart vara svenska

2.3 Mål

Målet med detta examensarbete är att skapa en bra grund till nätverksövervakning i form av en prototyp som är väl fungerande samt att få denna prototyp enkel och rak i sin design. Vidare ska fördjupningen kunna ge en grund inom mjukvaruarkitektur både för extern och intern personal.

2.4 Metoder

Relativt mycket var specifierat i förväg. Det största valet av metod som var tvunget att göras var att välja hur klienten skulle kommunicera med den lokala servern. Kravet på operativsystemoberoende var det enda som avgränsade. En annan faktor som också var värd att ha i åtanke var att systemet skulle vara passivt, det vill säga tjänsten skulle rapportera in till server medan den motsatta kommunikationen inte skulle existera. De två egentliga valmöjligheter som fanns var CORBA[1] samt webbservice[2], möjligheten att implementera en egen server-klient modell samt göra ett eget protokoll förkastades då det fanns två standarder som var lämpliga.

Fördelen med webbservice är att den skickar XML-dokument via HTTP[3] vilket betyder att två standarder används. HTTP skickar data via port 80, en port som är öppen i många brandväggar. Detta medför att en eventuell integrering blir enklare då ingen brandvägg behöver konfigureras. För att kunna kommunicera med CORBA-objekt måste ORB-biblioteken stödjas av klienten vilket innebär att klienten där webbservice tillämpas blir tunnare, vilket var ett krav i specifikationen. Det är förvisso inte särskilt kostnadseffektivt att parsa XML-dokument men eftersom en viss fördröjning inte påverkar systemet nämnvärt i denna applikation föll valet på webbservicen. En annan faktor som också påverkade valet var att webbservice var ett nyare område än CORBA och därmed ansågs mer intressant.

[1] Common Object Request Broker Architecture
[2] Se Appendix för förklaring av dessa
[3] HyperText Transfer Protocol

2.5 Relaterat arbete

De två delarna av relaterat arbete som är relevanta att diskutera är dels ett protokoll samt en tillgänglig produkt.

2.5.1 Microsoft Operations Manager 2005 (MOM)

Under examensarbetets gång har Microsoft lanserat sin produkt MOM vilken är ett övervakningssystem som har i uppgift att hålla reda på nätverket samt beroenden[19]. Författaren av denna rapport har inte haft möjlighet att testa detta och således kan ingen jämförande studie presenteras i denna rapport då det gäller skillnader och likheter mellan MOM och prototypen i detta examensarbete.

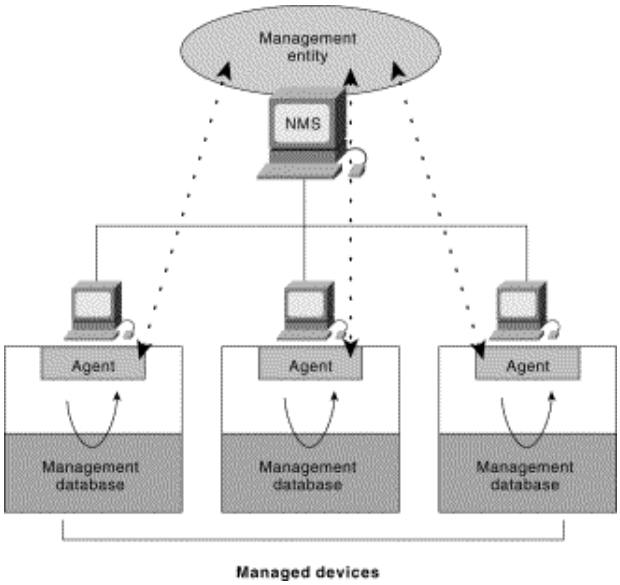
2.5.2 Simple Network Management Protokoll (SNMP)

SNMP är ett applikationslagerprotokoll som möjliggör att nätverksadministratörer kan hitta och lösa nätverksproblem samt planera för en utbyggnad av nätverket[23]. SNMP har utkommit i två versioner: SNMP version 1 och SNMP version 2 där version två är en utbyggnad av föregångaren med tillägg av operationer i protokollet. Protokollet består av tre grundkomponenter (se figur 2.1):

- `Monitorerad enhet`, även kallad nätverkselement, kan exempelvis vara en switch, skrivare eller en dator. Dessa noder innehåller en SNMP agent och samlar in och sparar monitoreringsdata och gör denna tillgänglig för NMS:s genom användande av SNMP. Om en speciell händelse inträffar skickar denna enhet ett *trap*-kommando till sin NMS.
- `Agent` är en mjukvarumodul i den monitorerade enheten. Agenten har information om den lokala datan och översätter denna till ett format som är kompatibelt med SNMP.
- `NMS` (Network-management system) kör den programvara som monitorerar och kontrollerar den monitorerade enheten. I ett monitorerat nätverk måste det finnas minst en NMS. NMS kan utföra tre kommandon på en monitorerad enhet, nämligen *read*, *write* samt *Traversal operations*. *Read* och *write* utförs av NMS på variabler i den monitorerade enhetens minne. *Traversal operations* används för att en NMS ska kunna avgöra vilka variabler en monitorerad enhet tillhandahåller.

Flera olika PDU:er[4] är fördefinierade för att sända trafik på ett kontrollerat sätt.

[4]Protocol data unit



Figur 2.1: Hur grundkomponenterna interagerar

Kapitel 3

Arkitektur och dess betydelse

3.1 Introduktion

Vad är en arkitektur? Susning[1] förklarar arkitektur på följande sätt:

> Arkitektur är detsamma som husbyggnadskonst. Ordet härleds ur yrkesrollen arkitekt, som idag är en titel som (i Sverige) inte kräver examen från teknisk högskola. Sådana examina finns, men titeln arkitekt är inte skyddad.

Historiskt gällande arkitektur anses Marcus Vitruvius Pollio vara en pionjär[26], dock inte för att han var den första betydande arkitekten utan snarare att han var först att skriva om arkitekturer. Hans verk, bestående av 10 böcker finns än idag bevarade och hans idéer påverkar än idag arkitekter. Han menar att arkitektur består av tre viktiga beståndsdelar:

- `Utilitas` är idén om användbarhet. Den struktur som byggs ska vara brukbart för sitt syfte. En vital del i denna princip är att syftet blir ordentligt beskrivet i specifikationen. För att detta ska kunna ske krävs det ofta att arkitekten hjälper till med sin kunskap vilket i sig kräver att arkitekten initialt bör skaffa sig god förståelse om problemet. Andra områden som innefattas av Utilitas är produktens flexibilitet, dvs. att den ska kunna anpassas till nya behov samt nya situationer.
- `Firmitas` innebär att produkten som byggs ska vara stark, uthållig, klara av föråldring samt ett flitigt användande.
- `Venustas`, principen som behandlar skönhet innebär att arkitekten är ansvarig för att en användare ska känna sig bekväm att använda eller vistas i en viss konstruerad struktur. Ur en mjukvaruarkitektonisk synpunkt kan denna punkt anses vara den som fått minst uppmärksamhet även om det är vitalt att en användare känner att strukturen är behaglig.

Ur mjukvarusynpunkt kan det sägas att om ett system, på ett begränsat utrymme, beskrivs så att designers, användare, programmerare och chefer kan förstå vad systemet gör, förstå hur det fungerar, utöka systemet, arbeta med delar av detta samt återanvända en del av systemet för att producera ett annat system har en beskrivning av systemets

[1] www.susning.nu

arkitektur gjorts, dvs arkitektur är när allt överflödigt skalats bort och inga fler saker kan tas bort utan att det påverkar förståelsen av systemet[20].

Arkitektur är ett ord som allt oftare nämns inom mjukvarukonstruktion. Numera är de flesta enklare system som kanske inte nödvändigtvis krävde en arkitektur (även om det skulle vara önskvärt) för att kunna genomföras implementerade. När dessa system ska utvidgas eller nya komplexa system byggas ställs en fråga på sin spets: Hur görs detta på det mest kostnadseffektiva sättet utan att försumma kvalitet och samtidigt erhålla ett lyckat system som följer kraven? Arkitektur är ett sätt att bygga en bro mellan kraven på ett system och dess kod[10]. Paul Clements m.fl.[3] tar upp tre skäl varför mjukvaruarkitektur är viktigt, speciellt i komplexa system:

- Systemarkitekturen som representerar system på en hög nivå ger samtliga stakeholders[2] en möjlighet att skaffa sig en gemensam förståelse och genom kommunikation skapa konsensus.
- Mjukvaruarkitekturen för systemet visar på de första designbesluten som tagits. Dessa beslut visar kompromisser mellan krav och önskemål och enligt författarna till boken är arkitekturen den artefakt som har störst inverkan av alla på ett systems kvalitet.
- Arkitektur är en överföringsbar abstraktion av systemet och dess arkitekturer är återanvändbara. Arkitekturen är en relativt liten modell av systemet som bland annat visar hur komponenter samverkar, en modell som kan överföras till andra projekt.

3.2 Skillnaden mellan design och arkitektur

Vad är skillnaden? En arkitektur behandlar frågor som rör hur, varför och inte bara vad. Således skulle design kunna anses vara en delmängd av arkitektur. Enligt Grady Booch[5] är all arkitektur design men inte all design är arkitektur. Det är förenklat uttryckt men förmedlar väl vad den övergripande skillnaden är. Vidare menar Booch att arkitekturer fokuserar på beslut som både påverkar strukturen samt viktiga beteenden, även beslut som kommer ha en varaktig effekt på system rörande ekonomi, tillförlitlighet, flexibilitet behandlas av arkitektur men inte design. Paul Clements m.fl.[3] försöker urskilja kriterier för vad som är arkitektoniskt. De menar att enbart de komponenter, relationer mellan komponenter eller egenskaper som behöver vara externt synliga för att diskussioner ska kunna säkerställa att systemet uppnår de satta kvalitetskraven är arkitektoniska. Denna kompakta definition kan utökas i tre punkter:

- Informationen i en beskrivning av en arkitektur ska vara den mest abstrakta men samtidigt den mest meningsfulla beskrivningen.
- En arkitektur ska vara möjlig att förstå. En för noggrann beskrivning medför att det blir svårt att diskutera och förstå densamma. Detta gör att syftet med beskrivningen försvinner.
- Arkitekturen ska vara begränsande.

[2] Översättning från http://lexikon.nada.kth.se

Det finns dock ett problem med detta som Paul Clements m.fl.[3] belyser. Arkitekturen ska vara abstrakt och bör därför till exempel inte beskriva specifika datastrukturer eller algoritmer. Däremot kan vissa algoritmers egenskaper påverka systemet märkbart och bör därför vara med i arkitekturen även om det inte är en abstrakt beskrivning av systemet. I slutändan är det således arkitekten som avgör var skillnaden mellan design och arkitektur ligger.

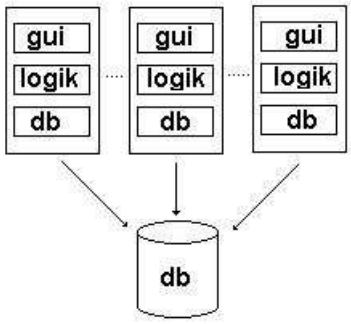
Kapitel 4

Delar i en arkitektur

En arkitektur består av flera olika delar. Nedan följer en förklaring på ett urval på de delar Land System Hägglunds ville ha med.

4.1 Referensarkitektur

Kort förklarat är en referensarkitektur(RA) ett regelverk för hur applikationer ska skapas och skiktas. Den specificerar även vilka komponenter som är valbara samt vilka som måste vara med i system. Den kan även behandla hur olika komponenter ska kommunicera med varandra. En viktig del med RA är att den inte ska tillföra någon komplexitet utan ska vara flexibel och förenkla utvecklingen. Om den försvårar utveckling är risken stor att den blir en papperstiger som inte används. Vanligen innehåller en RA ingen information rörande innehållet i komponenten. Se figur 4.1 för ett enkelt exempel på en RA:
Detta företag har bestämt sig för att applikationerna ska vara tre-skiktade. Dessa ska



Figur 4.1: En skiss på RA i ett litet företag

enbart kommunicera med en databas för att få fram information samt spara information. Ytterligare tillägg kan vara hur de olika delarna gränssnitt, logik och databas ska

kommunicera med varandra. Dessutom kan ett gränssnitt specificeras till att vara ett webbgränssnitt.

Att använda en referensarkitektur medför flera fördelar:

- Effektiviteten ökar, detta kan styrkas med att alla vet vad som ska göra
- Kvaliteten ökar eftersom referensarkitekturen specificerar hur saker får göras vilket minskar felaktigheter och därmed producerar säkrare system.
- Individberoendet minskar då alla känner igen sig i de olika lagren och kan med relativt små ansträngningar ta vid där en annan utvecklare slutade.
- Gemensamma komponenter kan användas med hjälp av likformighet och att främja återanvändbarheten ökar effektiviteten.

4.2 Arkitektoniska ramverk

4.2.1 Zachmans ramverk

1987 presenterade John Zachman sitt ramverk för informationssystemarkitekturer[21]. Det är ett annorlunda tillvägagångssätt att beskriva en process, istället för att presentera processen som en serie av steg organiseras den som en matris (se figur 4.2). Raderna i denna matris utgörs av olika aktörers perspektiv på hur utvecklingsprocessen bör se ut. Zachman har definierat dessa vyer[21]:

Planerarperspektiv (Scope) - definierar verksamhetens syfte och strategier. Vidare begränsas målet för verksamheten i denna vy.

Ägarperspektiv (Enterprise model) - definierar i verksamhetstermer strukturen, funktionen och organisationen med mera.

Designperspektiv (System model) - denna vy, även kallad arkitektvyn beskriver `Ägarperspektivet` men med mer inriktning på information.

Utvecklarperspektiv (Technology model) - definierar hur teknik kan användas för att hantera informationsprocesserna som beskrivits i raderna ovan. I denna rad väljs bland annat relationsdatabaser, programspråk, gränssnitt definieras samt skapas programstrukturer.

Underleverantörsperspektiv (Detailed representations) - i denna vy listas databasspecifikationer, nätverket med mera.

Fungerande affärssystem (Functioning enterprise) - denna innehåller bland annat fungerande databasen och exekverbara filer. Även den färdiga produkten behandlas i denna rad.

Till varje rad i Zachmans ramverk finns en tillhörande kolumn[21][16] innehållandes vad som ska undersökas av de olika aktörerna.

	data (what)	**process** (how)	**network** (where)	**people** (who)	**time** (when)	**motivation** (why)
Scope (planner)	List of important entities	List of business processess	List of operating locations	List of important organizationals	List of significant events	List of business Goals
Enterprise Model (Owner)	Semantic Model	Business Process Model	Logistic Network	Work Flow Model	Master Schedule	Business Plan
System Model (Designer)	Logical Data Model	Application Arch.	Distributed System Arch.	Human Interface Arch.	Processing Structure	Business Rule Model
Technology Model (Builder)	Physical Data Model	System Design	System Arch.	Presentation Arch.	Control Structure	Rule Design
Detailed Representation (Subcontractor)	Data Definition	Program	Network Arch.	Security Arch.	Timing Defintion	Rule Specification
Functioning Enterprise	Data	Function	Network	Organization	Schedule	Strategy

Figur 4.2: Zachmans ramverk

Data (vad) - varje rad i denna kolumn behandlar verksamhetsdata ur olika aspekter

Funktioner och procedurer (hur) - beskriver hur verksamhetens data används.

Nätverk (var) - denna vy berör de geografiska aspekterna såsom var affärer görs men även var datan finns samt hur denna data ska distribueras.

People (vem) - beskriver vilka som är inblandad i verksamheten samt vilka som är med i införandet av nya teknologier. Denna kolumn innehåller även en hierarki över personalen, en hierarki vilken kan avgöra vilka som får komma åt en speciell typ av data.

Time (när) - beskriver tidsplanering av processer, exempelvis kan det röra sig om en bestämd tid som aktiverar en process. En annan möjlighet kan vara en sekvensiell tidsordning som säger att en viss process måste vara klar innan nästa påbörjas.

Motivation (varför) - beskriver verksamhetens mål och strategier.

4.3 Arkitektoniskt mönster

Christopher Alexander har definierat mönster inom byggnadarkitektur: ”Ett mönster beskriver ett återkommande problem samt beskrivs de vitalaste delarna i lösningen till detta problem. Denna lösningsbeskrivning ska dokumenteras på ett sådant sätt att den ska kunna appliceras miljontals gånger utan att någon lösning blir den andra lik”[9]. Precis som tidigare är influenserna från byggnadarkitektur stora och Alexanders definition täcker väl in vad som menas med mönster inom mjukvaruarkitektur. Vid applicering på mjukvaruarkitektur kan mönster sägas dokumentera befintliga designerfarenheter som är väl beprövade med en abstraktionsnivå som ligger över både klass och komponent nivån. ”The Open Group”[1] definierar 10 punkter för att beskriva ett mönster[13]:

- Namn: Ett kort och koncist namn som väl beskriver mönstrets innebörd. I de fall där ett mönster har flera namn eller alias bör även dessa dokumenteras här. Likaväl som en skyskrapa framkallar bilder om hur en viss byggnad ser ut bör detta namn reflektera mönstret.

- Problem: En förklaring som beskriver målet samt målsättningen med mönstret. Dessa målsättningar motsägs dock ofta av kraven, vilket medför att det ofta införs en prioritering i denna del.

- Kontext: Anger vilka nödvändiga förutsättningar som krävs för att mönstret ska vara applicerbart. Kontexten beskriver också systemets initiala tillstånd innan mönstret appliceras.

- Krav: Här anges vilka krav som finns på systemet samt även hur kompromisser ska göras då olika krav oftast innebär en intressekonflikt. Exempel på krav kan vara modularitet, skalbarhet och olika aspekter på säkerhet.

- Lösning: Beskriver med ord och/eller bilder hur målen ska nås. Denna punkt kan även inkludera riktlinjer för implementering av lösningen.

- Resulterande kontext: Beskriver tillståndet efter att mönstret blivit applicerat.

- Exempel: Beskriver en praktiskt tillämpning av mönstret. Denna punkt är starkt kopplad till Användningsområden och det händer att Användningsområden utelämnas till fördel för Exempel.

- Logisk grund: Detta är en förklaring hur mönstret som helhet eller dess individuella delar fungerar samt en förklaring av olika regler och steg i mönstret. I och med detta erhålls även, förutom förståelse, en bekräftelse på att mönstret är funktionellt.

- Besläktade mönster: Denna punkt beskriver mönster som blivit utvecklade från detta mönster samt eventuellt mönster som detta dito blev utvecklat utifrån. Det beskrivs även vilka andra mönster som kan appliceras med detta.

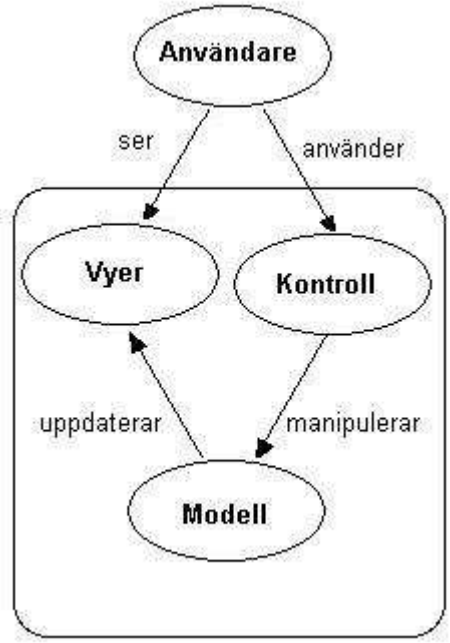
- Användningsområden: Beskriver kända applikationer som tillämpar mönstret dokumenteras. Detta kan ses som en ytterligare bekräftelse om att mönstret beskriver en lösning på ett återkommande problem.

[1] www.opengroup.org

Det finns olika sätt att kategorisera mönster inom mjukvaruarkitektur[13]. Dels kan ”design pattern” användas som ett samlingsnamn för samtliga mönster inom mjukvaruarkitektur. Frank Buschman m.fl.[13] har definierad en annan kategorisering där mönster delas in i tre typer: Arkitektoniska mönster, design pattern och idiom. I denna rapport kommer mönster användas konsekvent med syftning till samlingsnamnet ”design pattern”.

Mönster behandlar ingen design. Det innebär att inget mönster kommer att innehålla färdiga klasser med exempelvis listor eller hashtabeller som kan återanvändas[9]. De innehåller inte heller några algoritmer[22] utan bara ett tillvägagångssätt. Vid applicering av mönster är viktigt att inse att detta inte är en garanti för att ett projekt ska lyckas. Beskrivningen av mönstret visar bara på när det är applicerbart men för att veta om det kommer att förbättra slutprodukten krävs erfarenhet[25].

4.3.1 Model-View-Controller(MVC)



Figur 4.3: En modell som beskriver MVC.

Trots att MVC är väldigt användbart är det inte speciellt komplicerat. Mönstret består av tre moduler[30](den fjärde ringen i bilden innefattar ingen kod utan är en vanlig användare):

Modell - Modellen är det centrala i en MVC, den innehåller all information samt tillstånd rörande det applikationen representerar. Vid en förändring i modellen uppdaterar den samtliga vyer[4].

Kontroller - Tillhandahåller ett gränssnitt med vilka funktioner modellen kan manipuleras.

Vyer - Vyn visar modellrelaterad information till användaren, samtliga objekt som behöver information från modellen måste vara en registrerad vy hos modellen.

Ett exempel är en målservicetjänst där en användare via kontrollern registrerar match och mål i modellen som i sin tur uppdaterar vyn hos samtliga användare med nuvarande resultat. Att bryta ner en applikation i dessa tre delar genererar flera fördelar. Dels kan flera olika vyer skapas, detta skapar en mångfald i ett program eftersom data kan representeras olika och därmed fylla fler användares krav.

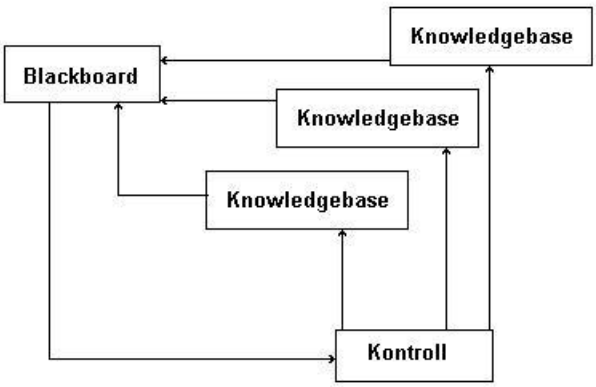
4.3.2 Blackboards

I ett blackboard mönster samarbetar flera specialiserade delsystem tillsammans mot gemensam data för att få fram en approximerad lösning på ett problem. Ett exempel på ett sådant problem är röstigenkänning[11]. Detta mönster består tre delar[11]:

Blackboard - innehåller rådatan. En blackboard kan inte ändra på sin egen data utan ändring sker enbart av knowledge source, en ändring som sker steg för steg till dess att en lösning är nådd.

Knowledge source - en samling oberoende applikationer där var och en har en specifik uppgift. Interaktionen mellan dessa sker enbart via blackboarden. Var och en knowledge source kan ses som en specialist.

Control - drivs av vilket tillstånd blackboarden befinner sig i och schemalägger utifrån detta vilken knowledge source som ska modellera datan.



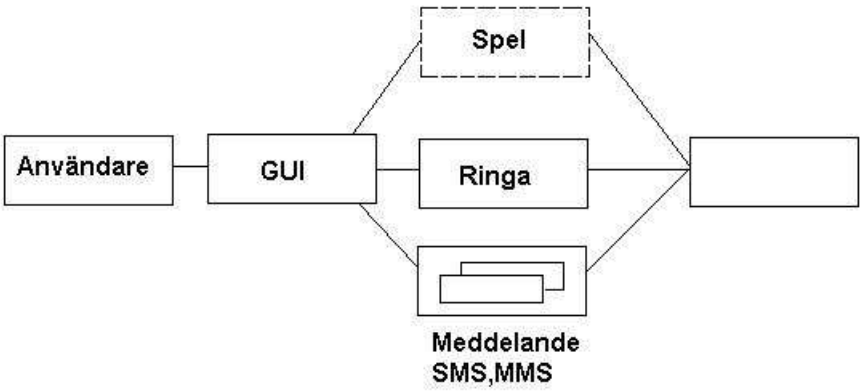
Figur 4.4: En modell som beskriver blackboard.

4.4 Produktlinjearkitektur

Ett sätt att maximera nyttan och minska kostnaderna för en viss investering är att producera flera liknande system och återanvända arkitekturen i dessa. Detta skulle minska kostnaderna eftersom skapandet av arkitekturer tar en tredjedel av den totala tiden för ett system[26]. Denna återanvändning är precis vad produktlinjearkitektur fokuserar på[15]. Fördelen med en produktlinjearkitektur är att en mängd simultana produkter kan utvecklas mer kostnadseffektiv eftersom de moduler som används är likadana. Produktlinjearkitekturen(PLA) definierar komponenter och hur de interagerar med varandra samt dess funktionalitet. Om de valbara enheterna i en PLA är valda samt de varierande enheterna är fastställda fås en instans av en PLA vilket heter produktarkitektur(PA). Nedan specificeras de olika elementen som kan finnas i en PLA.

- Element som finns med i alla produkter. Dessa markeras med en rektangel med en heldragen linje
- Element som är valbara att ha med. Notationen av dessa är en rektangel med streckad linje.
- Element som alltid finns med men är införda i systemet i olika varianter. Flervalen finns med som mindre block i ett stort.

Om dessa element appliceras i ett praktiskt sammanhang rörande en basal mobiltelefon, där streck berättar om deras relationer till andra element kan ett exempel på en PLA visualiseras enligt figur 4.4.



Figur 4.5: Exempel på en PLA där användare och gränssnitt alltid finns med medan spel är valbart och meddelande alltid finns med men i olika former. En PA kan vara att SMS är valt som meddelandetyp och spelenhets är exkluderad.

För att kunna testa en PLA behövs någon teknik. Det finns två egentliga val, antingen göra ett nytt test från grunden eller försöka anpassa ett befintligt. Eftersom PLA och vanlig arkitektur delar många likheter anser Muccini m fl.[15] att existerande tester inom mjukvaruarkitektur kan anpassas till tester för PLA.

4.4.1 Enhetstestning

I en enhetstestning genomgår varje enskild modul en buggtestning. I exemplet rörande mobiltelefonen kan det finnas tre olika moduler och samtliga dessa genomgår varsin enhetstestning. Vilken ordning testning sker i kan baseras på en rangordning av komponenterna sinsemellan. Rangordningen är valbar men Muccini m fl.[15] rekommenderar att komponenterna som alltid ska finnas med testas först. Därefter bör de två andra enhetsvarianterna testas i en ordning efter hur ofta de används i PLA:n.

4.4.2 Integreringstest

Detta test behandlar interaktionen mellan de olika enheterna. Enheterna bör redan vara enhetstestade i detta stadium för att maximera testets betydelse. Processen syftar till att sammanfoga en antal enheter och testa deras helhetsbeteende. Problemet med att genomföra detta test i en PLA är att denna till skillnad från mjukvaruarkitektur(MA) inte innehåller någon enhetlig information om hur komponenter och dess relationer ska läggas till. Istället måste relationer itereras fram för hela PLA:n. När man har tagit fram dessa relationer kan testningen genomföras[15].

4.4.3 Överensstämmelsetest

Överensstämmelsetest är till för att bekräfta att systemet överensstämmer med specifikationen. I en MA har detta använts för att finna skillnader mellan en MA och dess implementation. Vid applicering på PLA uppkommer ett större problem. Hur ska det faktum att en PLA har varianter av vissa komponenter täckas in? För att kunna testa överensstämmelse måste en av dessa strategier väljas:

- Om implementationen överensstämmer med en PLA samtidigt som den överensstämmer med en specifik PA är testet ok.
- Om implementationen stämmer överens med samtliga PA som kan fås ur en PLA uppfyller den kravet.
- En tredje tänkbar strategi är ”tillräcklig nära”. Om alla krav och all funktionalitet finns i implementationen som hör till de enheter som är obligatoriska i PLA:n överensstämmer implementationen.

4.4.4 Regressionstest

Ur en mjukvaruaspekt syftar ett regressionstest till att försöka säkerställa att en nyare, uppgraderad mjukvara inte inför några nya fel till den tidigare testade koden. I en PLA kan regressiontest tillämpas omfattande både under utveckling men även vid underhåll. Vid utveckling kan ett överensstämmelsetest appliceras för att testa regression. Om ett program P1 är överensstämmelsetestad med avseende på en produktarkitektur PA1 kan ett program P2 bli testad om överensstämmelse med avseende på PA2 genom att välja de testfall som är gemensamt för båda produktarkitekturerna. PA1 och PA2 är båda en PA från en given PLA. Underhåll förklaras enklast med ett exempel, det antas att en tillgänglig applikation A ska bli uppgraderad till en nyare applikation med namn A'. För att testa att inga nya fel införts testas $A' : s$ överensstämmelse mot mot sin PA med samma testfall som användes för att verifiera A mot sin PA.

4.4.5 Hur kopplas detta samman med prototypen?

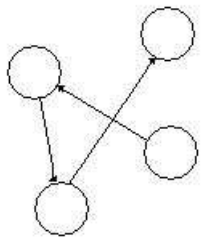
Prototypen som skapats under detta examensarbete har en stark koppling till PLA eller snarare PLA-testning. Samtliga servrar och applikationer på övervakningkartan är enheter. Vid en eventuell övervakning erhåller de olika enheterna på kartan färger i olika kulörer, detta är i sig ett enhetstest som kontrollerar om enheterna fungerar korrekt eller ej. I monitoreringsläget visas inte bara samtliga enheter utan även beroenden vilket i verkligheten är interaktion mellan dessa. Detta medför att systemet även gör en integrationstestning i monitoreringsläget. Det överliggande kravet på systemet, i detta fall det interna nätverket är en tillgänglighet på en viss procent. Ett överensstämmelsetest på systemet kan gälla att jämföra den verkliga tillgängligheten mot det mål som blivit uppsatt. Den verkliga tillgängligheten går att utläsa ur prototypen angett i procent för varje enskild server.

4.5 Arkitektoniska vyer

Arkitektoniska vyer är en representation av arkitekturen som var och en är av vikt för en intressent av systemet[14]. Det är arkitektens uppgift att välja samt konstruera dessa vyer på ett sätt så att alla intressenter förstår och därmed kan bekräfta att systemet kommer att uppfylla deras krav. Vyerna ska dessutom vara en grund för diskussioner samt resonemang mellan intressenterna angående arkitekturen[20]. Eftersom olika intressenter har olika problem samt har fokus på olika delar av systemet kommer multipla vyer att behövas för att beskriva systemet. Ett exempel på en arkitektur att representera med vyer är en vanlig byggnad. Denna har en planlösning, en skiss på den yttre fasaden, dokument som beskriver vattenledningar samt elkablar. Samtliga dessa är väldigt olika men är inte alls oberoende av varandra utan är starkt kopplade och kräver en noggrann koordinering för att resultatet ska bli lyckat. Utifrån detta exempel är det lätt att övertygas om att en vy innehållandes samtliga delar skulle bli väldigt svår att läsa alternativt omöjlig att producera. Det finns dock en skillnad mellan arkitekturen och representationen och den är att representationen är en abstraktion av arkitekturen. För att skapa vyer utgår arkitekten vanligast från en *viewpoint*. Denna definierar hur en vy skapas samt används. Viewpoints är generiska vilket innebär att de kan sparas och återanvändas. Detta gäller dock inte vyer utan de applicerar enbart på sig själv. För att förklara detta påstående kan ett elschema användas som exempel. Viewpoint beskriver hur elschemat ska skapas och denna mall kan återanvändas men det specifika elschemat kan inte appliceras på ett annat hus. Intressenter som arkitekten bör ta hänsyn till vid skapande av vyer och viewpoints är användare, administratörer, köpare och systemutvecklare. Enligt standarden IEEE 1471 är en viewpoint specificerad av: Ett namn, vilken intressent(stakeholder) en viewpoint berör, vilka intressen denna intressent har(Concerns), modelleringsteknik alt. språket för att beskriva vyn samt en eventuell källa eller litteratur[18]. Se tabell 4.1 samt figur 4.6.

Viewpoint element	Beskrivning
Stakeholders	Systemutvecklare
Concerns	Visa nätverkets uppbyggnad
Modellering	Cirkeldiagram, där cirklar är servrar och pilar emellan som visar hur servrarna är ihopkopplade samt hur dessa är riktade.

Tabell 4.1



Figur 4.6: En vy utifrån definierad viewpoint.

Det finns flera standardiserade sätt att välja vyer. Två av av dessa är 4+1 view vilket är en del i RUP[2] samt Enterprise architecture view.

4.5.1 Enterprise architecture view

Vid tillämpning av EA-views fokuseras det på verksamheten och detta återspeglas därför i vyerna. EA-view presenterar fyra olika intressenters intresse, med andra ord, fyra vyer. Det är viktigt att inse är att dessa vyer inte är några fysiska lager utan en konceptuell bild över arkitekturen sett ur olika synvinklar[24].

Verksamhetsvy

Denna vy beskriver arkitekturen utifrån verksamhetsstrategier, organisationsstruktur och de viktigaste verksamhetsprocesserna. Denna vy beskriver även hur företagets processer har sett ut, hur de ser ut i nuläget samt en framtidsplan om hur de ska se ut för att vara konkurrenskraftiga på den marknad verksamheten agerar. Ett praktiskt exempel är en process vilken behandlar ett köp av en bandvagn.

Informationsvy

Denna vy visualiserar samtlig information i verksamheten, beskrivet hur den är kopplad till verksamheten. Exempel på sådan information är vilka delar en bandvagn innehåller.

[2]Rational Unified Process

Applikationsvy

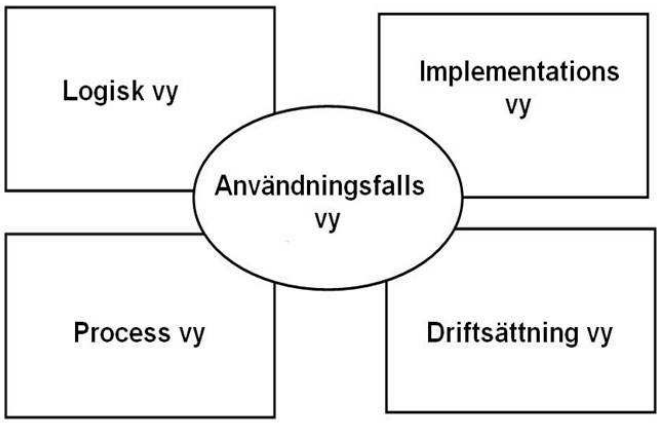
Denna vy beskriver de funktionstjänster som finns. Funktionstjänsterna arbetar ofta direkt mot informationsvyn för att erhålla information. Applikationer som skapas är uppbyggda av ett antal oberoende funktionstjänster vilka kommer att ge applikationer dess beteende. Om samtlig data från denna vy hämtas erhålls informationsvyn.

Infrastruktur

För att ett IT-system ska fungera är vissa anläggningar nödvändiga. Dessa anläggningar beskrivs i denna vy, typisk infrastrukturtjänster är namnuppslagningar och backup. Denna vy innefattar även fysisk placering samt logisk placering av hårdvara.

4.5.2 4+1 view

The Rational Unified Process rekommenderar fem olika vyer i en 4+1 struktur[20]. Eftersom vyerna fokuserar på processen är dessa vyer processtyrda.



Figur 4.7: 4+1 vymodellen över arkitektur.

Logiska vy

I denna vy hanteras systemets funktionella krav. Detta är en abstraktion som identifierar de mest relevanta klasserna samt delsystem. Det är denna vy som ska beskriva vad systemet gör för sina slutanvändare. Exempel är en flygning, färdplan och hamn.

Implementationsvy

Precis som namnet antyder behandlar denna vy implementationen. Det inkluderar källkod, komponenter, exekverbara filer samt hur dessa är organiserade i olika paket, bibliotek och skikt. Andra delar som innefattas är hantering av programvarutillgångar, hur lätt kan komponenter utvecklas samt återanvändas. Ett exempel som passar in i denna vy är källkod för styrningen av en båt.

Processvy

Områden som parallellism, samtidighet, systemstart, systemstopp, feltolerans, dödlägen, svarstider och kapacitet är några av de delar som behandlas under processvyn. Denna vy blir allt mer viktigt desto komplexare systemen tenderar att bli och är dessutom kritisk i realtidssystem där samtidighet är ett nyckelord. Exempel på innehåll i processvyn är beskrivning av olika exekveringstrådar och processer samt hur dessa interagerar med varandra.

Driftsättningsvy

I denna vy möter programvaruutveckling systemutveckling. Frågor som rör driftsättning, installation och prestanda hanteras i denna vy. Den behandlar även hur olika exekverbara komponenter är kopplade till underliggande plattformar. Även den fysiska miljön som mjukvaran exekveras på är även inkluderad i driftsättningsvyn. Till exempel kan det specificeras att styrningen av en båt ska köras på huvuddatorn i båten medan radarprogrammet kan köras på vilken arbetsstation som helst.

Användningsfallsvy

I denna vy finns några fördefinierade nyckelscenarion vilka används vid två olika tillfällen. Initialt används dessa till att identifiera och designa arkitekturen. Dessa scenarion även kallad användningsfall används även vid ett senare tillfälle i utvecklingen för att validera de olika vyerna. Exempel på scenario i denna vy är hur ett samtal rings på en mobiltelefon.

Applicerat på prototypen

Om dessa olika vyer appliceras på den konstruerade prototypen är det två vyer är intressanta. Dessa två är implementationsvyn samt driftsättningsvyn. En övervakning av nätverket i prototypen som monitorerar applikationer kan anses vara en implementationsvy där varje applikation som övervakas är en komponent. Eftersom driftsättningsvyn både hanterar hårdvara och prestanda på hårdvara ger prototypen två delar i denna vy. Dels täcker prototypen in hur exekverbara komponenter är kopplade till underliggande plattformar eftersom varje server kan visualisera vilka komponenter som körs på sig själv. Rörande prestanda kan varje server konfigureras till att enbart godkänna vissa specifika nivåer på ledigt ramminne samt processorutnyttjande. Likaväl som att detta kan vara ett test för att erhålla information om en server eventuellt håller på att gå ner kan det under en längre monitorering anses vara ett test som meddelar hur god prestanda en viss server kan leverera och därmed hur mycket kapacitet som finns att tillgå på en viss server. Prototypens driftsättningsvy kan ha skapats ur följande viewpoint:

Viewpoint element	Beskrivning
Stakeholders	Support samt extern hjälp
Concerns	Visa nätverkets servrars status samt beroenden till andra servrar.
Modellering	Cirkeldiagram, där cirklar är servrar och pilar som visar hur servrarna är beroende av varandra. Servrarna ska vara färgade beroende på status. Grön betyder att servern fungerar korrekt och gult att servern har ett beroende som felar. En röd server indikerar att en server är felaktig.

Kapitel 5

Tjänstebaserad arkitektur

Serviceorienterad arkitektur, hädanefter refererat SOA, är en arkitektur som har växt fram med tiden. Det är alltså ingen revolution det handlar om utan snarare en evolution. Med ordet tjänst menas en upprepningsbar uppgift. Exempel på detta är att kontrollera en persons kundnummer i ett register.
SOA finns överallt[17], när någon använder en CD-spelare stoppas en skiva i CD-spelaren och denna spelare gör att ljud strömmar ut ur någon högtalare, dvs. CD-spelaren erbjuder en tjänst.

5.1 Historia

Ett stort steg i programmeringsvärlden togs när det blev möjligt att att programmera funktionsorienterat. De gamla subrutinerna kunde ersättas med funktioner och procedurer. Den så kallade "spagettikoden" där GoTo och GoSub användes kunde undvikas och istället kunde funktionsanrop användas. Detta framsteg medförde även att det blev lönsamt att införa en designfas innan den egentliga programmeringen skulle genomföras. Många designprinciper som växte fram under denna tid används än idag vid design och strukturering av program. Strategierna för strukturerad programmering var dels abstraktion. Det blev möjligt att fokusera på de viktiga delarna och låta de mindre relevanta delarna bli lite negligerade. En annan designprincip, splittring, medförde att ett stort komplext system kunde delas upp i mindre komplicerade delsystem. Den sista viktiga biten, hierarkiskt organisation medförde att samtliga komponenter som tillhör en lösning organiseras i en trädstruktur. Att programmera funktionsorienterat löser dock inte alla problem. Funktioner och procedurer delar fortfarande på data vilket medför att dominoeffekten infinner sig. Kortfattat innebär denna effekt att en ändring av funktion ett kan medföra att data ändras på ett sätt som inte längre passar funktion två. Således kunde en ändring i en funktion(fallen bricka) göra att flera alternativt alla funktioner efteråt misslyckades.
Lösningen till detta problem finns i objektorienterad programmering och dess tre grundprinciper.

- Inkapsling
- Polymorfism
- Arv

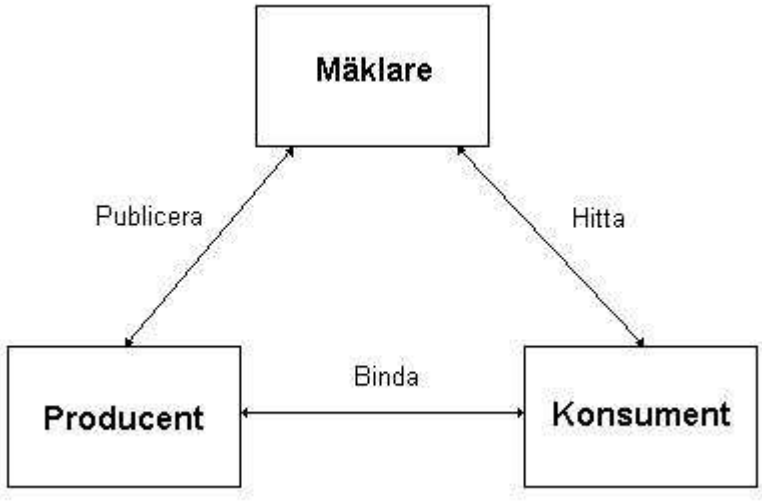
I och med nätverkens framväxt förändrades återigen villkoren för programmerare och nya problem uppdagades. Flera datorer skulle kunna kommunicera med varandra via nätverket. I början användes den enkla arkitektoniska mönstret server-klient. Detta mönster löser dock inte alla problem som kan uppstå i ett nätverk. Ett exempel rörande SQL-servrar är en klient som skickar komplexa frågor till en SQL-server. Komplexa strängar medför avancerad kontroll vilket med detta mönster måste ske i SQL-servern i form av antingen en trigger eller en lagrad procedur. Problemet med dessa kontroller är att dom snabbt blir oöverskådliga samt att dess versionshantering inte är optimerad. Detta föder en önskan om att kunna genomföra kontroll på en annan nivå. Att placera regelverket i klienten var inte tänkbart eftersom att det aldrig skulle kunna garanteras att det körs. Lösningen blev att ha det på serverdelen men på en annan nivå. Regelverken placerades i särskilda applikationsdatorer vilka hade programmerats så att avlusning kunde genomföras samt att en vettig versionshantering kunde genomföras. Detta i kombination med speciella behörigheter vilka gjorde att enbart dessa applikationsdator hade åtkomst till databasen löste de tidigare beskrivna problemen. Åtkomsten till dessa applikationsdatorer skedde via distribuerade objekt exempelvis CORBA och DCOM. Problemet med dessa är att objektåtkomsten krävde en homogen miljö vilket inte är särskilt vanligt i dagens nätverk. För att integrera heterogena system krävdes det därför stora investeringar och ett nytt problem hade fötts.
Enligt Sten Sundblad[27] ligger lösningen på integrationsproblemet i användningen av XML och SOAP. XML är en öppen standard på hur text ska formateras och SOAP är ett XML-baserat protokoll där meddelanden utbyts i en decentraliserad och distribuerad miljö. Genom att enbart skicka textmeddelande med öppna standarder löses problemet med integration. Detta leder fram till SOA. SOA är dock inte är beroende av SOAP eller XML men bygger på samma idéer. Resterande del av detta avsnitt kommer behandla SOA, vilket sammanfattat sägs vara ett sätt att organisera ett IT-system med en samling tjänster som nås via ett meddelandebaserat gränssnitt.

5.2 Arkitektur

SOA definierar tre roller: producent, konsument och mäklare.[27].
`Producent`
Producenter implementerar tjänsten och tillhandahåller tjänsten för konsumenter samt registrerar tjänsten hos mäklaren.
`Mäklare`
Mäklaren tillhandahåller en sökbar katalog av tjänster från olika producenter. Mäklaren tar emot och lagrar tjänster som en producent vill erbjuda samt erbjuder dessa till intresserade konsumenter
`Konsument`
Konsumenten söker efter tjänster hos en mäklare samt gör tjänsteanrop mot angivna ändpunkter hos respektive producent. Denna konsument är antingen en applikation, tjänst eller någon typ av mjukvara som behöver en tjänst.

5.3 Egenskaper i en tjänstebaserad arkitektur

Det är viktigt att inse att webbservices inte är SOA. Webbservices använder sig förvisso av SOAP för kommunikation och är speciellt implementerad för SOA. SOA är inte heller



Figur 5.1: En modell som visualiserar SOA:s tre roller.

någon konkurrent åt objektorienterad(OO) programmering utan snarare står SOA över OO eftersom att en tjänst är uppbyggd av objekt. För att ytterligare förklara SOA presenteras de egenskaper som utmärker en tjänst[27].

5.3.1 Autonom

En av de viktigaste egenskaperna hos en tjänst, även kallad service är att vara autonom. Detta betyder att tjänsten kan utvecklas, underhållas, installeras och vidareutvecklas utan att något beroende till dess konsumenter skadas. Den autonoma egenskapen ställer även krav på att tjänsten tar fullt ansvar för sin egen säkerhet och bör vara misstänksam för att skydda sig mot dels illvilliga användare men även användare som inte är behöriga.

5.3.2 Meddelandebaserat gränssnitt

Ett meddelandebaserat gränssnitt är centralt för en tjänst. Tanken med dessa gränssnitt är att meddelande p.g.a servicens autonoma egenskap inte ska komma i kontakt med någon inre del av tjänsten. Enda sättet att kunna ta del av en tjänsts inre del är att skicka ett meddelande till servicen och be att få visst uppdrag utfört. En annan viktig detalj är att ett gränssnitt som har exponerats inte får ändras så att dess konsumenter drabbas. När en tjänst väl har exponerats ägs den inte längre av sig själv utan utgör ett kontrakt mellan sig själv och alla konsumenter som använder den. Det som ägs av en tjänst är dess innehåll och det får ändras om det inte påverkar dess egenskaper.

5.3.3 Inkapsling av data

I en tjänst ska inte data vara tillgängligt direkt för någon konsument utan det ska vara inkapslat. Den enda möjligheten för en konsument att komma åt datan är att skicka ett meddelande där en viss metod anropas och vänta på den returnerade datan. Detta

liknar objektorientering där varje objekt vanligen har sin data inkapslad på primärt minne. Den stora skillnaden mellan objekt och tjänster är att tjänster kapslar in data på persistent minne, vanligtvis en databas.

5.3.4 Tydligt avgränsad från och löst kopplat till omvärlden

En annan viktig princip för tjänsteorientering är den tydliga avgränsningen mellan tjänst och dess omvärld. Detta innebär bland annat att en koppling mellan konsumenten och dess tjänst är kortvarig och flyktig. Den förstörs när en service har svarat på mottaget meddelande och ett ny koppling får skapas vid nästa meddelande. Motsatsen råder inom objektorientering då konsumenter kan ”koppla” upp sig mot ett objekt i en koppling som är fast och också ofta långvarig.

5.3.5 Delar schema inte klasser med sina konsumenter.

Eftersom en konsument och en service kan ha olika objektmodeller kan kommunikationen mellan dessa inte ske med objekt utan de måste kommunicera med meddelanden. P.g.a detta ska en tjänst inte publicera sina objekt eller delar av dessa utan enbart de scheman som reglerar strukturen på meddelanden som ska utbytas.

5.4 Typer av tjänster

En SOA kan innehålla tre olika sorters tjänster[28].

5.4.1 Entitetstjänster

Denna tjänst är grundläggande och en av byggstenarna i SOA. Den tillhandahåller och skyddar den data som verksamheten är beroende av. Kännetecken för dessa tjänster är att de är stabila och behöver därför inte ändras speciellt ofta. För att skapa en SOA är detta den enda typen av tjänst som krävs. Denna typen av tjänst är den lättaste att återanvända vilket enligt Paul Clements[8] är en viktig grund för arkitekturer. Denna återanvändning gäller om än i mindre mån även aktivitetstjänsterna.

5.4.2 Aktivitetstjänster

Strax ovanför entitetstjänsterna finns lagret med aktivitetstjänster, dessa sänder ut samt hämtar information från entitetstjänsterna.

5.4.3 Processtjänster

Dessa tjänster har två egenskaper, den ena är att dom är så kallad ”long-running” vilket betyder att de kan behålla tillstånd under en tid och fortsätta arbeta med detta tillstånd senare. Den andra är att de är flyktiga, de lever i ett skikt där förändring kan vara ett starkt konkurrensmedel eftersom de ofta kan ändras för att få högre effektivitet. Dessa ändringar sker dock utan påverkan av underliggande skikt.

5.4.4 Presentationstjänster

Dessa tjänster, eller applikationer som de i många fall är, är ingen egentlig tjänst utan mer ett gränssnitt mellan användare och processtjänst.

5.5 Olika typ av data i en entitetstjänst

I de grundläggande entitetstjänsterna kan data klassificeras olika beroende på olika egenskaper som rör den, denna datauppdelning består av resursdata, aktivitetsdata samt referensdata[29].

5.5.1 Resursdata

Resursdata är data som ofta efterfrågas av konsumenter och därför kan flera tjänsters logiska delar ha tillgång till den samtidigt. Detta medför att denna data måste delas ut på ett kontrollerat sätt. Eftersom att olika tjänster kan vilja skriva och läsa samtidigt bör ändringstransaktioner utformas så att en transaktion inte förstör en annan dito[28]. Typexemplet på resursdata är ett banksaldo.

5.5.2 Aktivitetdata

Aktivitetsdata är inte olik resursdata. Den utsätts för förändringar under sin livstid men det finns två skillnader, dels har aktivitetsdata en levnadstid som sträcker sig över en aktivitet. Efter en avslutad aktivitet kan denna data inte bli ändrad utan bidrar därefter till historik alternativt statistik. Den andra skillnaden är att konkurrensen att komma åt denna data inte är lika hög som i resursdata. Ett exempel kan vara en kundorder som är förändringsbar under tiden ordern är aktiv men efter avslutad order kan den inte längre modifieras.

5.5.3 Referensdata

Till skillnad från de två tidigare variationerna äger tjänsten inte denna data utan har bara tillgång till den för läsning. Det kan röra sig om en tjänst A som publicerar en prislista till tjänsten B som ofta använder sig av den. Katalogen är referensdata för B. En publicering kan göras på flera olika sätt, antingen kan prenumeranten, i exemplet ovan tjänst B, begära katalogen. Den andra varianten är att tjänst A trycker ut katalogen, antingen till förbestämda prenumeranter (multicast), en specifik mottagare (unicast) eller till samtliga tjänster (broadcast)[1].

5.6 Först utsidan, sen insidan

Sten Sundblad menar att en verksamhet ser alla system från utsidan medan utvecklarna ofta tenderar att se dem från insidan, vilket återspeglas i exponeringen då den ofta har en utgångspunkt från de interna strukturerna[28]. Tvärtemot detta förespråkar SOA en angreppsvinkel där utsidan rörande vilka meddelanden och vilket jobb som ska utföras av tjänsten specificeras innan någon tanke läggs på tankar rörande implementation. Pat Hellands något hårddragna citat: ”A service is it’s messages. It’s important to see it at first, as a black box” beskriver väl detta. Det ställs också krav på att ett ramverk ska finnas eftersom att SOA handlar inte om att strukturera en applikation utan att strukturera en organisations IT-stöd.

Kapitel 6

Arbetets gång

6.1 Planering

Planeringen bestod av fyra större faser:

- Fas ett: Den givna specifikationen ska färdigställas. Dessutom ska fördjupningsstudien genomföras.
- Fas två: Skapa systemets arkitektur. Den stora delen i denna fas var att designa databasen som skulle vara flexibel, dynamisk och intelligent.
- Fas tre: Implementera prototypen enligt specifikationen. Därefter intervjua framtida användare av systemet för att få feedback och slutligen genomföra förändringar om nödvändigt.
- Fas fyra: Skriva rapport, buggfixa och slutligen integrera systemet i nätverket.

6.2 Verkligt genomförande

Eftersom specifikationen inte initialt blev godkänd kunde inte fördjupningsstudien påbörjas i fas ett. Vidare förändrades specifikationen grovt både rent applikationsmässigt men även rörande fördjupningsstudien efter några veckors arbete. Detta gjorde att planeringen föll redan vid den första fasen. Det enda som kunde genomföras var att bekanta sig med .NET. Den andra fasen gick att genomföra enligt schema än om det innebar en viss omdesign när specifikationen ändrades. I fas tre uppstod det problem med intervjuerna. Ingen på supporten hade använt något liknande system och kunde därmed inte ge några konkreta förslag på utbyggnationer eller ändringar. Fas fyra genomfördes enligt planeringen dock med undantaget att prototypen inte integrerades i systemet eftersom Land System Hägglunds ville avvakta med det.

Kapitel 7

Implementation

7.1 Verktyg

De tre olika delarna användargränssnitt, webbservice och tjänst är samtliga programmerade i miljön *Microsoft Visual Studio .NET 2003* och samtlig kod är skriven i *C#*. Databasen är en Microsoft SQL Server. För att skapa tabeller, procedurer och triggers har Enterprise version Manager 8.0 använts. Vid modellering av olika sql-frågor har dess Query Analyser använts.
För att generera graferna över nätverket i gränssnittet har Graphwiz[1] används. Detta är ett program som kräver en genererad textfil och utifrån denna kan dels en bild men även en imagemap genereras, båda dessa är tillämpade i min applikation.

7.2 Modellen

För att få en förståelse för prototypen visas nedan en schematisk bild för hur en typisk inititering mellan monitorerad server samt webbservice ser ut:

[1] http://www.research.att.com/sw/tools/graphviz/

Installera tjänsten med hjälp av .msi filen på maskinen, starta tjänsten Tjänsten skickar ett anrop till webbservicen	
↘	
	Webbservicen tar emot anropet och kontrollerar om denna server finns sen tidigare. Om inte läggs den till i databasen. Därefter genereras konfigurationsfilen och returneras tillbaka till tjänsten.
↙	
Tjänsten tar emot datan, ställer eventuellt om sin uppdateringstid. Därefter returneras den begärda informationen till webbservicen.	
↘	
	Den inkommande datan tas emot och lagras i databasen.

7.3 Databas

Databasen är en central del i denna prototyp, all data som presenteras för en användare finns lagrad i denna. Databasen i sig lagrar bara information och har inte vetskap om nätverket fungerar korrekt eller ej, detta märks enbart i monitoreringsläget. Nedan beskrivs databasen med dess trigger samt de lagrade procedurerna.

7.3.1 Tabeller

color

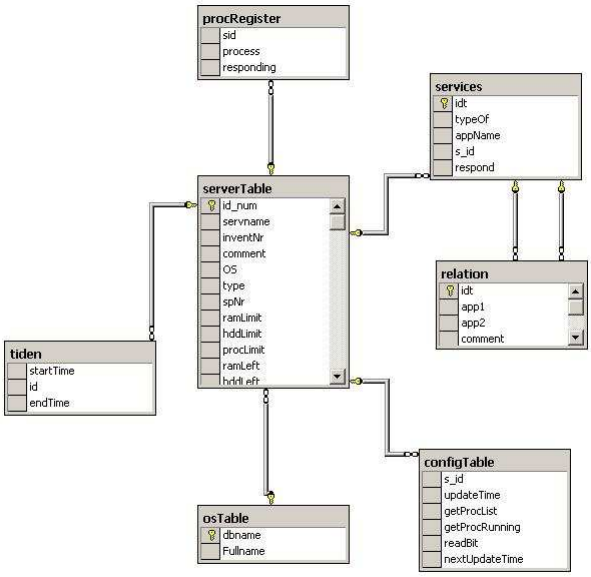
`color` är en hjälptabell för att översätta ett nummer till motsvarande färg, ex. ska siffran 1 översättas till 'green'. Denna färgsträng används senare i genereringen av monitoreringskartan. Denna tabell är till för att en eventuell ändring av färger ska kunna ske på ett enkelt sätt.

colorTable

Innehåller information om vilka applikationer/servrar som ska ritas ut samt deras färgstatus beroende på vilka val som gjorts i gränssnittet.

configTable

Denna tabell innehåller konfigurationer för samtliga registrerade servrar, dvs. hur ofta samt vilken information de ska rapportera in. Vid registrering av en ny server sätts denna konfiguration till standardvärden.
Vid en förändring av uppdateringsintervallet kan inte fältet `configTable` ändras omedelbart eftersom en eventuell minskning av inrapporteringsintervallet skulle kunna ge



Figur 7.1: De vitalaste delarna i databasen

användaren av gränssnittet felaktig information då en fungerande server skulle kunna anses som felaktig utifrån datan i databasen. För att motverka detta har en trigger och en läsbit införts. När en läsning av konfigurationen ska ske uppdateras alltid läsbiten varvid det värde som användaren ville ha som uppdateringsintervall uppdateras innan läsning sker av densamma.

osTable

Denna innehåller ett fullt namn för operativsystemet samt ett förkortat databasnamn. `osTable` är för att motverka felstavningar samt inkonsekventa namnval på operativsystem.

procRegister

I `procRegister` sparas samtliga processer som körs på servrarna. Utifrån dessa inrapporterade processer kan användaren välja ut vilka specifika applikationer som ska monitoreras i prototypen. Dessa rapporteras alltid in initialt och är därefter en valmöjlighet i konfigurationsfilen.

relation

Denna tabell beskriver beroendet mellan olika applikationer. Den innehåller två foreign keys, en till varje applikation som innefattas av beroendet. Dessutom har varje relation ett eget id för att göra sökningar snabbare. Det finns ingen direkt relation mellan två servrar. Relationen mellan servrar tas fram genom att se vilka applikationer som ligger på vilka servrar samt dessa applikationers relationer.

serverTable

`serverTable` är en grundpelare i databasen. Den innehåller information om samtliga servrar som har installerat tjänsten och startat den en gång. Denna tabell innehåller även serverspecifik information rörande vilket operativsystem som är installerat samt vilket service pack en Windowsserver är uppgraderat till. Förutom dessa delar innehåller den gränsvärden på hur låga minimivärdena för ledigt utrymme på hårddisk, ramminne är samt vilket maximalt processorutnyttjande som är tillåtet för varje specifik server, allt för att kunna förutspå en eventuell krasch i systemet. I denna tabell finns också en trigger som registrerar en tid i tabellen `tiden` om det är en ny server som registrerar sig.

services

Samtliga applikationer som är valda för monitorering i systemet finns registrerade i denna tabell. Applikationerna i denna tabell är alltså en delmängd av `procRegister`. Det finns även information om vilken status denna applikation hade vid den senaste inrapporteringen från dess server.

tiden

I denna tabell sparas samtliga tider när servrarna i nätverket har rapporterat in. Denna tabell innehåller även en trigger vilken gör att en inrapportering av en server som är i tid inte kommer att resultera i en ny post utan en uppdatering på den sista posten som denna server har i databasen. Om servern däremot är försenad klassas det som att server har haft en viss nertid och således skapas en ny post i tabellen. Detta innebär att samtliga servrar som haft nertid kommer ha flera poster lagrade i denna tabell. Denna data kan användas för att skapa statistik utifrån användarens önskemål, dock är statistiken väldigt begränsad i prototypen.

7.4 Procedurer

appDig

`appDig` är en rekursiv hjälpfunktion som tar ett applikations-id som inparameter och traverserar sig ner i nätverket till de applikationer som beror på inparametern.

apps

Vid monitorering utifrån en viss specifik applikation är det denna procedur som, utifrån de angivna inparametrarna returnerar den del av nätet som efterfrågats.

appsSchema

Denna procedur tar fram samtliga applikationer med tillhörande färg som finns i nätverket, dessa används senare för att skapa inputfilen för generering av monitoreringskartan. Denna använder sig av hjälpfunktionen `appDig`.

servers

Denna är servrarnas motsvarigheten till `apps`.

servSchema

`servSchema` är motsvarigheten för servrar vad `appsSchema` är för applikationer. Den stora skillnaden under skalet är att denna procedur anropar `appsSchema` och konverterar till det servernivå.

upTime

För att kunna presentera upptiden för varje enskild server används `upTime`. Den tar ett server id som inparameter och summerar samtliga poster i tabellen `tiden`och delar med den totala tiden.

7.5 Webbservice gränssnittet

Webbservicen tillhandahåller två metoder via sitt publika gränssnitt.

7.5.1 XmlNode getConfig()

Denna metod tar inga argument utan hämtar ut konfigurationsdata ur databasen med värdnamnet på datorn som gjorde detta anrop. Anledningen att inte värdnamnet får anges som parameter är ett försök att undvika illvilliga programmerare att få ta i information om andra servrar. Om en icke registrerad server ansöker om sin konfiguration skapas en ny konfiguration med standardvärden. Detta returneras som en XmlNode i formatet som står angiven nedan.

7.5.2 Webbservice → tjänst

Ett exempel på bodyn i xml-filen som skickas från webbservicen till tjänsten kan se ut som nedan.

```
<ConfigFile>
  <UpdateTime>300</UpdateTime>
  <getProcList>False</getProcList>
  <getProcRunning>True</getProcRunning>
</ConfigFile>
```

- UpdateTime anger intervallet inom vilket servern förväntar sig att tjänsten ska rapportera in. Tidsenheten är sekunder.
- getProcList är ett booleskt värde som anger om en tjänst ska rapportera in listan på samtliga körande processer eller inte. Detta används senare till att välja vilka specifika processer som ska monitoreras då inte alla processer på en server är relevanta för monitorering.
- getProcRunning anger om status ska kontrolleras på processerna som är valda för monitorering.

7.5.3 void getResponse(XmlDocument)

Denna metod anropas av tjänsten för att kunna returnera den efterfrågade datan. Svaret kommer i form av ett XmlDocument och inte heller här ges någon möjlighet att ange något värdnamn på avsändare utan det plockas ut ur kontexten.

Tjänst → webbservice

En exempel på bodyn i xml-filen som skickas från tjänsten till webbservicen kan se ut som nedan.

```
<serverStatus>
<ProcName>sql.exe</ProcName>
<Respondings>True</Respondings>
<ProcName>apache.exe</ProcName>
<Respondings>False</Respondings>
<ReQuestedProcName>sql.exe</ReQuestedProcName>
<ReQuestedRespondings>True</ReQuestedRespondings>
<spPack>1</spPack>
<procUsage>10</procUsage>
<hddLeft>29507</hddLeft>
<ramLeft>434</ramLeft>
</serverStatus>
```

- ProcName och Respondings är starkt sammankopplade, dessa kommer i en lista som är parvist sammankopplade, i exemplet ovan visar det på att tjänstens server har två processer där den ena, med namn apache.exe inte svarar. Denna lista skicka om det i konfigurationsfilen anges att en fullständig processlista ska skickas. Om getProcList är satt till falsk ska det inte förekomma några element av ProcName, Respondings.
- ReQuestedProcName samt ReQuestedRespondings rapporterar in status på de processer som är valda för monitorering. Antalet processer kan spänna från noll till godtyckligt antal.
- spPack är ett heltal vilket anger gällande servicepack som är installerat på den returnerande servern. En uppgradering av ett servicepack medför alltså en automatiskt uppdatering av informationen i databasen.
- procUsage innehåller ett heltal som är beräknat utifrån medelutnyttjande av processorn ifrån de sista tre anropen på en viss server.
- hddLeft meddelar hur många MB som finns kvar på hårdisken
- ramLeft meddelar hur många MB ramminne som är ledigt.

Kapitel 8

Sammanfattning

8.1 Säkerhet

Säkerhet är inget som har genomsyrat denna prototyp. Dels för att den är avsedd för ett internt nätverk vilket medför att data inte kan kommas åt utifrån samt att den skickade datan inte är speciellt känslig. Det finns ingen inloggning av användare och således ingen nivåindelning av användare. Detta skulle eventuellt kunna ses som en möjlig påbyggnad där enbart administratörer kan fylla i data och vanliga användare enbart kan monitorera nätverket och därigenom minska möjligheten till felaktigt data i databasen. Eftersom datan inte är speciellt känslig fyller kryptering ingen funktion. Vidare finns ingen autenticering av vilken slags dator som installerar tjänsten vilket gör att det kan vara en arbetsstation som registrerar sig. Detta behöver dock inte ses som begränsning en utan snarare en möjlighet att i framtiden även kunna monitorera arbetsdatorer.

8.2 Problem

Några veckor in på detta arbete korrigerades specifikationen grovt. Samtliga inblandade var dock positiva till denna förändring vilket i slutändan gav en mer användbar applikation samt resulterade i fler svårigheter vilket jag som examensarbetare ville ha. Även fördjupningen ändrades efter någon dags läsande inom ett annat område och redan här vill jag rikta ett speciellt tack till Kalle Hagström som tipsade mig om detta och gjorde att denna rapport blev intressantare att skriva samt förhoppningsvis även intressantare att läsa.

Ett annat tidskrävande problem var utritningen av grafen. Initialt gavs ett försök till att rita på en bitmap men efter hand då grafen växte och fler beroenden infördes blev det för komplext och därför kändes valet av Graphviz[1] naturligt.

8.3 Framtida arbete

– Det mest uppenbara är att implementera tjänsten på andra plattformar, i nuläget finns bara tjänsten som ska installeras på serversidan skriven för Windows. Ef-

[1] www.graphviz.org

tersom kommunikationen sker via SOAP vilket kort beskrivet är XML via HTTP krävs det egentligen bara att det går att öppna en socket på ett operativsystem för att det ska kunna genomföras vilket i sig inte borde vara någon begränsning på en server i ett nätverk.

– Ett schema för hur trafiken mellan tjänsten och webbservice ser ut bör skapas för att öka systemets robusthet.

– För att kontrollera om de monitorerade processerna svarar används ett anrop till operativsystemet. Det kan eventuellt vara önskvärt att kunna kontrollera processer med en anrop från en binärfil.

Kapitel 9

Tack

Utan vissa personer hade detta examensarbete inte gått att genomföra, därför vill jag rikta ett speciellt tack till:
· Gunno Hamberg, min externa handledare vid Land System Hägglunds
· Kalle Hagström
· Bosse Främling
· Sofie Borg
· Lars-Erik Janlert, min interna handledare vid Umeå Universitet.

Referenser

[1] Adobe. Service Oriented Architecture. *http : //www.adobe.com/enterprise/pdfs/Services_Oriented_Architecture_from_Adobe.pdf* (visited 2005-12-07).

[2] Jenny Ang. SOA antipatterns. http://www-128.ibm.com/developerworks/webservices/library/ws-antipatterns (visited 2005-12-02).

[3] Rich Kazman Apul Clements and Mark Klein. *Evaluating a Software Architecture*. Addison Wesley, London, 2001.

[4] Cristobal Baray. The model-view-controller (MVC) design pattern. http://cristobal.baray.com/indiana/projects/mvc[2-6].html (visited 2005-12-04).

[5] G. Booch. Software Arcihtecture. www.booch.com/architecture/ blog/artifacts/Software%20Architecture.ppt (visited 2005-11-19).

[6] Luca Chiarabini. CORBA vs. Web Services. http://www.mathematik.uni-muenchen.de/ chiarabi/middleware.pdf (visited 2005-12-06).

[7] World Wide Web Consortium. Simple Object Access Protocol (SOAP) 1.1 . http://www.w3.org/TR/2000/NOTE-SOAP-20000508/ (visited 2005-12-06).

[8] Jeff Cromwell. The Art and Science of Software Architecture. http://www.ercb.com/ddj/2000/ddj.0006.html (visited 2005-11-15).

[9] Ralph Johnson Erich Gamma, Richard Helm and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison Wesley Professional, London, 1994.

[10] David Garlan. Software architecture: a roadmap. Technical report, Carnegie Mellon University, Pittsburgh, USA, 2000.

[11] David Garlan and Mary Shaw. An introduction to software architecture. Technical report, Carnegie Mellon University, Pittsburgh, USA, 1993.

[12] Aniruddha Gokhale. Reinventing the Wheel? CORBA vs. Web Services. http://wwwconf.ecs.soton.ac.uk/archive/00000216/01/ (visited 2005-12-06).

[13] The Open Group. Architecture Patterns. http://www.opengroup.org/architecture/togaf8-doc/arch/p4/patterns/patterns.htm (visited 2005-12-15).

[14] The Open Group. Developing Architecture Views - Introduction. http://www.opengroup.org/architecture/togaf8-doc/arch/p4/views/vus_intro.htm (visited 2005-11-24).

[15] A. van der Hoek H Muccini. Towards Testing Product Line Architectures. http://www.henrymuccini.com/Research/Tacos03/Tacos03_CR.pdf (visited 2005-12-15).

[16] Dave Hay. Zachman Framework Introduction. http://www.tdan.com/i001fe01.htm (visited 2005-11-24).

[17] Hao He. What is Service-Oriented Architecture? http://webservices.xml.com/pub/a/ws/2003/09/30/soa.html (visited 2005-12-06).

[18] IEEE. IEEE Recommended Practice for Architectural Description of Software-Intensive Systems. http://idenet.bth.se/servlet/download/element/31818/1471-2000.pdf (visited 2005-11-24).

[19] Microsoft Inc. Microsoft Management Home. http://www.microsoft.com/management/default.mspx (visited 2005-12-06).

[20] Philippe Kruchten. *En arkitekturcentrisk process.* Addison-Wesley, London, 2002.

[21] Carla Marques Pereira and Pedro Sousa. A method to define an enterprise architecture using the zachman framework. *ACM Press New York, NY, USA*, pages 1366–1371, 2004.

[22] Rational Unified Process. Concepts: Software Architecture. http://lehre.ike.uni-stuttgart.de/wn/musoft/rupmanual/process/workflow/ana_desi/co_swarch.htm (visited 2005-12-02).

[23] SNMP Research. Simple Network Management Protocol(SNMP). http://www.cisco.com/univercd/cc/td/doc/cisintwk/ito_doc/snmp.htm (visited 2005-11-14).

[24] Pallab Saha. Analyzing the open group architecture framework from the geram perspective. Technical report, National University of Singapore, 25 Heng Mui Keng Terrace, Singapore.

[25] Inc. Sun Microsystems. Patterns. *http : //java.sun.com/blueprints/patterns/* (visited 2006-01-17).

[26] Sten Sundblad. Mjukvaruarkitektur - en översikt. http://download.microsoft.com/download/4/e/c/4ec239bd-3424-4b97-9e40-2286bf33d9b8/Oversikt_arkitektur.pdf (visited 2005-11-14).

[27] Sten Sundblad. Serviceorienterad arkitektur - en översikt. http://download.microsoft.com/download/0/5/6/05632426-99e8-42be-9ffd-aa1c31567cc7/soa_oversikt_0410.pdf (visited 2005-12-06).

[28] Sten Sundblad. Serviceorienterad arkitektur och data. http://download.microsoft.com/download/6/3/1/63127a19-cddf-4fdb-96a8-b1c94f48de8e/SOA_och_data_0411_2.pdf (visited 2005-12-09).

[29] Sten Sundblad. Serviceorienterad arkitektur och processer. http://download.microsoft.com/download/8/d/6/8d6cc1bd-6be8-4710-a6cd-172dc7300141/SOA_och_processer_0412.pdf (visited 2005-12-08).

[30] Mattias Weit and Stephan Herrmann. Model-view-controller and object teams: a perfect match of paradigms. *ACM Press New York, NY, USA*, pages 140–149, 2003.

Bilaga A

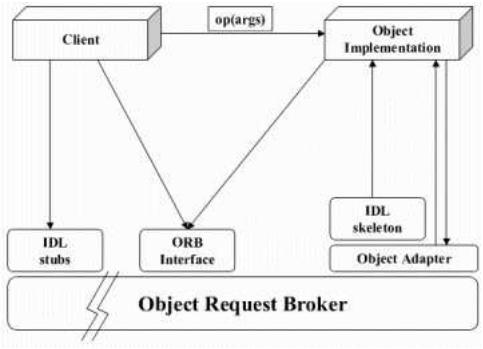
Ordlista

A.1 CORBA

CORBA eller Common Request Broker Architecture är en öppen standard definierad av OMG[1] vilken distribuerar objekt. CORBA är en buss via vilken klienter kan anropa metoder på fjärrobjekt på en server oberoende på vilket programspråk som använts eller vart dessa objekt är lokaliserade[6]. För att kunna kommunicera mellan server och klient finns en ORB[2] närvarande på båda sidorna. Denna kommunikation sker vanligast via protokollet Internet Inter-ORB Protocol(IOOP)[12]. Samtliga CORBA-objekt blir antingen samlade på klienten eller distribuerade på en fjärrserver, detta påverkar dock inte på vare sig implementationen eller på användandet utan ORB:en tar hand om dessa detaljer. För att definiera metoderna samt operationerna i CORBA-objekten på ett enhetligt sätt används IDL[3]. Eventuell felhantering kommer att ske genom IDL-exceptions. En framtagning av en CORBA applikation kan ske enligt följande[12]:

- Definiera upp servicens IDL
- Kompilera IDL för att generera stubbar på klientsidan och skelett på serversidan, dessa ska översätta från ORB till applikation.
- Implementera servicen och associera den med skeletten med hjälp av Object Adapter.
- Publicera tjänsten

[1] Open Management Group
[2] Object Request Broker
[3] Interface Definition Language

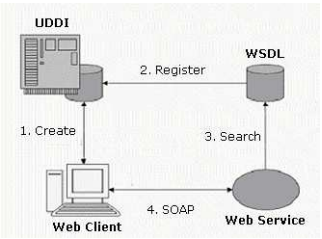


Figur A.1: En översikt på corbaarkitekturen

A.2 Webbservice

En webbservice erbjuder en samling med tjänster, för att kunna erbjuda dessa tjänster används ofta WSDL, ett XML-baserat dokument som beskriver var servern är lokaliserad och vilka tjänster som är tillgängliga samt åtkomsten till dessa. Detta dokument registreras sedan hos en UDDI[4] vilken är en mekanism där olika producenter av webbservices kan exponera sina tjänster. UDDI blir således ett bibliotek med erbjudna tjänster där klienter kan söka. För att interagera med webbservicen används SOAP[5] vilket också är ett XML-baserat protokoll som däremot, trots namnet, inte innehåller några objekt utan enbart text formaterat enligt XML. SOAP består av tre delar[7]:

- SOAP envelope har en väl definierad syntax som beskriver vad som finns i ett meddelande, vem som ska hantera det samt om det är valfritt eller krävs att ett visst meddelande är närvarande.
- Regler för serialisering rörande datautbyte.
- SOAP RPC representation definierar hur RPC och dess svar ska användas.



Figur A.2: Webbservice arkitekturen

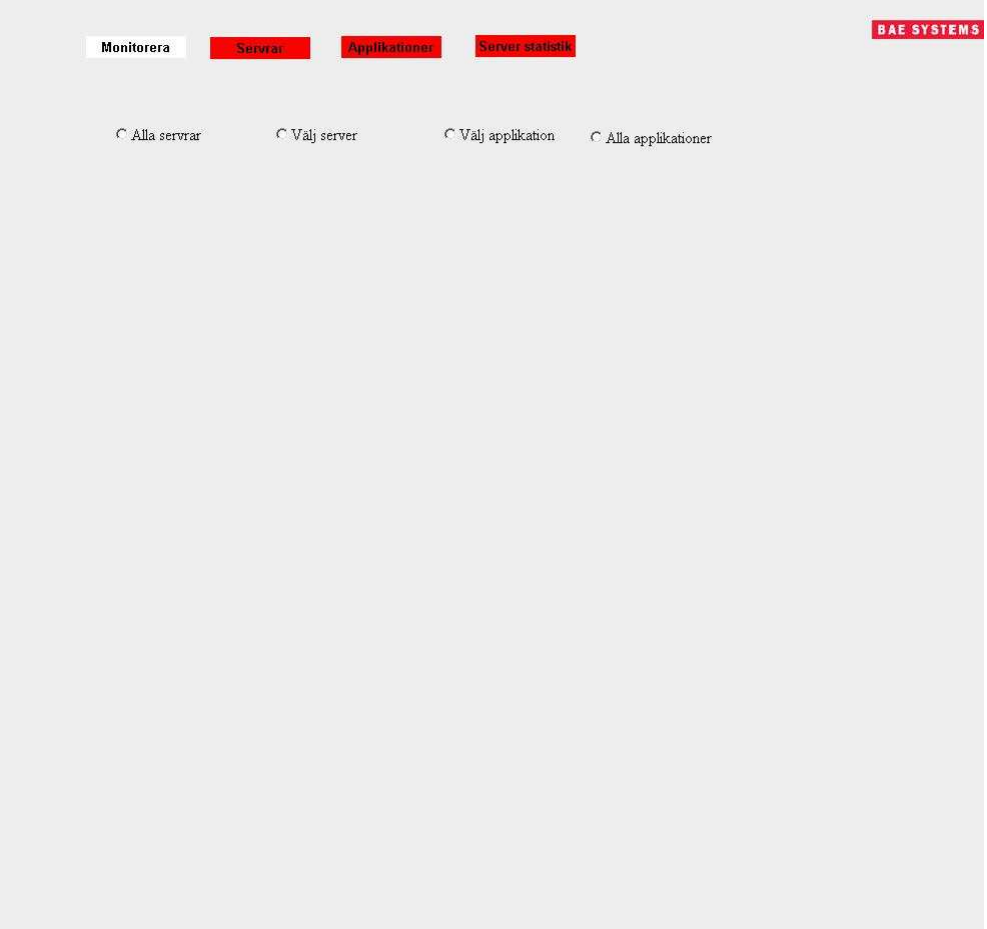
[4]Universal Description, Discovery and Integration
[5]Simple Object Access Protocol

Bilaga B

Användarhandledning

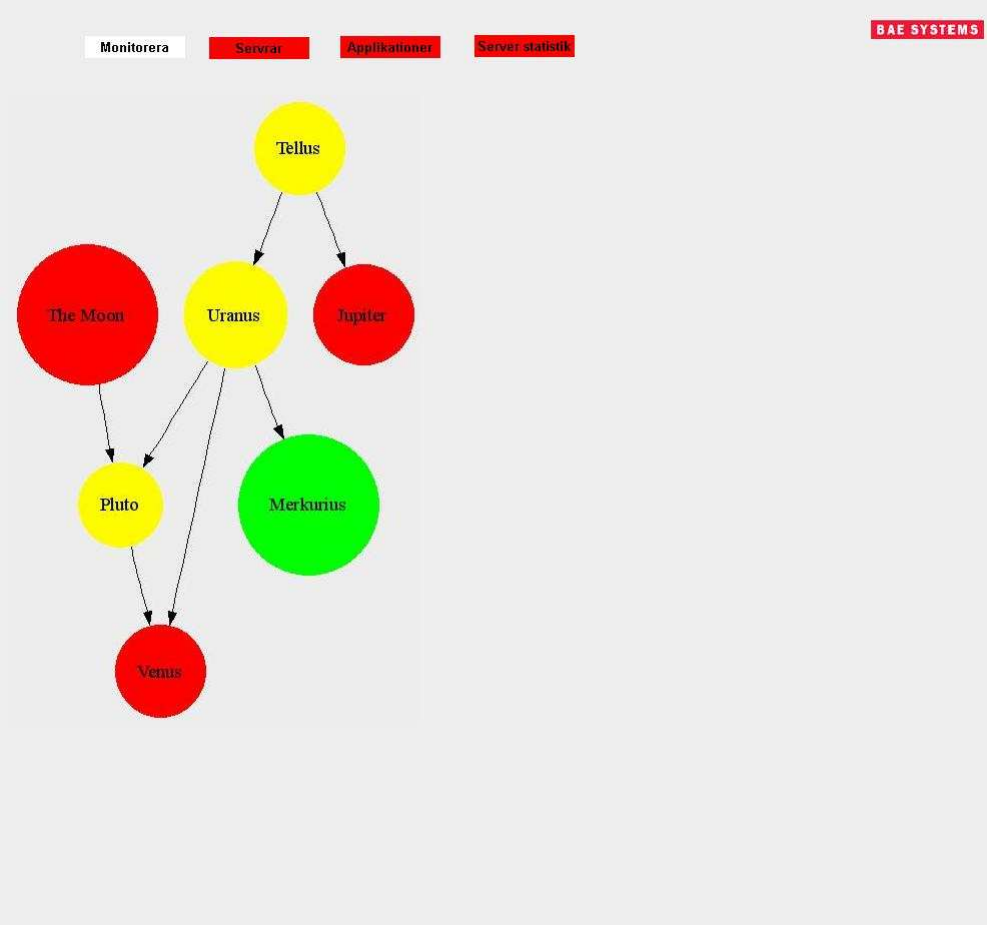
Användarhandledningen kommer enbart att behandla det grafiska användargränssnittet eftersom all interaktion sker där. Det finns fyra större val att göra i gränssnittet och varje val har en egen sektion i denna användarhandledning.

B.1 Monitorering



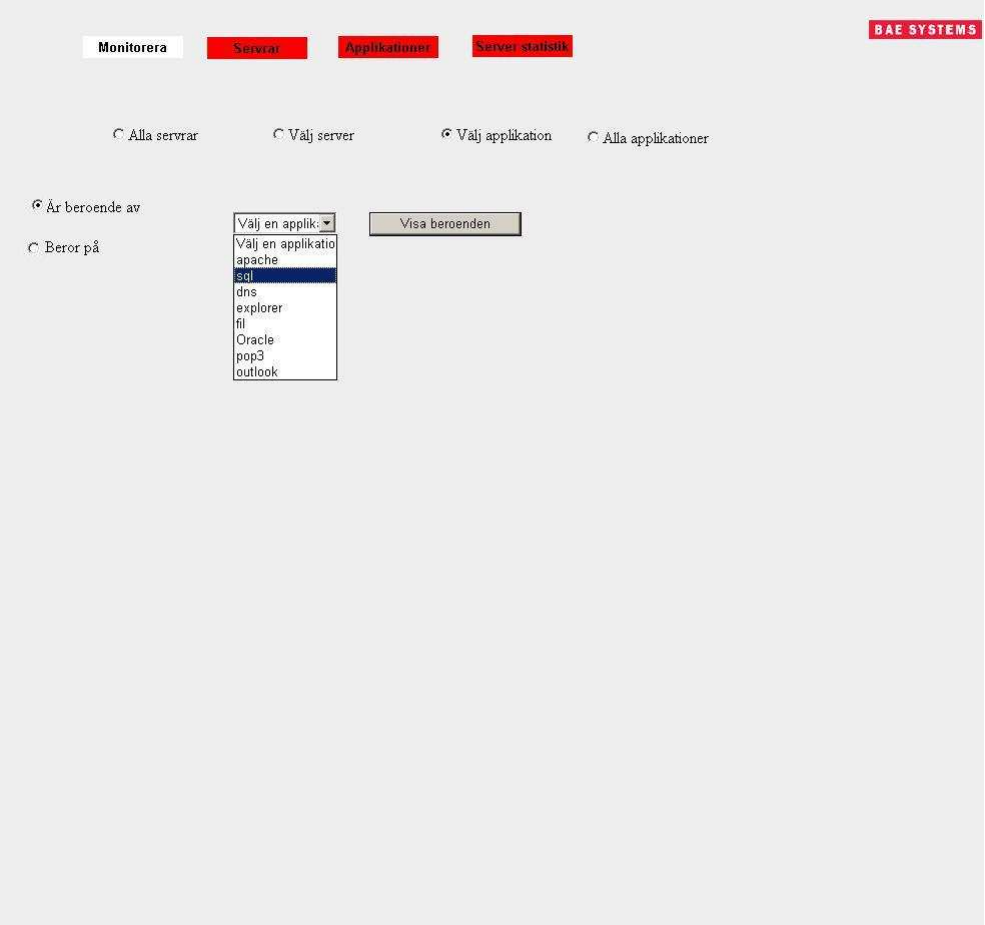
Figur B.1: Grundmeny

Monitoreringsmenyn erbjuder fyra val. Nätverket kan monitoreras utifrån samtliga applikationer eller servrar. De andra möjligheterna som finns är att monitorera utifrån en specifik server alternativt applikation. För att kunna monitorera en applikation måste den vara inlagd för monitorering vilket görs under meny valet ”applikationer”.



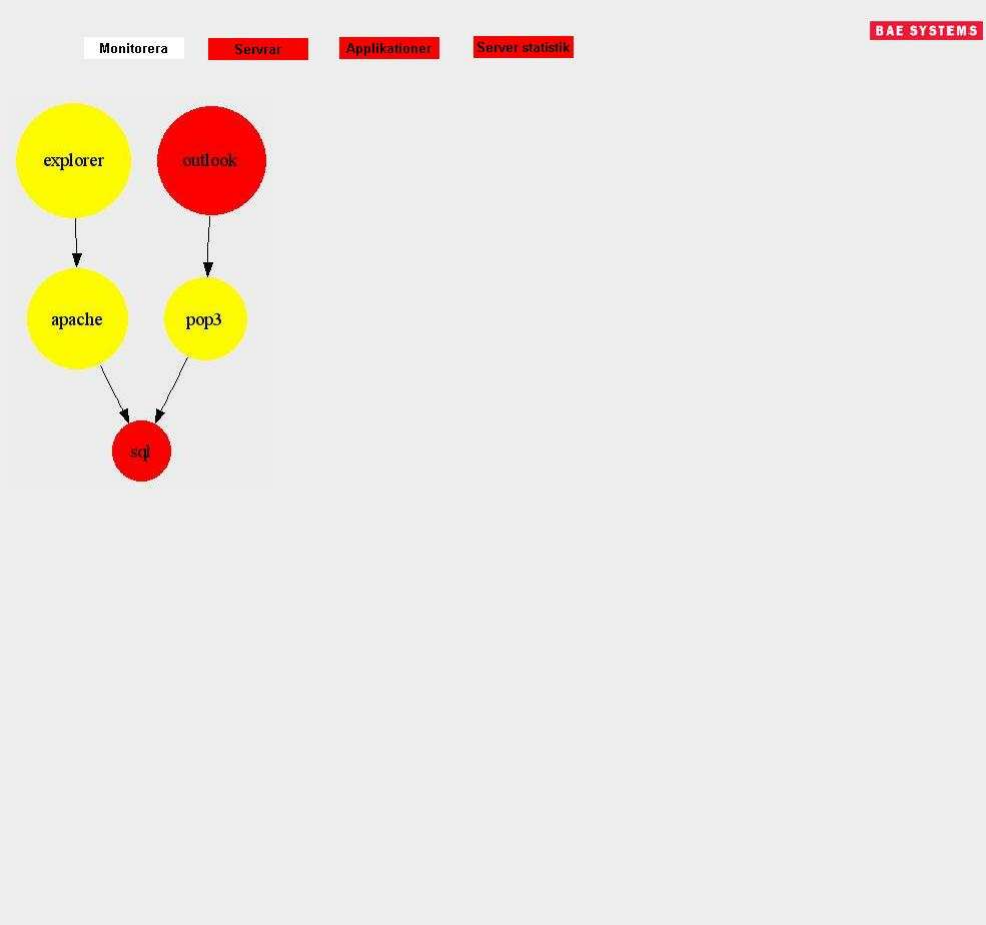
Figur B.2: Samtliga servrar monitoreras

Vid valet att monitorera nätverket utifrån ”Alla servrar” genereras en karta över nätverket. Denna bild är genererad med det senaste datat från databasen och ger en vy där administratörer enkelt kan avgöra om något är felaktigt i nätverket. Pilarna som genereras visar beroenden mellan olika servrar. Samtliga servrar är färgade i en kulör. En grön server betyder att den utifrån dess konfiguration och inrapporteringar anses fungera korrekt. En gul server har en applikation som är beroende av en annan dito som inte fungerar korrekt. En röd server markerar en verklig felkälla alternativt en möjlig felkälla då det inte behöver betyda att den inte svarar utan även att exempelvis nivån på ramminne är lägre än den nivån som är tillåten enligt konfigurationen av servern. Denna vy appliceras på samma sätt vid val av ”Alla applikationer” med skillnaden att applikationerna visas i kartan. Denna vy visar vilka enheter som samtliga ”Beror på”(läs Vid val av specifik applikation för förklaring). För att en server ska finnas med i monitoreringen krävs det att den har minst en applikation som är inlagd för monitorering. Om mer information om servern önskas vänsterklickas denna en gång varvid en popup visualiseras och visar mer information om en viss server. Kartan uppdateras var 60:e sekund.



Figur B.3: Vid val av specifik applikation

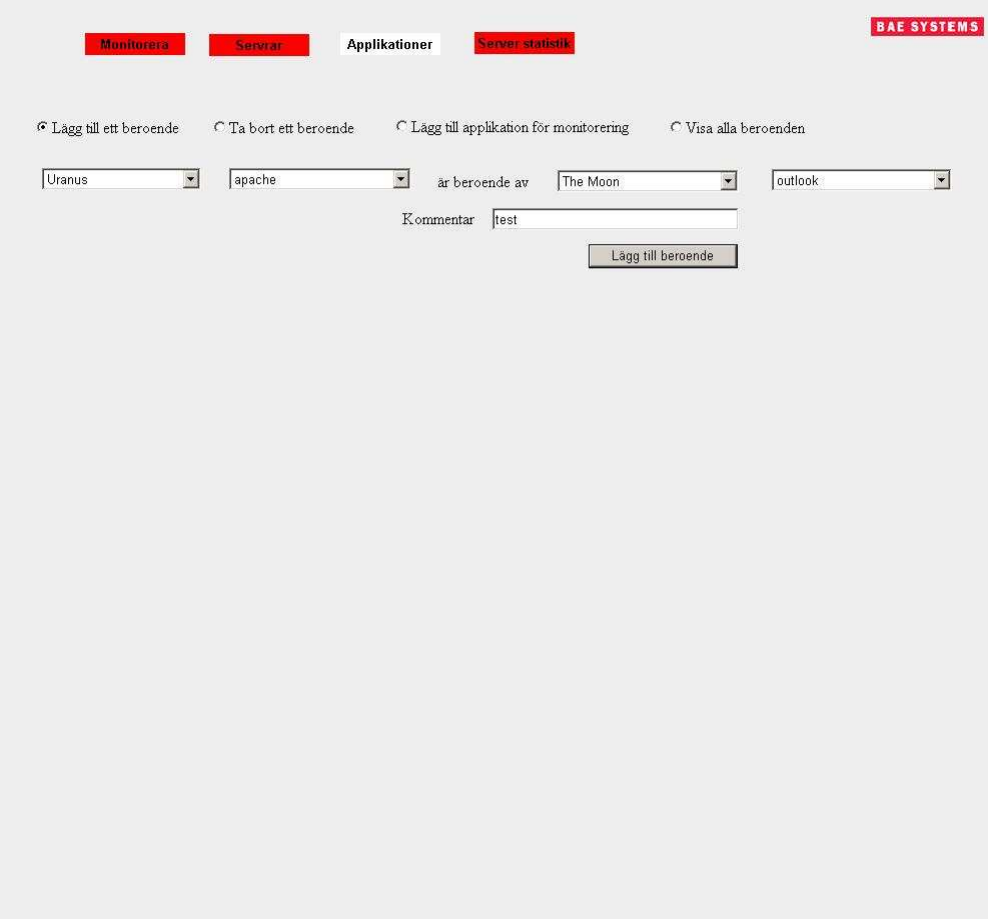
Om en specifik applikation är av intresse kan övrig och för stunden onödig information exkluderas. Vid val av "Välj applikation" specificeras vilken applikation som är av intresse i rullistan. Det andra valet specificerar om monitoreringen ska ske utifrån vilka applikationer den valda är beroende av/beror på. "Beror på" monitorerar specifik applikation samt hela kedjan med applikationer som kan göra att den valda applikationen får problem. Om intresset istället rör vilka applikationer som exempelvis skulle få problem om denna applikation togs bort väljs "Beroende av". Detta appliceras på samma sätt för servrarna i systemet. För att kunna monitorera en applikation måste denna vara inlagd för monitorering, dock ställs inga krav på att den ska ha beroenden.



Figur B.4: Resultat av val av applikation

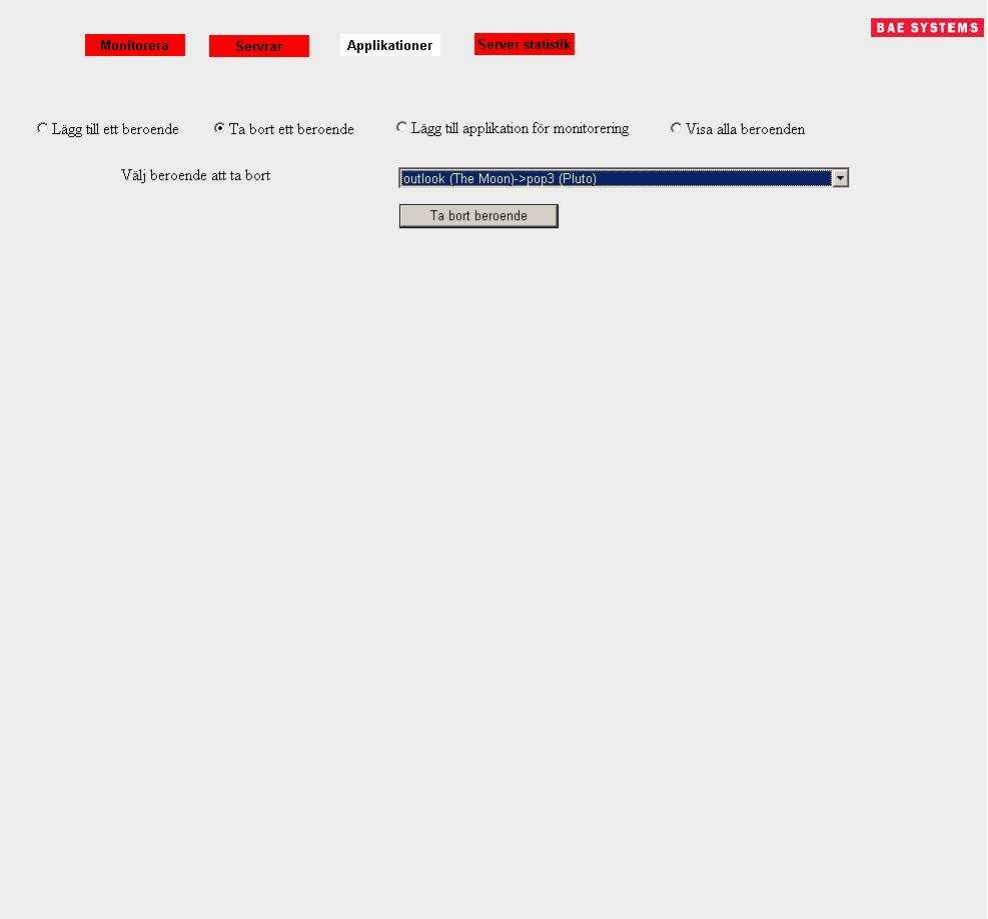
Resultatet av val av specifik applikation. Precis som i monitoreringen av servrar kan användaren klicka på en applikation för mer specifik information. I exemplet ovan kan det utläsas att samtliga applikationer har problem men att om *sql* skulle fixas skulle det i sin tur göra att tre andra(samtliga gula) skulle börja fungera. Dock skulle inte *outlook* fungera eftersom att den är röd. Precis som vid monitorering av hela nätverket uppdateras kartan var 60:e sekund till aktuell status.

B.2 Applikation



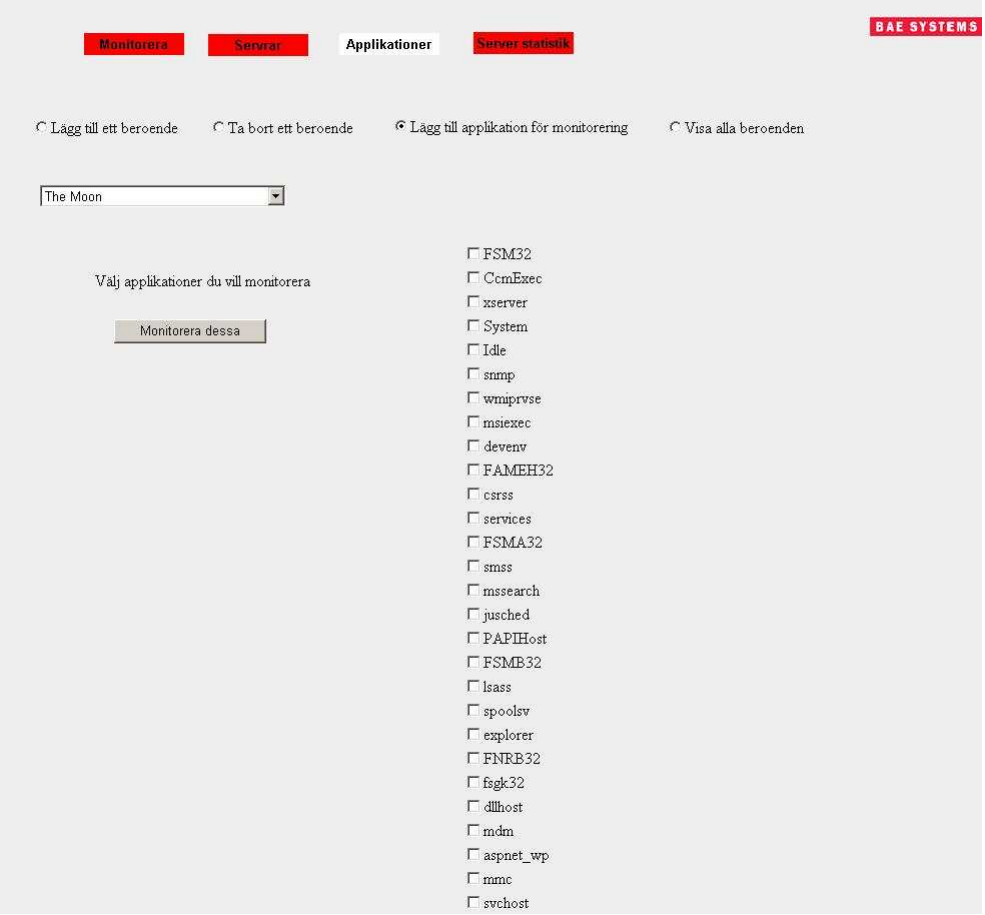
Figur B.5: Lägg till beroende mellan applikationer

För att erhålla en pil i monitoreringskartan måste beroenden läggas till. Detta verkställs genom att välja server samt applikation i de två rullistorna längst till vänster samt välja vilken applikation samt tillhörande server som beroendet är kopplat till. Det är även möjligt att fylla i en kommentar för beroendet för att förtydliga varför. Ett felmeddelande kommer att genereras om beroendet redan finns alternativt om en applikation sätts till att bero på sig själv.



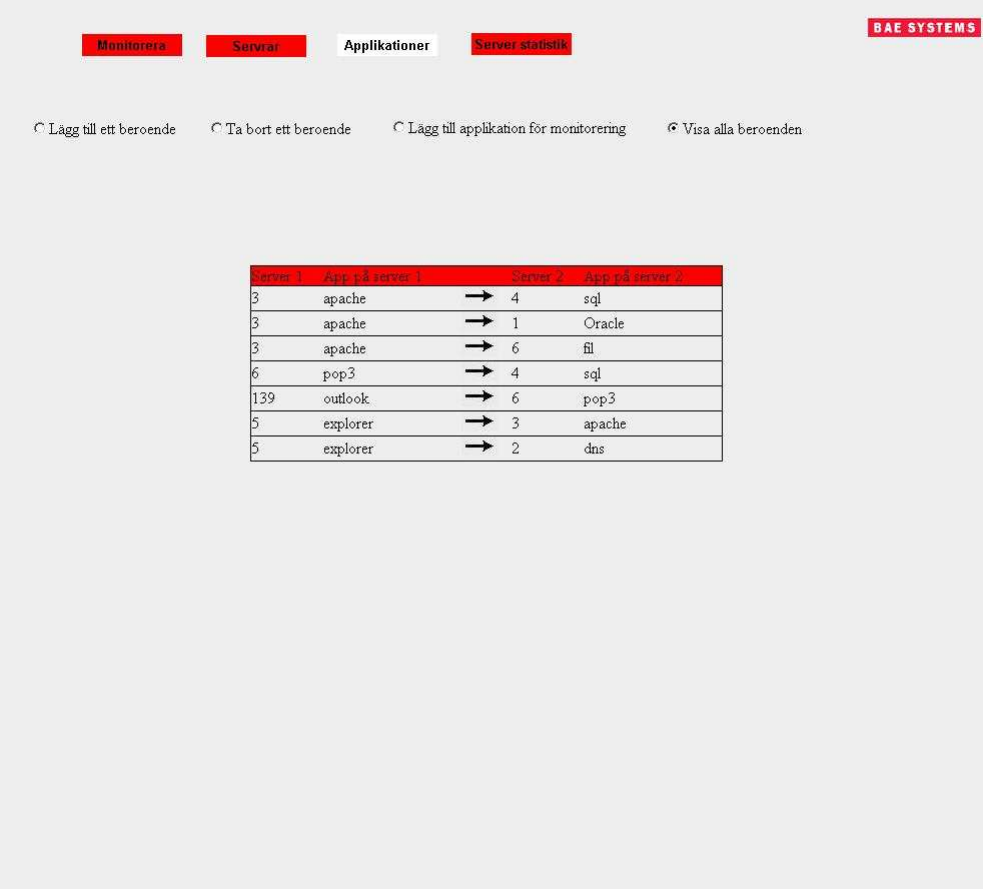
Figur B.6: Ta bort beroenden

För att ta bort ett beroende väljs det önskade beroendet ur listan, bekräfta med att trycka på ”Ta bort beroende”.



Figur B.7: Lägg till applikationer för monitorering

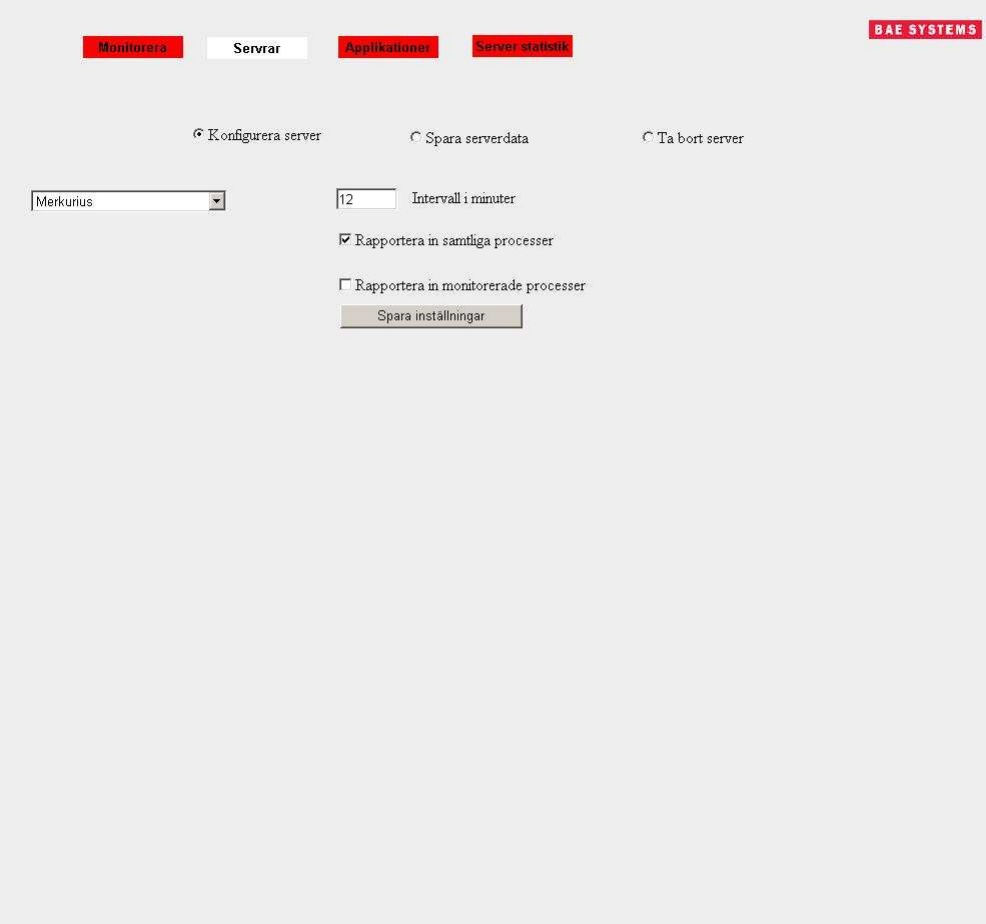
I denna vy läggs applikationer till för monitorering. Först väljs server i rullistan till vänster, därefter markeras i tur och ordning vilka applikationer som är intressanta för systemet. Efter att en applikation lagts till för monitorering kommer den dels att synas i monitoreringsläget samt även finnas tillgänglig för att läggas till i ett beroende.



Figur B.8: Visa samtliga beroenden i tabellformat

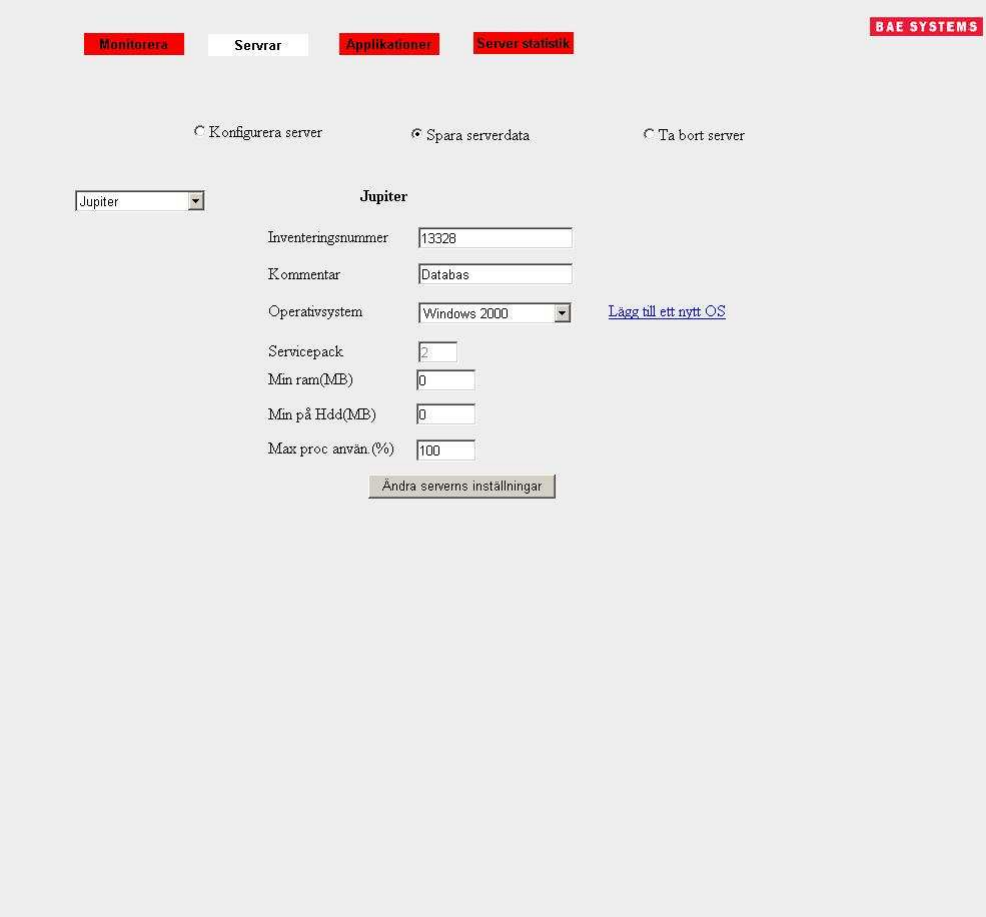
Denna vy listar samtliga beroenden och någon interaktion är inte möjlig.

B.3 Server



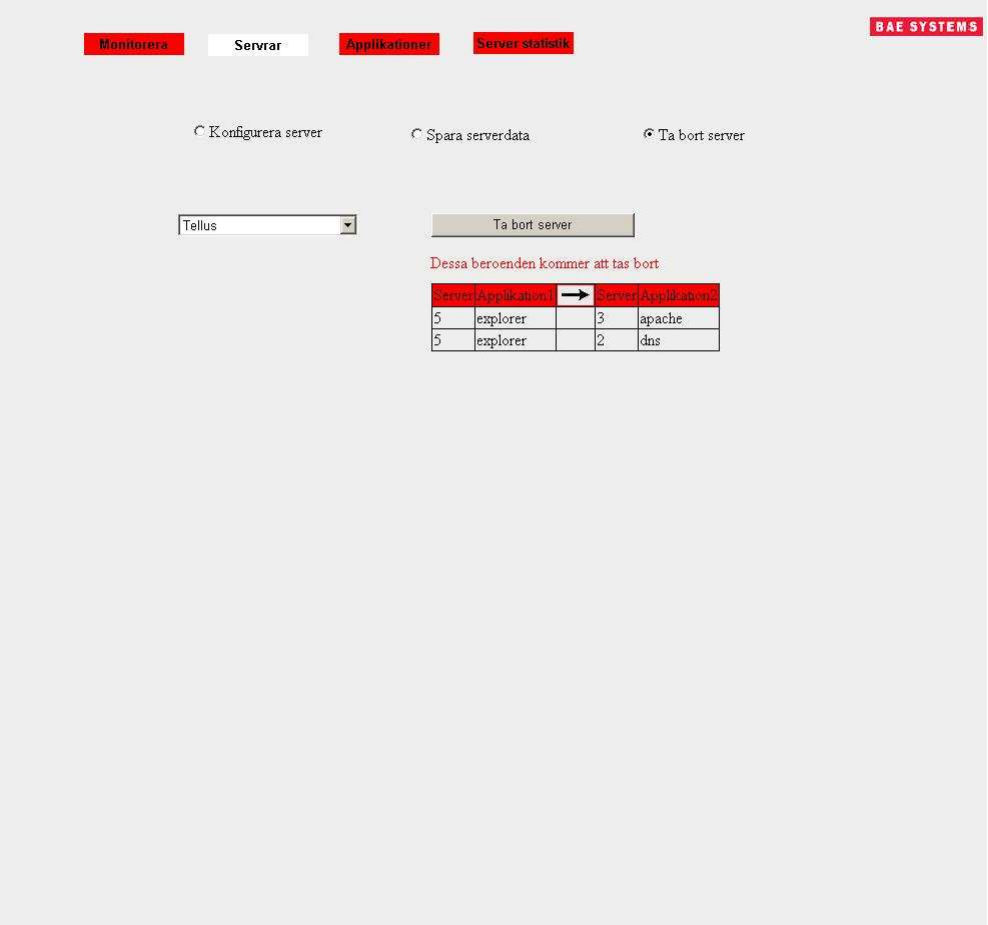
Figur B.9: Visa konfigurationen för en specifik server

Detta val används för att konfigurera med vilket intervall samt vad en server ska rapportera in. En ändring av konfigurationsfilen kommer dock inte att träda i kraft förrän nästa gång en server rapporterar in. Felmeddelande kommer att genereras om uppdateringstiden inte är av numerisk typ.



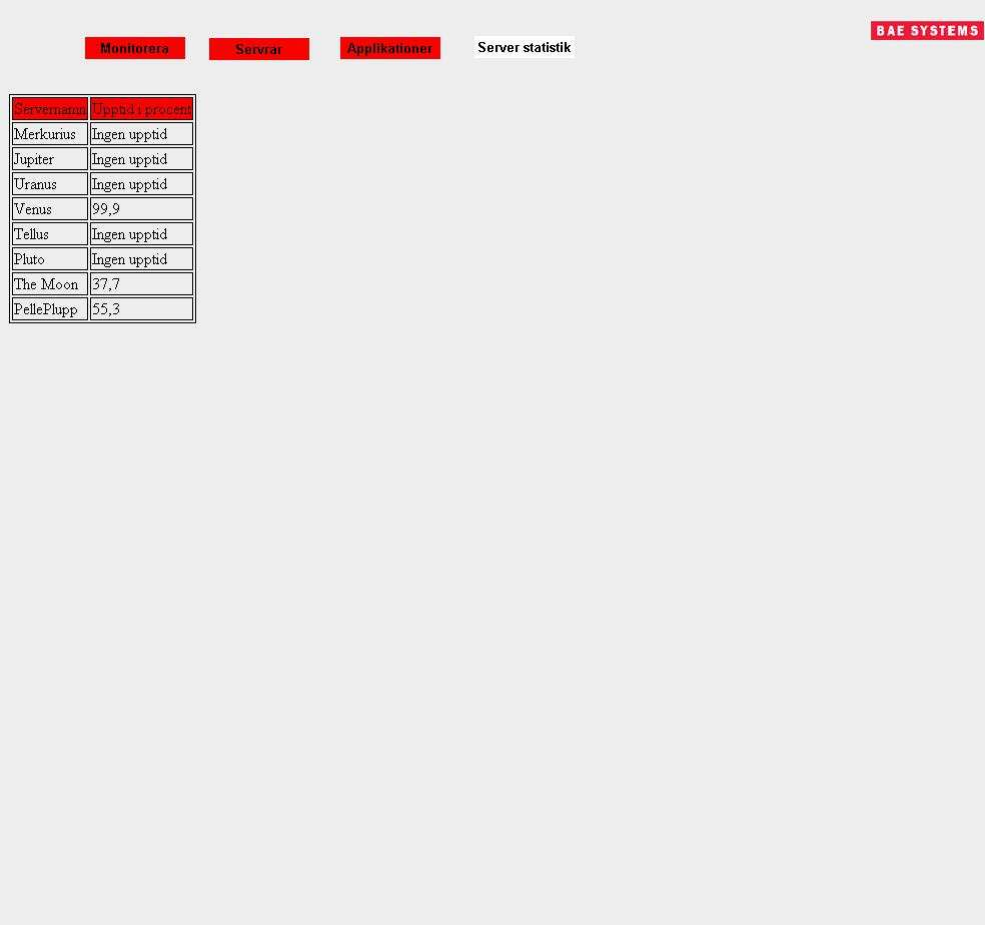
Figur B.10: Visa serverdata för en specifik server

I denna vy sparas serverspecifik data. Likaväl som i konfigurationen kommer förändring på de olika gränsvärden på ramminne etc. inte att träda i kraft förrän nästa gång server rapporterar in till webbservice. Noterbart är att Service Pack numret inte är möjligt att ändra utan det uppdateras per automatik. Om ett önskat operativsystem inte finns med i rullistan måste det först läggas till genom att klicka på ”Lägg till OS” och därefter välja operativsystemet ur listan. Samtliga fält förutom ”Kommentar” ska vara numeriska värden och felmeddelanden kommer att visas om användaren försöker mata in ogiltiga värden.



Figur B.11: Ta bort en server

Denna vy används för att ta bort servrar från systemet. Detta görs genom att först välja server i rullistan till vänster vilket kommer att medföra att de beroenden som denna server är involverad i kommer att visualiseras. För att ta bort den valda server klickas knappen ”Ta bort server” en gång. Detta medför att samtlig data rörande servern kommer att tas bort från systemet. Observera att de beroenden som visas enbart är beroenden som direkt beror på servern, det kan hända att dessa i sig är långa kedjor vilket kan förstöra en stor del i monitoreringen.



Figur B.12: Serverstatistik

Statistiken är i prototypen något begränsad och det går enbart att utläsa vilket upptid de olika servrarna har. Eftersom att applikationen inte kan avgöra i förtid om en server kommer att rapportera in nästa gång kommer upptiden att presenteras i garanterad minimumtid, dvs. basera sig på redan inrapporterade tider.

B.4 Installationsguide

Lösningen kräver fem filer: servService.msi, gui.zip, webservice.zip, db.bak samt Graphviz installationsfil.

B.4.1 Databasen

Databasen ska vara av typen MSSQL och för att få en korrekt databas återställs(läs Restore database) den medföljande databasfilen `db.bak`.

B.4.2 Server

För att installera och därmed lägga till en server till monitorering installeras den exekverbara filen `servService.msi` på vald server. Inga övriga krav finns på servern, däremot måste webbreferensen ändras i tjänsten för att servern ska rapportera in till korrekt server.

B.4.3 Webbservice server

Detta är den gemensamma servern dit samtliga servrar rapporteras in. Ett krav är att `Graphviz`[1] finns installerat i katalogen $C:\backslash program\backslash ATT\backslash Graphviz$. Vidare ska filen `webservice.zip` packas upp och läggas i en katalog under en IIS. Vilken databas samt vilken användare och lösenord som ska som ska användas specificeras i filen Web.config.

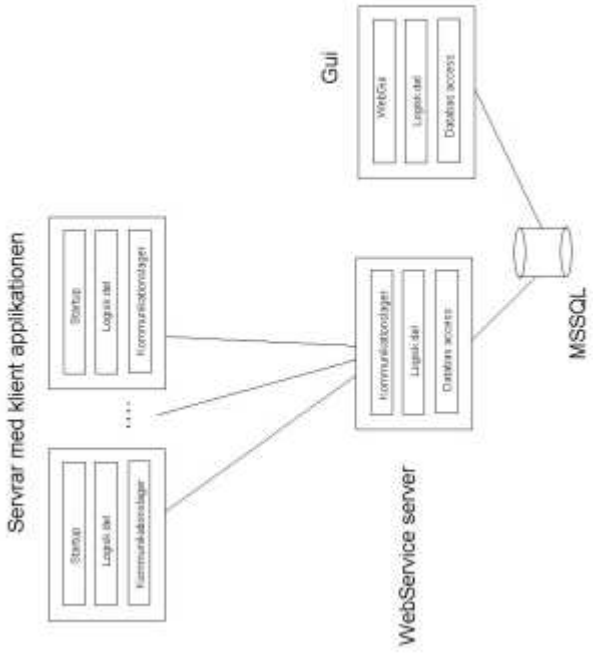
B.4.4 Gui

Likväl som i webbservicen finns anslutningssträngen till databasen i filen `Web.config`. Denna samt det andra begärda filerna finns i filen `gui.zip`, den nedpackade katalogen ska packas upp och läggas på en webbserver.

[1] www.graphviz.org

Bilaga C

Bilaga



Figur C.1: Översikt av systemet