A distributed system for data management
Ett distribuerat system för datahantering

Mikael Andersson
mikan@e.kth.se

27 oktober 2002

Abstract

This thesis describes how to design and implement a distributed data management system. It is required that the system should be independent of operating system and underlying database system. It should be possible to run different parts of a system on different database systems and operating systems and it should also be possible to set up rules for the data flows beetween nodes in the system.

The system is implemented in Java to get operating system independence. To make the implementation independent of the underlying database system an abstraction layer was implemented using XML. The XML layer also makes it easier to send database commands embedded in XML documents between different nodes in the system. The rules for the data flows is realized by creating an access control system. The access control system controls how nodes in the system can interact with each other. It is also possible to decide how updates from other nodes effects the local data.

This thesis identifies the different parts needed and describes them. Parts of the system is also implemented and conclusions are drawn from that experience.

Sammanfattning

Rapporten behandlar förutsättningarna för att skapa ett distribuerat databassystem. Systemet är plattformsoberoende, när det gäller både operativsystem och underliggande databasmotor, för att olika delar av systemet ska kunna köras i skilda miljöer. Krav skall gå att ställa på dataflödet mellan olika delar av systemet

Plattformsoberoendet har uppnåtts genom att göra implementationen i java. För att vara oberoende av den underliggande databasen specificerades ett eget databasgränsnitt. Detta är XML-baserat för att enkelt kunna skickas mellan olika datorer, och för att det då enkelt går att specificera vilka delar som måste eller kan vara med. Det är också enkelt för ett program att tolka informationen i ett XML-dokument. Krav på dataflöden uppnås genom att skapa rättigheter för vilka som har tillåtelse att läsa, få uppdateringar och uppdatera data. Det går också att styra vilka delar av systemet som känner till varandra och hur uppdateringar från andra noder i systemet skall appliceras.

De nödvändiga komponenterna identifieras och beskrivs i rapporten. Delar av systemet är också implementerat och erfarenheter därifrån finns i rapporten.

Innehåll

Kapitel 1

Introduktion

1.1 Bakgrund

Vid detta arbetets början 1999 skapades datakonton på Kungliga Tekniska Högskolan, Royal University of Technology (KTH)[20] vid varje institution utan någon nämnvärd samordning. Detta medförde problem med att bl a att personer slutade utan att deras konton blev avslutade överallt mm. Detta examensarbete var tänkt att se över om ett distribuerat system för datahantering skulle kunna lösa några av dessa problem.

1.1.1 Datorkonton

När detta examensarbete påbörjades skapades datorkonton på KTH oberoende av varandra och det fanns ingen koppling mellan de olika institutionerna. T ex hade ett konto på EIT (fd E) ingen koppling till ett konto på NADA. Det gjorde det svårt att administrera och uppdatera uppgifter som adress, om en person hade slutat, avslutat sina studier osv.

1.2 Syfte

Syftet är att undersöka möjligheterna att distribuera data på ett säkert sätt för att underlätta administration av delvis gemensamma data vid KTH. Datan avser främst person, konto och kortinformation. Systemet ska inte vara beroende av att alla t ex kör samma operativsystem, databasmotor och liknande. Det lokala systemet ska inte heller vara beroende av någon annan del av systemet för att lämna information. Det är acceptabelt att ändringar tar längre tid att spridas ut i systemet om delar av systemet är nere.

1.3 Struktur

Basen är ett grundläggande bibliotek med distributionsfunktioner på vilken det är möjligt att implementera bl a distribuerat konto- och korthanteringssystem. Där det går ska det grundläggande biblioteket baseras på fria programvaror.

Ett behörighetssystem behövs för att på ett säkert sätt hantera data. Systemet ska klara grupper i grupper och olika identifikationssystem. Grupper i grupper innebär t ex att en grupp för ekonomiavdelningen och en annan för teknikavdelningen kan sammanfogas till en ny grupp som innehåller både ekonomi- och teknikavdelningen. Denna nya grupp kan kallas stödavdelning.

Det identifikationssystem som används till att börja med är Kerberos V[1].

Den plattformsoberoende standarden XML används för att distribuera data mellan olika delar av systemet.

Systemet implementeras i java för att redan från början få ett system som kan användas på många plattformar

Följande delar behövs:

- Databas - För att lagra information.
- Användaridentifiering och kryptering - För att säkerställa identitet.
- Grupp och rättigheter - För att se vem som har rätt att göra vad.
- Distribution - För att ändringar ska kunna komma andra tillgodo.

Kapitel 2

Teknisk bakgrund

Detta kapitel är en kort introduktion till de olika tekniker och standarder som har påverkat utformningen av systemet.

2.1 Java

Java[6] är ett objekt-orienterat programmeringsspråk utvecklat av Sun Microsystems (SUN)[5]. Programmen körs i en virtuell maskin och språket har utvecklats för att vara så plattformsoberoende som möjligt. I det här fallet är det plattformsoberoendet som gör språket intressant.

2.2 Agentspråk

Agentspråk är ett försök att standardisera hur agenter ska kunna kommunicera med andra agenter och servrar. Det har utformats för att skapa fristående moduler(agenter) som ska kunna kommunicera med andra moduler även om de är skrivna vid olika tillfällen och använder olika programmeringsspråk eller är direkt implementerade i hårdvara. Genom att skapa en gemensam nämnare för vilka indata och utdata olika operationer kan behöva är det lättare att skapa bryggor mellan olika system.

Nedan följer en genomgång av två olika agentspråk:

2.2.1 KIF

Knowledge Interchange Format (KIF)[3] är ett agentspråk som bl a har tagits fram vid Stanford University, där det har skett en hel del forskning inom agentkommunikations-

området. Det verkar inte hända något nytt på KIF fronten utan det verkar som om det är FIPA som har tagit över.

2.2.2 FIPA

Foundation for Intelligent Physical Agents(FIPA)[4] är en organisation som försöker fortsätta det arbete som startade vid Stanford och de försöker ge ut en specifikation för agentspråk per år. De prioriterar hellre att komma ut med en ny specifikation per år än att åstadkomma en fullständig specifikation. Trots detta finns det en hel del intressant att läsa på deras hemsida.

2.3 XML

eXtensible Markup Language (XML)[7] är en standard från W3C, World Wide Web Consortium. XML är en förenkling av Standard Generalized Markup Language (SGML), och är också besläktat med t ex HyperText Markup Language(HTML). XML är liksom HTML ett taggbaserat språk, det finns starttaggar och sluttagar, t ex:

```
<tagg>Denna text innesluts av tagg taggarna</tagg>
```

Enkelt sagt är XML en standard som används för att skriva specifikationer, där bl a HTML är en sådan standard. En sådan specifikation kallas för Document Type Definition(DTD)

Ett XML-dokument kallas giltigt om det uppfyller specifikationen för dokumentet. Om specifikationen (DTDn) medföljer dokumentet kallas dokumentet för fristående. Att arbeta med giltiga dokument är en stor fördel eftersom programutvecklaren redan vet vilka delar dokumentet kan och får innehålla och kan skriva sin kod därefter.

2.3.1 Gränssnitt

Det finns standardiserade gränssnitt för att validera och läsa information ur XML-dokument.

2.3.1.1 SAX

Simple API for XML (SAX) är den första standarden för att kunna läsa information ur ett XML-dokument. Det finns SAX gränssnitt för flera programmeringsspråk som C, java och perl. SAX baserar sig på registrering av olika funktioner som åberopas för respektive tagg i XML-dokumentet. Det gör att det enkelt går att skapa en egen datamodell där information från XML-dokumentet kan lagras.

2.3.1.2 DOM

Document Object Model(DOM)[8] är standardiserad modell av W3C för hur ett XML-dokument kan symboliseras som objekt i t ex java. Den modell som skapas är generell, innehåller mycket överflödig information och är svår att hämta information från. Detta innebär att en ny datamodell behöver skapas där information är mer lättillgänglig.

Det finns ett antal olika företag och organisationer som har implementerat program som följer DOM standarden. Två av dem är Oracle och IBM. Både Oracle och IBM hade licenser som gjorde att de var svåra att använda. Bl a hade IBM en paragraf om att man kunde bli tvungen att sluta använda programvaran med en månads varsel. Som tur var så "gav" IBM bort sin programvara till Apache där den vidareutvecklas under namnet Xerces[9], under Apache vanliga licens som är betydligt friare än IBMs ursprungliga.

I början användes IBM's parser fram till dess att apache tog över den, sedan dess användes Xerces istället, vilket har fungerat utmärkt. Det som ännu inte är standardiserat i DOM standarden är hur programmet som returnerar ett DOM objekt skall åberopas. En sådan standard är på väg och kommer att göra det betydligt lättare att byta till en annan DOM programvara vid behov.

2.3.1.3 JDOM

JDOM[10] är ett gränssnitt ovanpå DOM och SAX i java. JDOM är mer anpassat till javas namnstandard när det gäller funktioners namn. Den har bl a mindre överflödig information och bättre struktur. JDOM kan rekommenderas i framtida projekt.

2.4 Relationsdatabaser

Det finns flera olika typer av databaser, bl a relations-, hierkiska- och objekt-orienterade databaser. Idag används vanligtvis relationsdatabaser.

Det finns ett antal olika relationsdatabaser och det gemensamma för de flesta är frågespråket Structured Query Language (SQL)

2.4.1 SQL

2.4.1.1 Standarder

Det finns en standard för SQL som heter SQL92. Den skrevs 1992 och har utvecklats av American National Standards Institute (ANSI)[22]. Standardtexterna är dock svåra att få tag i om man inte är redo att betala för dem. Databasleverantörerna har implementerat olika delar av SQL92 standarden och också tillfört egna varianter vilket medfört

kompabilitetsproblem.

SQL består huvudsakligen av två delar.

2.4.1.2 DML

Data Modification Language (DML) är den del av SQL som är mest standardiserad. DML används för att ändra och sätta in data i databasen. Det finns de som innefattar även att hämta data från databasen i DML begreppet(vilket görs i denna rapport) andra har det som en tredje del, utöver DML och DDL. Det som skiljer de olika databaserna åt är vanligtvis datatyperna, datumhanteringen och om de har ett transaktionssystem eller inte. Transaktionssystem används bl a för att på ett atomärt sätt genomföra flera förändringar (Commit/Rollback).

2.4.1.3 DDL

Data Definition Language (DDL) används bl a för att definiera vilka tabeller och restriktioner som ska gälla, vilka främmande nycklar som ska finnas mm. DDL syntaxen skiljer sig ofta mycket mellan olika leverantörer.

2.4.2 Tillverkare

Det finns många kommersiella tillverkare av relationsdatabassystem och ett fåtal fria. Två av dessa beskrivs nedan.

2.4.2.1 Oracle

Oracle är en av de största leverantörerna av kommersiella databassystem (den andra är IBM med DB2). Den är tyvärr dyr att använda i kommersiella syften, men eftersom KTH lyckats få ett bra avtal var det intressant att stödja Oracle. Oracle har implementerat stora delar av SQL92, men har bl a en egen datumhantering.

2.4.2.2 MySql

MySql är ett fritt databassystem som bl a har utvecklats i Sverige. Målet med MySql är inte att stödja alla delar av SQL92 eller liknande, utan har inriktat sig mot att få en så snabb databas som möjligt. Detta har medfört att det inte finns någon transaktionshantering och att en del av de DML kommandon som brukar finnas inte stöds.

2.4.3 Gränssnitt

Java Database Connectivity (JDBC) är ett gränssnitt i java mot databaser, vilken baserar sig på en c++ standard vid namn Open Database Connectivity (ODBC). ODBC/JDBC mål är att försöka komma förbi de tillverkarspecifika gränssnitt som var dominerande innan ODBC/JDBC kom. Det är fortfarande problem med att SQL-kommandon som skall utföras (genom ODBC/JDBC gränssnittet) inte är kompatibla mellan olika databaser. Detta problem håller på att lösas för DML kommandon, men det kommer dröja länge innan det är löst för DDL kommandon.

2.5 Kerberos

Några termer:

Principal Principal är en användares eller en dators identitet i ett kerberossystem.

KDC Key Distribution Center, Nyckelcentral. Nyckelcentralen är den viktigaste datorn i ett Kerberos system. KDC:n innehåller alla identiteters krypterade nycklar.

Realm Kerberos domän. Alla datorer och användare tillhör en kerberos domän. Kerberos domänen är oftast den samma som den dns-domän datorerna tillhör, men med versaler. T ex en dator vid namn gaston.e.kth.se tillhör kerberos domänen E.KTH.SE.

Ticket Biljetter används för identifikation och kryptering med de program som använder kerberos. Dessa är tidsbegränsade, vanligtvis med 8 timmar.

TGT Ticket Granting Ticket. Biljett som används för att få andra biljetter.

Interrealm Interrealmsupport, Mellan kerberos domänsupport krävs för att använda kerberos mellan olika kerberos domäner, t ex olika delar av KTH.

Kerberos är ett identifikationssystem som utvecklades som en del av Athena projektet vid Massachusetts Institute of Technology (MIT)[21]. Systemet skapades för att inte vara beroende av att arbetsstationerna var säkra eftersom detta inte gick att uppnå med studenter på en teknisk högskola. Systemet utformades så att alla krypterade nycklar finns på nyckelcentralen, som i sin tur måste vara säker. Vid inloggning i ett Kerberosierat system används lösenordet för att hämta en TGT från nyckelcentralen. TGT används sedan för att hämta ut en biljett som används för att identifiera och kryptera kommunikationen till de olika tjänsterna som används. Eftersom alla biljetter är tidsbegränsade är det inte en katastrof om någon student lyckas få tillgång till dem.

Tidstämplar i protokollet finns för att skydda systemet mot omsändningsattacker, vilket för med sig att systemet inte fungerar för den klient vars klocka går fel.

Mellan kerberos domänsupporten används för att på ett för användarna omärkbart sätt kommunicera med en tjänst i en annan realm.

Vid utvecklingen av Kerberos version 5 tillvaratog men erfarenheterna från version 4:a. Det infördes bl a:

- Ett mer dynamiskt krypteringssystem. Version 4:a använder endast DES, medan version 5 kan använda flera olika krypteringsalgoritmer.
- Interrealmsupporten förenklades administrativt, antalet nyckelutväxlingar minskades.
- Ett ökat skydd mot lösenordsattacker baserat på t ex en ordlista.

2.5.1 JCSI

Java Crypto and Security Implementation (JCSI) är en implementation av bl a en kerberosklient i java. Tyvärr är inte implementation riktigt färdig. De begränsningar som framkommit är bl a

1. Om KDCn är inställd på version 4 kompabilitets läge uppstår problem när man försöker få en biljett. Den använder inte informationen från KDC om hur den ska skapa avkrypteringsnyckeln.
2. Det saknas interrealmsupport, vilket gör det svårt att få den att kommunicera med servrar i andra domäner.
3. Den följer inte standarden för var de egna nycklarna ska sparas ner på disk, vilket gör att man manuellt måste kopiera nycklarna till det ställe som JCSI tycker att de ska ligga på eller så måste koden ändras.

JCSI har utvecklats en del sedan koden skrev, och den är bättre än förut, men har fortfarande brister.

Kapitel 3

Implementation och begränsningar

Som beskrevs i introduktionen behövs följande delar

- Databas
- Användaridentifiering och kryptering
- Grupp och rättigheter
- Distribution

Databasen behvös för att lagra information i. När en användare vill hämta eller ändra information i databasen måste användaren identifieras. Grupp och rättighets systemet kontrollerar om användaren har de rättigheter som krävs och tillåter annars inte användaren att hämta information från databasen. Kommunikationen med användaren kan ske krypterat när användaren är identifierad. Distibutionssystemet behövs för att innehållet i databasen ska vara tillgänglig på mer än en nod. Större delen av databaslagret och delar av rättighetssystemet är implementerat.

3.1 Databasgränssnitt

Målet var från början att skriva ett databasgränssnitt oberoende av vilken databasmotor som finns i botten. Anledningen till detta är att man vill kunna köra olika delar av systemet på databaser från olika leverantörer. Nackdelen med den lösningen är att det är svårare att dra nytta av de olika speciallösningar som olika leverantörer har gjort, bl a annat när det gäller säkerhetslösningar.

Man vill kunna skapa tabeller och kontrollera rättigheter. SQL är inte tillräckligt strikt definierat för att enkelt kontrollera rättigheter. Dessutom är inte DDL kommandona för att skapa tabeller tillräckligt standardiserade.

För att lösa detta problem utvecklades ett nytt gränssnitt i XML. Det blir tyvärr ytterligare ett gränsnitt att lära sig, men det är likt SQL för att underlätta för den som redan kan SQL. Intryck från utvecklingen av XSQL hos oracle[17] togs tillvara när gränssnittet utvecklades. På grund av begränssningar i de databaser man vill stödja blev man tvungen att införa diverse begränsningar igränsnittet. Det går bl a inte att använda Commit och Rollback[1] och det går inte att ändra en definition av en tabell.

3.1.1 Specifikation av databasgränssnittet

XML-definitionen av gränsnittet finns i bilaga A.1 på sid 31. Nedan följer en genomgång av de olika delarna.

3.1.1.1 Tjänstebegreppet

För att skapa olika delar i en databas har det införts ett tjänste begrepp som i XML representeras av `service`-taggen. tjänsten är det som i Oracle kallas schema, och MySQL databaser kallas det för olika databaser, vilket innebär att det går att ha tabeller som heter samma sak under olika tjänster namn. Tjänsten Internal används internt för att skapa de tabeller som systemet använder. Den tjänst man har för avsikt att arbeta mot måste alltid anges.

Exempel

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE service SYSTEM "database.dtd">
<service name="Internal">
...
</service>
```

Detta exempel är början på ett databasanrop. Med `service` taggen anges den tjänst som resten av av kommandona ska arbeta mot.

3.1.1.2 Skapa tabeller

Nödvändiga parametrar är de kolumner och restriktioner som tabellen ska innehålla. I den kod som finns går det tyvärr inte att ändra definitionen av en tabell. Detta kan implementeras genom att ge kommandot för att skapa tabellen på nytt. Om tabellen redan

[1] godkänna alla ändringar eller ta tillbaka de, detta går att komma förbi om man endast stödjer databasmototer som stödjer detta.

finns tar programmet reda på det som har ändrats i definitionen och ändrar tabellen där efter.

Kolumner Nödvändiga parametrar är namn och datatyp på den kolumn som skall skapas. De datatyper som är implementerade är:

- Datum, med möjlighet att ytterligare specifiera datumformatet, t ex YYYY-MM-DD HH:MI:SS.
- Tal, där precision och skala kan anges.
- Strängar med statisk och varierande längd.

Ett specialfall på en talkolumn är en räknare. En räknarkolumn får alltid ett unikt värde varje gång något sätts in i tabellen. Detta implementeras olika på olika plattformar. Vissa databaser har en sådan datatyp från början, medan det i t ex Oracle får implementeras med en stored procedure, en trigger och en räknare. Det går att ange standardvärden för alla kolumner.

Restriktioner De restriktioner som finns är kolumnens värde är unikt, att kolumnen har ett värde eller att kolumnen är en del av en primärnyckel. Vidare går det även att specifiera vilka främmandenycklar som ska finnas. Främmande nycklar är en beskrivning av vilken annan kolumn kolumnen är kopplad till. Detta gör att du inte kan sätta in en post i denna tabell utan att ha en motsvarande post i tabellen som specifiseras med den främmande nyckeln.

Exempel

```
<table name="ServiceNames">
  <defcolumn name="Id" unique="true" notnull="false">
    <number><counter/> </number>
  </defcolumn>
  <defcolumn name="Name" primarykey="true">
    <varchar len="30"/>
  </defcolumn>
</table>
```

Detta exempel skapar en tabell med namnet `ServiceNames`. Tabellen innehåller kolumnerna `Id` och `Name`. `Id` är en räknare och `Name` är en textkolumn. `Name` är primärnyckel för tabellen.

Resultat Resultatet av att skapa en tabell är antingen att allting gick bra, vilket innebär att tabellen skapats, eller så har något gått fel. Felet kan bestå i att tabellen redan fanns eller att någon av parametrarna är felaktiga. En vanlig felkälla kan vara att man

har försökt att ge tabellen ett namn som är ett reserverat ord, t ex *Comment* (i Oracle). För att undvika dessa fel behövs en undersökning som har i uppgift att ta reda på alla reserverade ord i de databaser som skall kunna användas.

3.1.1.3 Fråga

Nödvändiga parametrar för att ställa en fråga mot databasen är vilka värden som eftersöks. En frivillig parameter är vilka krav resultatet ska uppfylla.

Värden Värden kan antingen vara konstanter, kolumner från en eller flera olika tabeller eller aggregerade funktioner.

Konstanter kan antingen vara strängar, tal eller dagens datum.

De aggregerade funktionerna som stödjs är:

- Summering av talen i en kolumn
- Antalet rader
- Det största värdet på en kolumn
- Det minsta värdet på en kolumn

Aggregerade funktioner är funktioner som verkar på ett antal rader av en kolumn för att t ex räkna ut summan av alla rader.

Kolumner specifieras genom att ange vilken tabell och kolumn värdet ska tas ifrån. Det går även att specifiera tjänst om man vill ha data från en annan tjänst. Om kolumnen ska ha ett annat namn i resultatet specifierar man ett alias.

Tabeller `Table` taggen används för att ange de tabeller data behövs från. Det går att ange ett alias som kan användas när man anger kolumner och krav. Detta används bl a för att göra motsvarigheten till SQLs self-join.

Krav Du kan ställa krav som de rader som returneras måste uppfylla. Det går att jämföra kolumner med andra kolumner, konstanter eller aggregerade funktioner. De jämförelse operationer som stöds är:

- Större än

- Mindre än
- Likhet
- Olikhet
- Text jämförelse

Det går också att att gruppera och sortera på olika kolumner

Exempel

```
<query>
  <srccolumn table="ServiceNames" column="Id"/>
  <srccolumn table="ServiceNames" column="Name"/>
  <constant alias="enkonstant">EnKonstant</constant>
  <aggfun type="sysdate" alias="DagensDatum"/>
  <from>
    <from_table table="ServiceNames"/>
  </from>
  <where>
    <where_cond type="or">
       <where_exp type="equal">
         <srccolumn table="ServiceNames" column="Id"/>
         <constant>1</constant>
       </where_exp>
       <where_exp type="like">
         <srccolumn table="ServiceNames" column="Name"/>
         <constant>Foo%</constant>
       </where_exp>
    </where_cond>
  </where>
  <order_by>
    <srccolumn table="ServiceNames" column="Name"/>
  </order_by>
</query>
```

Detta är ett exempel på en fråga. Frågan är efter de två kolumerna `Id` och `Name` i tabellen `ServiceNames`, en konstant och en funktion som returnerar dagens datum. Kraven på resultatet är att `Id`-kolumnen är 1 eller att `ServiceNames` börjar med `Foo`. Resultatet är sorterat efter `Name`-kolumnen.

Resultat Resultatet av en fråga består av de rader som uppfyller kraven från tabellen.

3.1.1.4 Insättning

För att göra en insättning av data anges den tabell som värden ska sättas in i. De kolumner där värden ska sättas in anges parvis tillsammans med värdet.

Exempel

```
<insert>
  <valcolumn>
    <srccolumn table="ServiceNames" column="Name"/>
    <constant>Nytt värde</constant>
  </valcolumn>
  <from_table table="ServiceNames"/>
</insert>
```

Detta exempel sätter in värdet `Nytt värde` i kolumnen `Name` i tabellen `ServiceNames`.

Resultatet av en insättning innehåller värdet på de räknare som har uppdaterats.

Exempel

```
<insert status="ok">
  <counter name="ServiceNamesId" value="17"/>
</insert>
```

Detta exempel beskriver hur resultatet kan tänkas se ut efter ovanstående insättning, där värdet på den automatisk räknaren returneras. Det är det värdet som är insatt i `Id`-kolumnen.

3.1.1.5 Uppdatering

Uppdatering av poster kräver samma information som insättning, men kan också specifiera vilka krav som ska gälla. Kraven anges på samma sätt som för en fråga. Resultatet av en uppdatering returnerar hur många rader som har uppdaterats.

Exempel

```
<update>
  <valcolumn>
    <srccolumn table="ServiceNames" column="Name"/>
    <constant>Nytt värde</constant>
  </valcolumn>
  <from_table table="ServiceNames"/>
  <where_cond>
```

```
    <where_exp type="equal">
      <srccolumn table="ServiceNames" column="Id"/>
      <constant>1</constant>
    </where_exp>
  </where_cond>
</update>
```

Detta exempel updaterar kolumnen `Name` så den innehåller `Nytt Värde` för den rad där `Id`-kolumnen är 1.

3.1.1.6 Radering av post(er)

Poster kan raderas genom att ange vilken tabell och vilka krav posterna ska uppfylla. Kraven anges på samma sätt som för en fråga.

3.1.2 Användning av gränsnittet

Följande stycke beskriver vad som krävs för att utföra ett databaskommando.

3.1.2.1 Skapa en kommando sträng

Först skapas en sträng som innehåller det kommando som skall utföras. Exempelvis:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE service SYSTEM "database.dtd">
<service name="Internal">
  <table name="ServiceNames">
    <defcolumn name="Id" unique="true" notnull="false">
      <number><counter/> </number>
    </defcolumn>
    <defcolumn name="Name" primarykey="true">
      <varchar len="30"/>
    </defcolumn>
  </table>
</service>
```

Detta exempel försöker skapa en tabell med två kolumner i tjänsten `Internal`.

3.1.2.2 Skapa ett DOM-objekt

Därefter skapas ett DOM-objekt. DOM-standarden specifierar inte hur detta görs på ett standardiserat sätt, men detta är det skapat ett gränssnitt för i koden. DOM-programvaran kontrollerar att kommando strängen bildar ett giltigt XML-dokument.

3.1.2.3 Skapa ett tjänste-objekt

Ett tjänste-objekt skapas sedan från DOM-objektet. Tjänsteobjektet går igenom DOM-objektet och skapar sedan lämpliga underobjekt, beroende på hur DOM-objektet ser ut.

3.1.2.4 Kontrollerar rättigheterna

Kontroller av rättigheterna görs på olika sätt för olika typer av kommandon. Generellt behöver man först ta reda på vilket rättighetsobjekt som berörs för att sedan kontrollera vilka rättigheter användaren har på objektet. Alla objekt som har med databasgränssnittet att göra ligger under databasobjektet direkt under roten. Mer information om detta finns i stycket Grupper och Rättigheter(3.3) på sida 18.

3.1.2.5 Utför kommandot

SQL-kommandon generas från DOM-objektet och kör denna mot databasen över JDBC. Delvis olika generatorer måste skrivas för olika databasmotorer, men för den del av SQL-kommandon som är lika går det att återanvända kod genom arv.

3.1.2.6 Resultat

Resultatet visar om allting gick bra och ifall det finns något resultat (t ex vid en fråga) så returneras detta. Ett exempel:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE service SYSTEM "/home/mikan/src/xjobb/xml/dtd/serviceresp.dtd">
<service name="Group">
<insert status="ok"></insert>
</service>
```

3.2 Identifikation och Kryptering

JCSI och Kerberos5 används för identifikation och kryptering, men det är också tänkt att det enkelt ska gå att använda ett annat identifikationssystem om så önskas. Det behövs några enkla klasser och funktioner för att kunna initiera en server eller klient för att sedan kunna skicka och ta emot data.

Om kommunikationerna sker med Kerberos5 kommer identifieringen av klient och server se ut enligt följande format “AUTH:KRB5:id@REALM”. Det data som skickas är XML-dokument som innehåller kommandon till olika delar av systemet.

3.2.1 Klient

Klient-klassen ska ha funktioner för:

1. Starta kommunikation mot en server på valfritt ipnummer och port.
2. Identifiera servern.
3. Skicka data.
4. Ta emot data.
5. Avsluta kommunikationen med servern.

3.2.2 Server

Server-klassen ska ha funktioner för:

1. Initieras och lyssna på valfri port.
2. När en klient ansluter sig initieras en ny tråd som har hand om all kommunikation med klienten.
3. Identifiera klienten, dess ipnummer och port.
4. Ta emot data.
5. Behandla mottagen data.
6. Skicka tillbaka resultatet från operationen.
7. När klienten avslutar kommunikationen så ska tråden avslutas

3.3 Grupper och Rättigheter

Målet var att skapa ett generellt grupp- och rättighetssystem som går att använda för andra applikationer än det ursprungliga databasgränsnittet. Modellen för systemet inspirerades från en vidare utveckling av GSS-API vid namn XGSS-API[18], samt från GAA-API[19] som är en modell för generisk authentifiering och rättighetskontroll.

3.3.1 Grupper

Grupper har följande attribut:

- Id, ett unikt numeriskt id på gruppen
- Klassbeteckning
- Gruppens namn
- Ägaren till gruppen
- Skaparen av gruppen
- När gruppen skapades.
- Kommentar

Klassbeteckningen anger vilken typ av grupp det är. Exempel på klasser är:

- System, som används internt av systemet.
- AUTH:KRB5, är de som identifierat sig med kerberos 5.
- GROUP, som alla vanliga grupper har.

Det finns några speciella grupper:

- AuthUser i klassen System som innehåller alla användare i systemet.
- AuthUser i klassen AUTH:KRB5, som innehåller alla användare i systemet som har identifierade med Kerberos 5.

När en ny användare skapas (t ex mikan i domänen E.KTH.SE) görs följande:

1. Skapar gruppen mikan@E.KTH.SE i klassen GROUP
2. Skapar gruppen mikan@E.KTH.SE i klassen AUTH:KRB5. Om man vill identifiera sig med något annat än Kerberos5 används en annan klassbeteckning som börjar på AUTH:.

3. Sätter gruppen med klassen AUTH:KRB5 som medlem i gruppen med klassen GROUP.

4. Om man vill använda fler identifieringsmetoder, upprepa punkt 3 och 4:a.

5. Ge de önskade rättigheter för användaren på gruppen i klassen GROUP. Rättigheter kan också ges efter hur folk har identifierat sig genom att använda gruppen med den identifieringsklass du önskar, t ex AUTH:KRB5.

3.3.2 Säkerhetsobjekt

Ursprungligen var avsikten att identifiera och ge rättigheter på kolumner som uppfyllde krav såsom "Raderna där värdet från kolumnen `namn` börjar med E". Tyvärr visade det sig svårt att kontstruera ett dylikt system, delvis pga att det inte finns någon standard för detta. Att lösa det som ett lager ovanför kan göra att databasanropen tar orimligt lång tid. Detta beror på att man för varje rad som ska visas behöver göra en kontroll i en annan tabell. På databasnivå går det att lösa detta med en join mellan två tabeller.

Modellen har därför förenklats så att det bara går att ge rättigheter med ett specifikt id. Ett objekt måste innehålla följande data.

- Id, en numeriskt id för objekten, där Root-objektet alltid har nr 1.
- Parent, fadern till objektet, är null på Root-objektet
- Name, vilken namn objektet har, exempelvis Databas, Group, Table, Row.
- Value, textsträng som identifierar objektet i t ex en tabell. Detta är ett fristående fält för att lättare kunna göra en jämförelse med idfältet.
- Desc, en kort beskrivning av objektet. Behöver inte vara satt och är till för att användare lättare ska förstå vad det är för objekt.

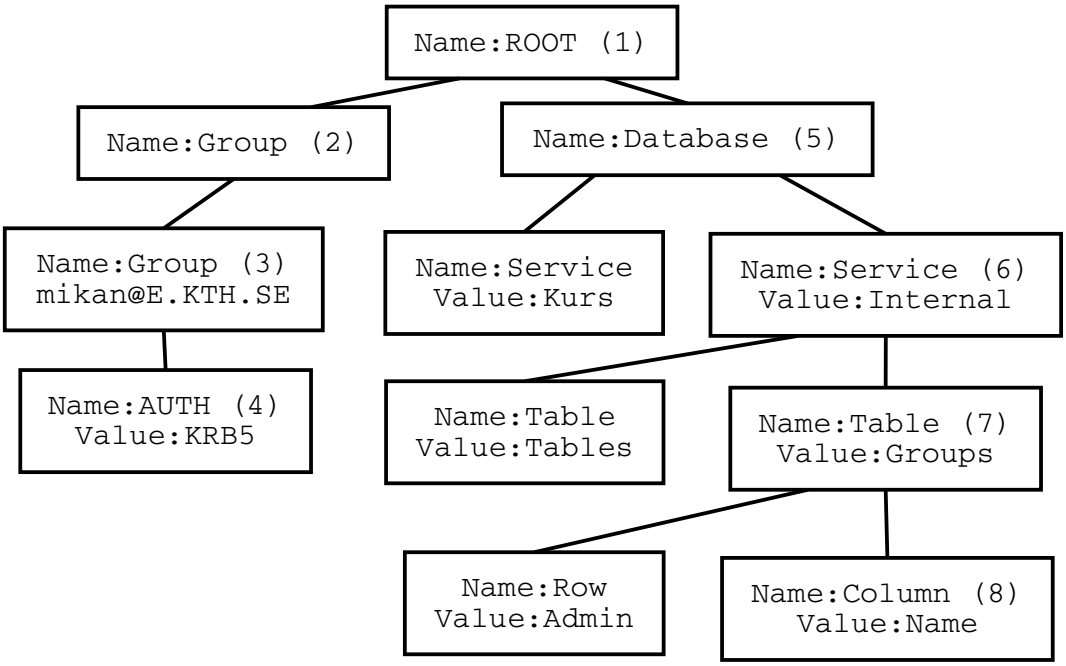
Av objekten byggs ett träd med ett Root objekt i toppen (se figur 3.1).

3.3.3 Rättigheter

För att kontrollera vilken rättigheter som gäller måste trädet (som byggs av objekten) traverseras. För varje nivå i trädet kontrolleras vilka rättigheter som finns, och därefter skickas resultatet ner till nästa nivå. Detta utföres för alla nivåer.

Rättigheter ges för en grupp på ett objekt och har följande attribut:

- Grupp, vilken grupp som har rättigheter
- Objekt, objektet rättigheterna gäller



Figur 3.1: Träd av säkerhetsobjekt. 1) Root-objektet som är urfader till alla objekt. 2) Fadern till alla gruppobjekt och här kan gruppapplikationen härja fritt. 3) Gruppen mikan@E.KTH.SE i klass GROUP. 4) Gruppen mikan@E.KTH.SE i klassen AUTH:KRB5.
5) Fadern till alla objekt som rör databaslagret. 6) Fadern till alla objekt i tjänsten Internal. 7) Tabellen Groups i tjänsten Internal. 8) Kolumnen Name i tabellen Groups.

- Rättighet, sträng som bestämmer vilken rättighet det gäller, t ex “CREATE-TABLE”
- Value, antingen true el false.

Nedan beskrivs rättigheter som olika operationerna kräver.

Skapa tabell För att skapa en tabell behövs rättigheten CREATE-TABLE på ett objekt av klassen Databas.

Fråga För att kunna göra en fråga mot en tabell måste man dels ha SELECT-rättigheter på de kolumner som eftersöks och ha SELECT-rättigheten på de rader som ska returneras.

Insättning För att kunna sätta in data i en tabell behövs INSERT-rättigheten på tabellen och ha INSERT-rättigheten på kolumnerna.

Uppdatering För att kunna sätta in data i en tabell behövs UPDATE-rättigheten på tabellen och ha UPDATE-rättigheten på kolumnerna.

Radering av post(er) För att kunna sätta in data i en tabell behövs DELETE-rättigheten på tabellen och ha DELETE-rättigheten på kolumnerna om rättigheterna saknas sätts kolumnerna till NULL. Om rättigheterna finns på primär nyckeln men inte på alla värdekolumner returneras ett fel.

3.3.4 Gränssnitt

Gränsnittet är uppdelat i tre delar, grupp-,objekt- och rättighetsoperationer.

3.3.4.1 Gruppoperationer

Skapa en grupp För att skapa en grupp anges gruppens namn, gruppens klass och ägaren till gruppen. Det går också att ange en kommentar.

Hämta information om en eller flera grupper För att hämta information om en eller flera grupper behöver man ange vilken typ av jämförelse som skall göras, en exakt jämförelse eller finna alla grupper som innehåller ett angivet attribut. De attribut som kan anges är:

- Namnet på gruppen
- Klassen på gruppen
- Gruppens kommentar
- Ägaren av gruppen
- Gruppens skapare

Ändra en grupp För att ändra en grupp måste namn och klass på den grupp som skall ändras anges. Utöver detta kan man ange:

- Den nya klassen på gruppen
- Det nya namnet på gruppen
- Den nya ägaren av gruppen

Addera en grupp till en grupp För att lägga till en grupp anges gruppen samt den grupp den ska bli medlem i. Det går också att ange en kommentar till medlemskapet.

Medlemmar i en grupp För att att se vilka medlemmar en grupp har måste gruppnamnet och klassen anges. Det som erhålles är en lista på vilka grupper som är medlemmar och vilken kommentar som gavs när de adderades till gruppen.

3.3.4.2 Objektoperationer

Skapa objekt För att skapa ett Objekt måste det namn och värde som objektet ska ha anges. Om fadern inte anges fås Root-objektet som fader. Det går att ange en kommentar på objektet.

Hämta information om ett objekt För att hämta information om ett objekt krävs att Id:et för objektet anges.

Hämta information om barn till ett objekt För att hämta information om vilka barn ett objekt har måste objektets Id anges. Det går också att ange vilka krav som barnen ska uppfylla när det gäller namn och värde.

Radera ett objekt Ett objekt raderas genom att ange dess värde. Observera att dess barn också raderas.

3.3.4.3 Rättighetsoperationer

Ge rättigheter på ett objekt För att ge rättigheter på ett objekt måste gruppen man vill ge rättigheter på anges. Rättigheten samt dess värde måste också anges.

Ta bort rättigheter Parametrarna är:

- Gruppen som rättigheter ska tas bort ifrån
- Objektet som rättigheten ska tas bort ifrån
- Rättigheten som ska tas bort

Rättigheten `Gruppen` har på `Objektet` tas bort.

Lista rättigheter Parametrarna är:

- `Gruppen` funktionen utgår ifrån
- `Objektet` som gruppen har rättigheter på
- Rättigheten som eftersöks

Funktionen kontrollerar om `Gruppen` har några rättigheter på `Objektet`, i så fall returneras de rättigheterna, annars returnerar den att inga rättigheter har hittats. Rättigheterna på förfäderna till objektet måste också kontrolleras om det finns en mer generell rättighet högre upp i trädet.

Lista rättigheter rekursivt Parametrarna är:

- `Gruppen` som man ska utgå ifrån
- `Objektet` som rättigheter eftersöks på.
- Rättigheten som eftersöks

Denna funktion kontroller vilka rättigheter som `Gruppen` har på `Objektet`. För att göra det kontrollerar man först med den icke rekursiva metoden om `Gruppen` har den önskade rättigheten. Om den sökta rättigheten inte hittas ska man undersöka om de grupper som `Gruppen` är medlem i har den sökta rättigheten. Detta görs genom att först gå igenom de grupper som `Gruppen` är direkt medlem i. Detta görs i bokstavsordning (categori,namn). Om den sökta rättigheten fortfarande inte har hittats går man igenom alla grupper som `Gruppen` är indirekt medlem i. Först alla grupper som `Gruppen` är medlem i genom en annan grupp, därefter de grupper som `Gruppen` är medlem i genom två andra grupper, osv. Detta för att de grupper som `Gruppen` är närmast medlem i ska ha störst betydelse.

3.4 Distributionslager

Distributionslagret är indelat i prenumereration-, distribution- och mottagarsystem. Prenumerationssystemet tar hand om vilka noder som skall få vilken information. Distributionssystemet tar hand om hur informationen ska förmedlas till dessa noder som registering av nya noder osv. Mottagarsystemet tar hand om hur updateringar av information ska användas på den lokala noden.

3.4.1 Prenumererationssystem

Prenumererationssystemet tar hand om registeringar och avregistrering av vilken data som de olika noderna i systemet vill prenumerera på.

3.4.1.1 Lista prenumerbara tabeller och kolumner

Denna funktion returnerar en beskrivning på de tabeller och kolumner som går att prenumerera på. Resultatet är beroende på vilka prenumererationsrättigheter användaren har.

3.4.1.2 Registrera en prenumereration

Poster går att prenumerera på enligt följande:

- Enskilda poster. Detta kräver en registrering per post som noden vill prenumerera på i en tabell, och en nyckeln för posten behöver också anges.
- Alla poster. Alla nuvarande poster i tabellen och alla poster som i framtiden kommer att sättas in i tabellen.
- Endast nya poster. Noden får endast updatering av poster som sätts in i tabellen, och för dessa skapas en enskild prenumereration.

Det går att prenumerera på enskilda kolumner eller alla kolumner som man har rättigheter att prenumerera på.

Detta implementeras antingen genom en trigger[2] i databasen, eller genom en kontroll i databaslagret av vilka prenumererationer som finns. Det går givetvis snabbare att implementera det med en trigger, men i de fallen det inte finns stöd för trigger i databasen får man göra det i databaslagret. Även om funktionen implementeras med en trigger är det lämpligt att ha information om vilka prenumererationer som finns i en intern tabell, där det även är lämpligt att ha med triggernamnet som en kolumn. När ändringar sker

[2]Funktion som automatiskt utförs vid ändringar i databasen

registeras vad som ska sändas ut till de andra noderna i en intern tabell tillsammans med ett transaktionsid. I denna tabell kontrolles sedan om det finns någon data som ska sändas till andra noder.

3.4.1.3 Lista prenumererationer

Returnerar de prenumererationer som finns för en angiven tabell som utförts av den som frågar. En person som har administrativa rättigheter kan även lista alla prenumererationer som sinns för den angivna tabellen.

3.4.1.4 Avregistrera en prenumereration

Detta tar bort prenumererationsinformationen ur den interna tabellen. Om det är den sista preumererationen för en tabell tas triggern på tabellen också bort.

3.4.2 Distributionssystem

Distributionssystemet tar hand om registering av vilka noder som finns, utsändning av ändrad data osv.

3.4.2.1 Nodregistrering

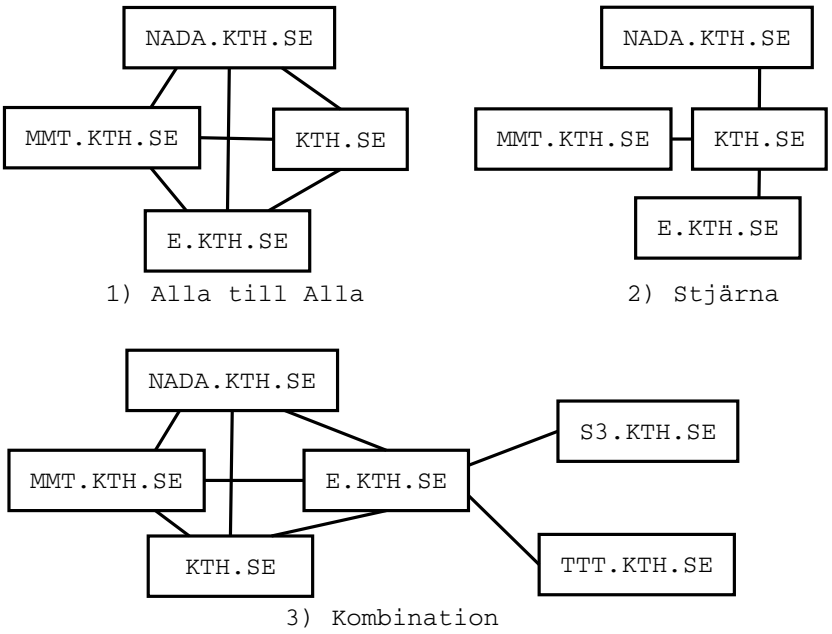
De noder som finns registrerade finns i en intern tabell. Här kan administratorn av systemet bestämma hur distribution av data ska ske.

Det går att bygga olika strukturer (se figur 3.2). Genom att antingen ha en version av tabellen (där alla noder finns med i) som är gemensam för alla noder (alt 1), alternativt går det att gömma noder från varandra genom att inte låta andra noder prenumerera på uppdateringar av de poster som ska gömmas(alt 2). Det går att kombinera på olika sätt (alt 3). Dessa gömda noder kan då bara prenumerera på data som finns på den närmaste noderna, dvs de noder som vet om nodens existens.

3.4.2.2 Distribution av sammanhängande ändringar

Uppdatering som sker i en transaktion, dvs utförs i en kommandosträng skickas vidare tillsammans och bildar en transaktion. Det går också att bifoga en kommentar med transaktionen.

Eftersom räknare i en nod inte behöver ha samma värde på andra noder så måste de specialbehandlas vid överföring av data mellan olika noder. När ändringar som innehåller räknare eller referenser till räknare görs, bifogas information om vilken



Figur 3.2: Nodstrukturer

nod ändringen härstammar ifrån. När ändringen mottages av en annan nod finns en översättningstabell för att översätta ett räknarid på en nod till ett annat räknarid på den lokala noden. Om det sker några framtida uppdateringar av posten måste de berörda posterna i översättningstabellen skickas med, så att andra noder i systemet vet vilken post det rör sig om.

3.4.2.3 Distribution av rättigheter

Vid tilldelning av rättigheter på poster eller tabeller som det finns prenumererationer på måste man även se till att det registreras en prenumereration på rättigheten, för att den ska följa med och vara densamma i hela systemet.

Vid prenumereration på en tabell eller kolumn kontrolleras också vilka rättigheter som påverkar det objektet. Det fås automatiskt en prenumereration på tillhörande rättigheterna.

3.4.2.4 Distribution av data till andra noder

Det finns möjlighet att välja om man vill att ändringar på tabeller ska skickas direkt till andra berörda noder eller det skall vänta till en viss tidpunkt innan data skickas ut.

Om det är en stor mängd data som ska skickas till andra noder kan det vara lämpligt att komprimera den först eftersom XML-dokument lätt blir stora.

3.4.3 Mottagarsystem

Systemet som får en ändring från en annan nod i systemet kan bestämma om ändringen ska godkännas eller inte. Om ändringen godkänns och påverkar tabeller eller rader som andra noder i systemet prenumererar på behöver ändringarna vidaresändas.

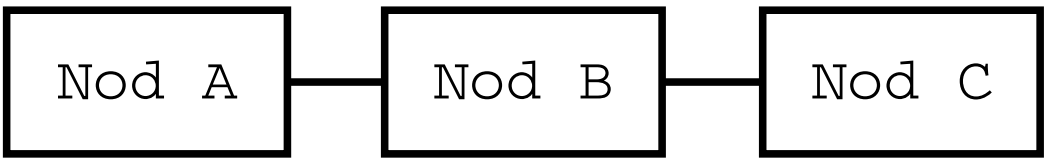
3.4.3.1 Kontroll av ändringar

Mottagande system skall ha möjlighet att välja vilka uppdatering som ska påverka den lokala noden beroende på var ifrån ändringen härstammar, vem som har gjort den, vilka andra noder som har godkänt ändringen och om den påverkar en viss post. Antingen godkänns ändringen, ändringen godkänns inte eller så behöver en operatör tillfrågas om ändringen ska utföras eller inte.

3.4.3.2 Vidare sändning av ändringar till andra noder

Om det finns prenumererationer på tabeller som påverkas av att ändringar utförs på den lokala noden behöver dessa ändringar skickas vidare. Då sänds informationen från orginalnoden vidare med transaktionsidet och vem som har godkännt ändringen på den lokala noden. Detta för att noder som får en uppdatering från flera ställen ska kunna se om dom får dupletter.

3.4.3.3 Exempel



Figur 3.3: Exempel system med tre noder, där nod A och nod C vet om nod B

- Man har ett system bestående av tre noder, A,B och C enligt figur 3.3.
- På nod A skapas tabellen users. Tabellen sätts som allmänt prenumererbar. På nod A sätts regeln upp att ändringar på tabellen users måste godkännas om dom inte är gjorda på nod A, eller tidigare godkännda av staff@B som man litar på.
- Nod B prenumererar på tabellen users från nod A med Alla poster. I och med att nod B prenumererar på tabellen users och att tabellen är allmänt prenumererbar så ser också nod C att tabellen finns tillgänglig. På nod B godkänns alla ändringar som kommer från nod A eller gjorde av staff@C och övriga måste godkännas.

- Nod C prenumererar på tabellen users från B och godkänner alla ändringar från andra noder.
- På nod A sätter admin@A in post 1 i tabellen.
- Nod As distributionssystem skickar insättningen vidare till nod B.
- På nod B kontrolleras vilka regler som gäller vid uppdatering i tabellen users. Eftersom uppdateringen kommer från nod A godkänns den automatiskt.
- Nod Bs distibutionssystem skickar insättningen vidare till nod C.
- På nod C kontrolleras vilka regler som gäller vid uppdatering i tabellen users. Eftersom alla uppdateringar tillåts godkänns den automatiskt.
- På nod C ändrar sedan en användare som inte är medlem i gruppen staff i en rad i tabellen users.
- Nod Cs distributionssystem skickar uppdateringen av raden till Nod B.
- På nod B kontrolleras vilka regler som gäller uppdatering i tabellen users. Eftersom det inte är staff@C som har gjort uppdateringen godkänns den inte, utan läggs i den kö på operationer som måste godkännas innan dom appliceras.
- Administratören för nod B, som är medlem i staff@B, loggar in mot systemet. Administratören ser ändringen av raden i tabellen och godkänner den.
- Nod Bs distributionssystem skickar uppdateringen av raden till nod A.
- På nod A kontrolleras vilka regler som gäller uppdatering av tabellen users. Eftersom ändringen är tidigare godkännd av staff@B så godkänns uppdateringen.

Kapitel 4

Slutsatser

4.1 Svårigheter

Det har dykt upp en hel del svårigheter, dels den vanliga att allting tar mycket längre tid än vad man först tror. Om det hade funnits någon/några som kunde hjälpa till och koda och hade erfarenhet om hur halvstora projekt i Java ska läggas upp, hade det nog kunnat gå att få ett fungerande system på ca 3 manår. Tyvärr, är det något längre än ett examensarbete.

Att skapa ett icke databasspecifikt gränssnitt för ddl kommandon var svårt men också intressant. Efter mycket möda börjar faktiskt systemet att fungera mot en Oracle databas.

Att ha grupper i grupper skapar en del extra traverserande i jämförelse med vad som behövts om begränsningen att inte kunna ha grupper i grupper hade funnits.

Ett generellt distributionssystem är mycket svårare att implementera och handa än ett databasspecifikt. Att göra verifieringen om användaren får göra en operation på ett databas oberoende sätt är också betydligt tyngre och mer komplicerat än att använda de ostandardiserade operationer som finns i vissa databasmotorer.

Rent distributionsmässigt är det svårast att bestämma vad som ska hända när uppdateringar krockar med varandra. Det är tänkt att man skulle kunna fråga en administratör genom administrationsgränsnittet. I större system kommer detta att vara ohanterligt, och en alternativ metod är att föredra. Ett sådant system skulle kunna vara att om en krock matchar ett visst mönster ska A utföras, annars ska B utföras osv.

4.2 Framtid

För att det ska vara intressant att fortsätta utvecklingen av projektet måste man först komma förbi interrealm begränsningarna i jcsi. Det kan ha kommit nyare versioner där det är fixat, men senaste tiden har det varit svårt att komma åt deras websida. Om resurser läggs ner kan ett fungerande system erhållas, med begränsningar avseende generaliteten i systemet.

Det finns ett projekt för att kunna synkronisera data mellan mobila klienter vid namn SyncML[12]. IBM har framfört planer på att använda SyncML för att synkronisera databaser på ett databasoberoende sätt, men det är inte klart än hur strukturer och säkerhetsinformation ska överföras.

Det finns nog en framtid för plattformsoberoende distribuerade datahanteringssystem, även om dom antagligen kommer vara enklare i sitt utförande.

4.3 Summering

Detta arbete kan ligga till grund för en fortsatt utveckling i ett större projekt. Efter minst 2 manårs arbete, lämpligen av två personer under ett helår kan det vara möjligt att få ett fungerande ramverk(programbibliotek).

Att använda XML som gränssnitt för databasoperationer fungerar bra. Det ger ett väldefinierat gränssnitt, där man vet vilka operationer som är möjliga. XML rekommenderas för att överföra data mellan olika plattformar.

Om man bara vill lösa det primära problemet med kontodatabaser, är det lämpligt att försöka använda en ldap katalog eller liknande att spara ner informationen i. Det största argument mot ldap förut var att det inte fanns ett standardiserat sätt för säkrare authentifiering mot ldap servern än klartext, eller klartext över ssl. Nyare versioner av openldap har stöd för sasl authentifiering[13] och TLS[14].

Bilaga A

Databas specifikationer

Nedan följer DTDer för databasgränssnittet.

A.1 Databasgränssnitt

A.1.1 Service.dtd

Används för att databaskommandon.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- Så att vi kan ha åöä i våra kommentarer -->
<!-- TODO:
          Hinta om index i create table
       OBS: Ha inte ngt attribute som innehåller space.
-->

<!-- ..................................................... -->
<!-- .XML specifikation DTD of database interface ........ -->
<!-- ..................................................... -->

<!-- .Du kan inte förutsätta att ordningen mellan -->
<!--  table,query,insert, update bibehålls        -->
<!ELEMENT service (table*,query*,insert*,update*)>
<!ATTLIST service
  name        CDATA #REQUIRED>

<!ELEMENT table (defcolumn+,foreign_key*)>
<!ATTLIST table
  name        CDATA #REQUIRED>
```

```
<!ELEMENT defcolumn ((date|number|char|varchar))>
<!ATTLIST defcolumn
  name        CDATA #REQUIRED
  unique      (false|true) "false"
  notnull     (false|true) "false"
  primarykey  (false|true) "false">

<!ELEMENT default_value (#PCDATA)>
<!ATTLIST default_value
  type       (false|true|sysdate) "false">
<!--
If value is true, you read the #CDATA.
If value is false, you havent any default value.
If value is sysdate, the default value is sysdate,
  only valid when you have a date
-->

<!ELEMENT char_constraint (enum_values*)>
<!ATTLIST char_constraint
  name        CDATA #REQUIRED
  type        (none|upper|lower|enum) "none">

<!ELEMENT enum_values (#PCDATA)>

<!ELEMENT number_constraint (#PCDATA)>
<!ATTLIST number_constraint
  name        CDATA #REQUIRED
  type        (none|greater|less) "none"
>

<!ELEMENT date (default_value?)>
<!ATTLIST date
  format      CDATA "YYYY-MM-DD HH:MI:SS">

<!--
Valid default values are false,sysdate or a date matching the
date format string
-->

<!ELEMENT number ((default_value|counter)?,number_constraint*)>
<!ATTLIST number
  precision   CDATA "10"
  scale       CDATA "0"
>

<!ELEMENT counter EMPTY>
<!ATTLIST counter
  start       CDATA "1"
  step        CDATA "1">
```

```
<!ELEMENT char (default_value?,char_constraint?)>
<!ATTLIST char
  len         CDATA #REQUIRED>

<!ELEMENT varchar (default_value?,char_constraint?)>
<!ATTLIST varchar
  len         CDATA #REQUIRED>

<!ELEMENT foreign_key (foreign_key_local+,foreign_key_remote+)>
<!ATTLIST foreign_key
  name        CDATA #REQUIRED>
<!ELEMENT foreign_key_local EMPTY>
<!ATTLIST foreign_key_local
  column      CDATA #REQUIRED>
<!ELEMENT foreign_key_remote EMPTY>
<!ATTLIST foreign_key_remote
  service     CDATA "default"
  table       CDATA #REQUIRED
  column      CDATA #REQUIRED>

<!ELEMENT query (constant*,srccolumn*,aggfun*,from?,
                 where?,group_by?,order_by?)>
<!ATTLIST query
  distinct    (false|true) "false">

<!ELEMENT srccolumn EMPTY>
<!ATTLIST srccolumn
  table       CDATA #REQUIRED
  column      CDATA #REQUIRED
  alias       CDATA #IMPLIED
  service     CDATA "default"
>

<!-- Table name or alias -->
<!-- alias should be a legal xml tag -->

<!ELEMENT aggfun (srccolumn?)>
<!ATTLIST aggfun
  type        (none|sum|count|max|min|sysdate) "none"
  alias       CDATA #IMPLIED>

<!-- Borde skrivas om
  functioner borde laggas i skilda taggar
  Constant borde få ett attribute som heter type och
  default är none.
  Om det är date så är attributet dateformat viktigt.
 -->

<!ELEMENT constant (#PCDATA)>
```

```
<!ATTLIST constant
   alias       CDATA #IMPLIED
   character   (false|true) "false">

<!ELEMENT from ((from_table|from_subquery)*)>

<!ELEMENT from_table EMPTY>
<!ATTLIST from_table
   table      CDATA #REQUIRED
   alias      CDATA #IMPLIED
   service    CDATA "default">

<!ELEMENT from_subquery (query)>
<!ATTLIST from_subquery
   alias      CDATA #REQUIRED>

<!ELEMENT where (where_cond)>
<!ELEMENT where_cond ((where_exp|where_cond)+)>
<!ATTLIST where_cond
  type          (and|or) "and">
<!ELEMENT where_exp (srccolumn,(srccolumn|aggfun|constant))>
<!ATTLIST where_exp
  type        (none|greater|lower|equal|nequal|like) "none"
  outerjoin   (false|true) "false"
>


<!ELEMENT group_by (srccolumn+,having?)>
<!ELEMENT having (where_cond)>
<!ELEMENT order_by (srccolumn+)>

<!ELEMENT valcolumn (srccolumn,constant)>
<!ELEMENT insert (valcolumn+,from_table)>
<!ELEMENT update (valcolumn+,from_table,where?)>
<!ELEMENT delete (from_table,where?)>
<!-- end of DTD -->
```

A.1.2 Serviceresp.dtd

Dtdn som används för att returnera resultatet från database.dtd

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- Så att vi kan ha åöä i våra kommentarer -->
<!-- TODO:
-->

<!-- ..................................................... -->
```

```
<!-- .XML specifikation DTD of database interface ..... -->
<!-- ..................................................... -->

<!-- .Du kan inte förutsätta att ordningen mellan -->
<!--  table,query,insert, update bibehålls        -->
<!ELEMENT service (table*,query*,insert*,update*)>
<!ATTLIST service
  name        CDATA #REQUIRED>

<!ELEMENT table  EMPTY>
<!ATTLIST table
  name        CDATA #REQUIRED
  status      (ok|error) "error">

<!ELEMENT query (resprow*)>
<!ATTLIST query
  status      (ok|error) "error">

<!-- resprow contains respcol -->

<!ELEMENT resprow (respcol*)>
<!ELEMENT respcol (#PCDATA)>
<!ATTLIST respcol
  name        CDATA #REQUIRED>

<!--  type        (date|number|char) "char"> -->

<!ELEMENT insert (counter*)>
<!ATTLIST insert
  status      (ok|error) "error">
<!ELEMENT counter EMPTY>
<!ATTLIST counter
  name        CDATA #REQUIRED
  value       CDATA #REQUIRED>

<!ELEMENT update EMPTY>
<!ATTLIST update
  number      CDATA #REQUIRED
  status      (ok|error) "error">
<!-- end of DTD -->
```

Litteraturförteckning

[1] RFC 1510 The Kerberos Network Authentication Service (V5). J. Kohl, C. Neuman. September 1993.

[2] Heimdal, en kerberos 5 implementation.

`http://www.pdc.kth.se/heimdal`

[3] Genesereth, M.R., Fikes, R. E. et al. Knowledge Interchange Format(KIF) Version 3
`http://logic.stanford.edu/kif/`

[4] FIPA
`http://www.fipa.org`

[5] Sun Microsystems, SUN `http://www.sun.com`

[6] Java,
`http://www.sun.com/java`

[7] eXtensible Markup Language, XML `http://www3.org/TR/REC-xml`

[8] Document Object Model (DOM) `http://www.w3.org/DOM/`

[9] Xerces - XML parsers in Java, C++ `http://xml.apache.org/xerces2-j/index.html`

[10] Java interface to DOM objects, JDOM `http://jdom.org`

[11] Java Crypto and Security Implementation (JCSI) `http://security.dstc.edu.au/projects/java/jcsi.html`

[12] SyncML,
`http://www.syncml.org`

[13] Authentication Methods for LDAP `http://www.ietf.org/rfc/rfc2829.txt`

[14] Lightweight Directory Access Protocol (v3):
Extension for Transport Layer Security `http://www.ietf.org/rfc/rfc2830.txt`

[15] Java Crypto and Security Implementation
Implementation av cryptering i java, bl a ett kerberos bibliotek. http://security.dstc.edu.au/projects/java/jcsi.html

[16] Oracle Documentation http://technet.oracle.com/docs/content.html

[17] XSQL Xml interface to SQL http://www.oracle.com/oramag/oracle/01-jan/index.html?o11xml.html

[18] Extended Generic Security Service APIs: XGSS-APIs Access control and delegation extensions http://www.ietf.org/proceedings/98dec/I-D/draft-ietf-cat-xgssapi-acc-cntrl-03.txt

[19] Generic Authorization and Access control Application Program Interface
http://www.ietf.org/proceedings/00jul/I-D/cat-gaa-cbind-04.txt
http://www.isi.edu/gost/info/gaaapi/

[20] Kungliga Tekniska Högskolan, Royal University of Technology (KTH) http://www.kth.se

[21] Massachusetts Institute of Technology (MIT) http://www.mit.edu/

[22] American National Standards Institute (ANSI) www.ansi.org

[23] Källkod från detta examensarbete. http://www.mikan.net/~mikan/xjobb