IT 08 008
Examensarbete 30 hp
Mars 2008

Användning av realtidsdatabas i diagnostiksystem

Paulina Hermansson

Institutionen för informationsteknologi
Department of Information Technology

UPPSALA
UNIVERSITET

Teknisk- naturvetenskaplig fakultet
UTH-enheten

Besöksadress:
Ångströmlaboratoriet
Lägerhyddsvägen 1
Hus 4, Plan 0

Postadress:
Box 536
751 21 Uppsala

Telefon:
018 – 471 30 03

Telefax:
018 – 471 30 00

Hemsida:
http://www.teknat.uu.se/student

Abstract

Usage of a Real Time database in a diagnostic system

Paulina Hermansson

The main purpose in this degree thesis is to investigate changes in performance when introducing a newly produced Real-Time database in one of CC Systems products, the Diagnostic Runtime Engine, DRE. The thesis is a collaboration between CC Systems, creator of DRE, and Mimer AB, developer of the Real-Time database.

The result reveals that the time spent on database access is considerably less compared to the originally used database. The research has lead to several additional requirements on the real-time database and these are presented as part of the result in this thesis. The additional features will help making the product more suitable for DRE and similar applications.

Additional research is required to be able to recommend the use of this product or not. Primarily the performance in another operating system, Windows CE, but also the introduction of additional features should be investigated. Part of this new functionality is included in the first formal release of the product, likely to be introduced in late 2007. Further research is continuing within both companies involved.

Product investigated:
Mimer SQL Real-Time Edition C API, 9.3.7D (Beta Version)
A relational database that handles hard and soft real time requirements in an application. It uses standard SQL-queries for the soft Real-Time and a new Real-Time API for the hard real time.

Handledare: Carl-Magnus Moon
Ämnesgranskare: Tore Risch
Examinator: Anders Jansson
IT 08 008
Tryckt av: Reprocentralen ITC

SAMMANFATTNING

Titel: Användning av realtidsdatabas i diagnostiksystem.

Examensarbetets syfte är att undersöka eventuella prestandaförändringar vid införandet av en ny realtidsdatabas i en av CC Systems produkter; diagnostikmotorn DRE (Diagnostic Runtime Engine). Arbetet är ett samarbete mellan CC Systems, skapare av DRE, och Mimer, utvecklare av realtidsdatabasen.

Resultatet av arbetet påvisar att tidsåtgången för att lägga in information i databasen blir betydligt mindre med realtidsdatabasen jämfört med den förut använda databasen. Under arbetets gång har flera ytterligare önskemål på funktionalitet i produkten framkommit och dessa presenteras också som ett resultat i studien. Dessa utökningar syftar till att skapa en produkt som är mer lämpad för användning i DRE och i liknande tillämpningar.

Ytterligare undersökning av produkten behöver göras för att avfärda eller rekommendera användandet av den. Huvudsakligen prestanda i ett annat operativsystem, Windows CE, men också införandet av önskad ny funktionalitet behöver undersökas. Flera av dessa nyheter kommer att finns med då produkten introduceras på marknaden i början av 2008, och vidareutveckling fortgår genom fortsatt samarbete mellan de medverkande företagen.

Undersökt produkt:
Mimer SQL Real-Time Edition C API, 9.3.7D (Beta Version)
En relationsdatabas som hanterar hårda och mjuka realtidskrav för en applikation med hjälp av ordinära SQL-kommandon för mjuk realtid och nytt realtids-API för hård realtid.

1 FÖRORD

Tack till CC Systems och min handledare där, Carl-Magnus Moon, för bra handledning genom hela arbetet. Tack till Lars Gustavsson på CC Systems för vägledning kring databasskripten. Tack till Dag Nyström och Anders Eriksson på Mimer för teknisk support gällande realtidsdatabasen. Tack till mina examensarbetarkollegor, som delat arbetsrum med mig och bidragit till att upprätthålla den goda stämningen under arbetets gång.

2 INLEDNING

CC Systems levererar styrsystem, produkter och tjänster till ett flertal industriföretag. De utvecklar produkter och plattformar för fordon och maskiner. CC Systems har i sin produktserie med styrsystemslösningar, CrossTalk, utvecklat en diagnostikmotor kallad Diagnostic Runtime Engine (DRE). DRE används för att övervaka ett styrsystem och bland annat logga händelser som sker i styrsystemet i en databas. Styrsystemet och diagnostikmotorn körs på CC Systems Windows CE-baserade dator CCP-XS. Databasen som används av DRE är en Mimerdatabas anpassad för inbyggda system, men som använder ODBC (Open Database Connectivity [11] vilket är ett gränssnitt som beskriver koppling till en databas. Detta gör att skrivningar mot databasen inte har en garanterad svarstid och dessutom är relativt långsamma. Detta är inte optimalt för DRE vilket har höga krav på prestanda och egentligen inte behöver ett komplett ODBC-gränssnitt.

Mimer arbetar med utveckling av databaser och har en ny databas under utveckling – Mimer SQL Real-Time Edition. Den databasen har ett realtidsgränssnitt mot databasen och denna produkt förmodas kunna passa väldigt bra i den fordonstillämpning (DRE) som beskrivs ovan.

En begränsning man kan se med databaser är att det tar lång tid att utföra databastransaktioner, och framför allt är de svåra att tidsbestämma transaktionerna då de tar olika lång tid vid olika tillfällen i tiden. Mimer har insett att databaser nu används i allt större grad i inbyggda system och att det där ställs högre krav på hastighet och realtidsegenskaper. De har utifrån detta gjort en satsning på att utveckla en realtidsdatabas. Denna databas är ett realtidsgränssnitt mot en ”vanlig” databas som innebär att man kan utföra vanliga SQL-frågor i mjuk realtid (d.v.s. icke-realtid) samt utföra vissa specifika operationer i hård realtid. Realtidsfunktionaliteten i databasen använder ett koncept kallat *databaspekare* som pekar på en cell inne i en databastabell i den databas men valt att realtidsanpassa. Skapandet av dessa databaspekare är en mjuk realtidsoperation, men att läsa och skriva till den utpekade cellen sker sedan i hård realtid, via databasoperationer och blir betydligt snabbare än vanliga SQL-frågor.

2.1 Problemformulering

Examensarbetet går ut på att undersöka möjligheterna till prestandaförbättringar av DRE genom användandet av det nya realtidsgränssnittet mot databasen. Arbetet skall leda fram till en demonstrator som påvisar vilka prestandavinster och eventuella problem man skulle få vid införandet av realtidsdatabasen i DRE.

2.2 Mål och syfte

I magisterutbildningen; datavetenskapligt program (160 poäng) vid Uppsala Universitet ingår som sista projekt att göra ett examensarbete om 20 poäng. Syftet med examensjobbet är att tillämpa de tekniska färdigheter man lärt sig under utbildningen samt att öva på att presentera sitt arbete i skriftlig och muntlig form.

Detta examensarbete utförs som att samarbete mellan Mimer och CC Systems. Mimer utvecklar realtidsdatabasen och CC Systems utvecklar produkter där realtidsegenskaperna skulle kunna komma till användning. Företagens syfte med arbetet är tillsammans undersöka vilken funktionalitet som är användbar att ha med i en realtidsdatabas som skall användas i miljöer där begränsat minnesutrymme är ett faktum och där korrekthet och snabbhet vid databasåtkomst ändå måste vara högt prioriterad.

2.3 Avgränsning

Denna rapport beskriver CC Systems diagnostiksystem DRE i stora drag och de delsystem som är aktuella för just detta examensarbete. Den beskriver också Mimers realtidsgränssnitt, Mimer SQL Real-Time Edition, och kopplingen däremellan. För ytterligare information kring de system som behandlas i denna rapport hänvisas till företagens egna manualer och dokumentation i ämnet [3][9]

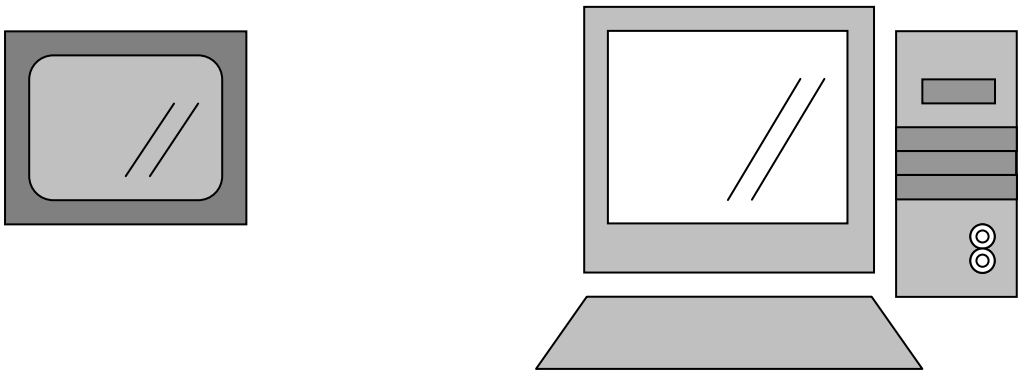
Då Mimer RT fortfarande är under utveckling då detta exjobb genomförts blir undersökningar av eventuella tidsvinster något preliminära. En komplett undersökning skulle kräva en Mimer RT med mer funktionalitet, bland annat skrivning till disk. Några utvalda delar av denna funktionalitet och dess förmodade inverkan på systemet beskrivs mot slutet av denna rapport.

Det avsnitt om relaterad forskning som finns i rapporten är inte någon komplett redogörelse för realtidsforskningen i nuläget men kan ses om en god grund för att söka mer information i ämnet.

Inom ramen för detta examensarbete ligger att införa realtidsfunktionalitet på endast en delmängd av de tabeller som finns i DREs databas. Att realtidsanpassa alla tabeller är inte möjligt inom tidsramen för examensarbetet, dessutom finns i nuläget en övre gräns för antalet realtidsanpassningar i en applikation, för att undvika allt för långa väntetider då man startar systemet. Vår lösning med att realtidsanpassa en delmängd av tabellerna är möjlig p.g.a. den nya realtidsdatabasens förmåga att hantera både realtidsdata och ordinär SQL, vilket är det som används i DRE nu. Den begränsade anpassningen är ändå fullt tillräcklig för att få fram en mätbar och tillförlitlig skillnad i prestanda för de två systemen.

Prestandamätningarna utförs i Windows XP och inte på den plattform, Windows CE, där systemet används i skarp miljö i fordonsdatorerna. DRE kan köras simulerat på PC och det är den miljön som används i examensarbetet på grund av att MimerRT inte är portad till Windows CE ännu (se figur 1).

Originalutförandet	I denna rapport
Windows CE (fordonsdator)	Windows XP (PC)
Diagnostiksystemet DRE	Diagnostiksystemet DRE
Databas: Mimer SQL Mobile	Databas: Mimer SQL Real Time Edition



Figur 1. Plattformsskillnad för diagnostiksystemet i originalutförandet och så som det testats i denna rapport.

2.4 Disposition

Denna rapport beskriver arbetet med att göra prestandamätningar efter införandet av en realtidsdatabas i DRE. De teknikstudier som utförts behandlar DRE, databaser i allmänhet och realtidsdatabasen Mimer SQL Real-Time Edition i synnerhet. Rapporten beskriver också modifieringen av DRE samt införandet av realtidsgränssnittet i detsamma. En demonstrator beskrivs i form av syfte, utförande och resultat med fokus på prestandavinster.

Resultat och diskussion beskriver resultatet från demonstratorns mätningar samt de ytterligare funktioner som under arbetets gång framkommit som önskvärda och som kommer att utvecklas och införas i Mimer SQL Real-Time Edition inom kort.

En kortare analys av andra Realtidssystem har gjorts under rubriken relaterat arbete och dessa delar skall ses som tips för den intresserade läsaren för vidare forskning inom området.

I rapporten förekommer ett antal förkortningar och dessa förklaras finns vid första förekomsten och/eller i den ordlista som finns i slutet av rapporten. I ordlistan beskrivs också många av de fackuttryck som förekommer i arbetet.

3 TEKNIKSTUDIE: DIAGNOSTIKSYSTEMET [10]

3.1 Crosstalk

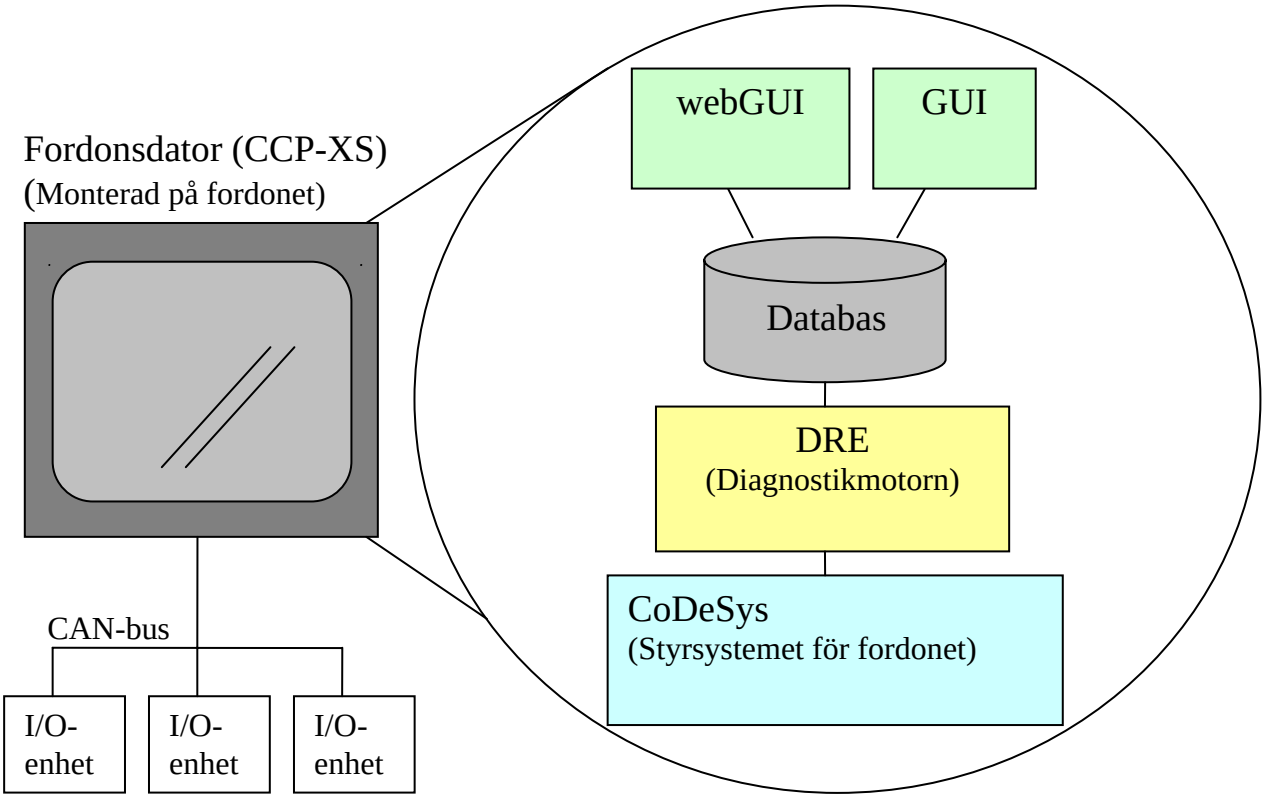
Crosstalk är en plattform för att skapa styrsystem, och är en del i CC Systems produktportfölj. Crosstalk är en samling hårdvaru- och mjukvarukomponenter som man kombinerar för att skapa ett skräddarsytt styrsystem. Crosstalk använder sig av CAN-nätverk, där CAN är en nätverksstandard för kommunikation mellan enheter i fordon, för överföring av signaler samt CoDeSys-baserade styrsystem för maskiner. CoDeSys är en mjukvaruPLC, och PLC är en digital dator som används vid styrning av maskiner. Exempel på Crosstalkprodukt är sopbilars tippningssystem och skogsmaskiners avverkningssystem där föraren övervakar maskinens arbete med hjälp av en pekskärm.

Crosstalk kombinerar styrsystemet för maskinen med en diagnostikmotor, d.v.s. utökar styrsystemet med funktionalitet som gör lösningen mer komplett med egenskaper som gör att den kan bearbeta och i förutse händelser i styrsystemet. Händelser kan t.ex. vara en trasig lampa eller kritisk oljenivå i maskinen och diagnostikdelen i systemet kan då aktivera ett alarm eller skicka ett sms eller e-brev med information om händelsen.

3.1.1 Skapande och användande av systemet

Den kund som köper in ett crosstalksystem kan vara t.ex. ett åkeri, med en maskinpark bestående av sopbilar till vilka de valt att köpa Crosstalk i form av fordonsdatorer med pekskärm, och styrsystemet som skall styra tippning av sopbehållare. Styrsystemet installeras och DRE ställs in så att de hanterar signaler från styrsystemet och utför rätt åtgärder när händelser inträffar. När systemet är installerat och på plats på fordonen integrerar man med systemet via pekskärmen där man kan ge respons på händelser i diagnostiksystemet. Exempel på respons är att kvittera alarm (eng: acknowledge alarm) och vidta de aktuella åtgärderna så som att t.ex. fylla på olja. Exempel på övervakning är ett se de tio senaste alarmen eller textmeddelandena (sms) som skickats ut.

De delkomponenter som systemet består av beskrivs nedan.



Figur 2. Schematisk skiss över fordonsdatorn XS och dess delkomponenter samt koppling till I/O-enheter.

3.1.2 Systemets komponenter

Styrsystemet består av tre huvudsakliga komponenter samt ett par hjälpsystem, som kan kombineras och anpassas efter kundens behov och önskemål (se figur 2):

- **CoDeSys** (Controller Development System)
  Det styrsystem som används är baserat på CoDeSys, och kommunikationen sker via CAN-bussar. CoDeSys är ett av de vanligaste programmeringssystemen för PLC och industriella styrsystem.

- **DRE** (Diagnostic Runtime Engine)
  En diagnostikmotor som är ett ramverk för tillverkandet av diagnoser. DRE hanterar signaler den får från CoDeSys

- **GUI och WebGUI** (Graphical User Interface)
  Användargränssnitt för åtkomst och hanterande av diagnostikfunktionaliteten. Det som operatören ser av systemet.

- **Konfigurationsverktyg**
  Det verktyg som används av för att skapa diagnostikapplikationer.

Det färdiginstallerade styrsystemet, med tillhörande diagnostikfunktionalitet (DRE) körs på CC Systems egenutvecklade Windows CE-baserade dator CCP-XS.

CCP-XS har pekskärm och det är den som är det gränssnitt mot vilket operatören av styrsystemet arbetar. På skärmen visas resultatet av diagnostiksystemets arbete och i de fall där återkoppling från operatören krävs sker också detta via denna pekskärm

3.2 Diagnostikmotorn DRE

Diagnostikmotorn Diagnostic Runtime Engine (DRE) är en del av mjukvaran i styrsystemet och fungerar som övervakare av CoDeSys. DRE kan anpassas så att det utför olika typer av diagnostik och övervakning, samt reagera på olika signaler och händelser i styrsystemet. En reaktion kan t.ex. vara att vid låg oljenivå skicka ett e-brev till en i förväg vald mottagare.

Den bakomliggande logiken för de reaktioner som systemet skall utföra specificeras i diagnostikblock. Ett diagnostikblock tar in signaler, och beroende på hur diagnosblocket är konfigurerat reagerar det och skickar nya signaler vidare till Actionlagret där vidare bearbetning sker och koppling till databasen finns. Diagnostikblocken läses in av systemet vid uppstart.

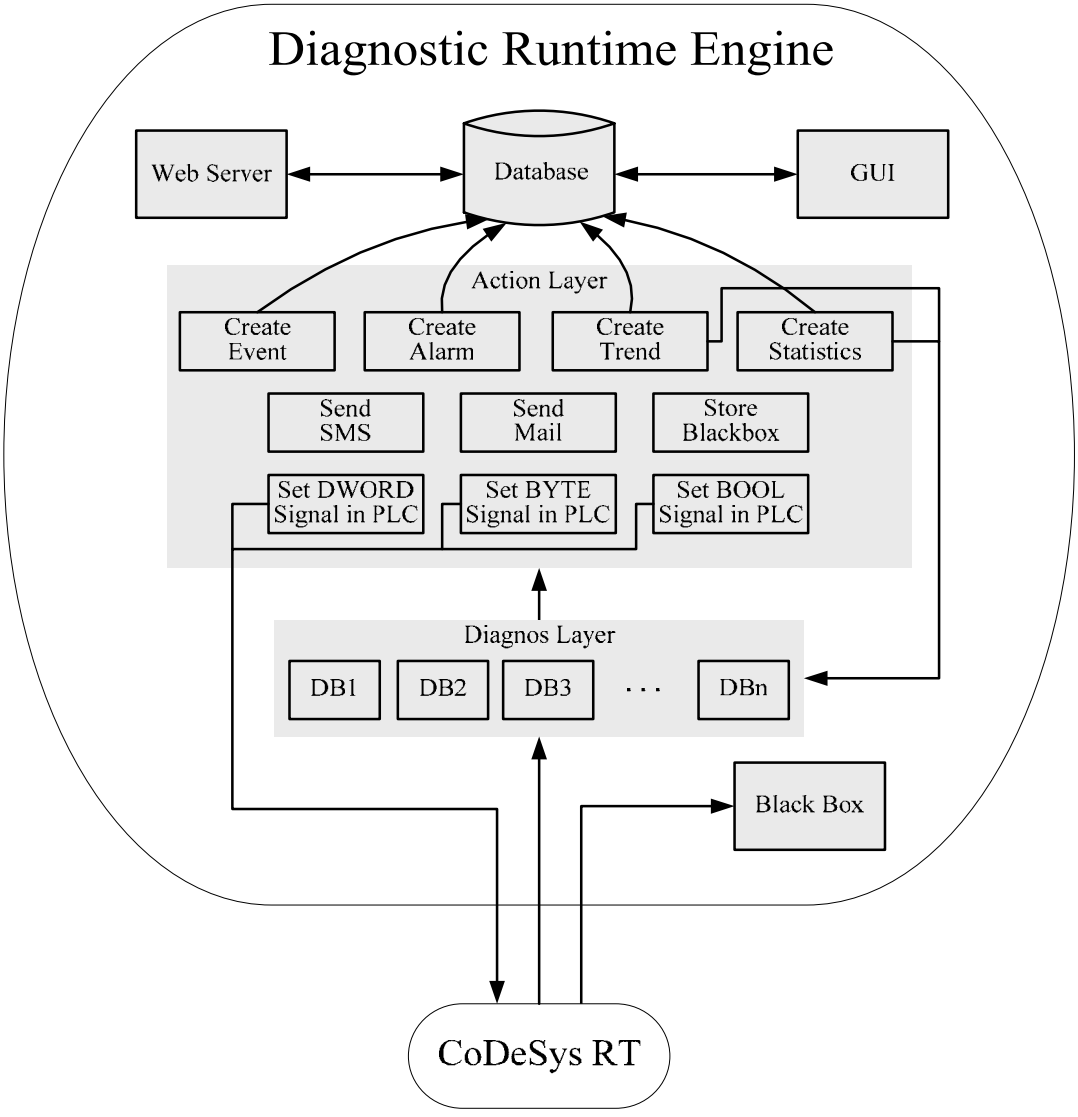
Ett grafiskt PC-program används för att konfigurera DRE . Vid konfigureringen bestäms hur styrsignalerna skall kopplas till olika diagnostikblock och vilka reaktioner systemet skall utföra när ett specifikt diagnostikblock är aktiverat. Konfigurationsfilerna (XML-filer) innehåller information om vilka diagnostikblock och actionblock som skall användas, namnen på dessa block och kopplingarna mellan dem. Filerna som skapas kan föras över automatiskt till DRE.

3.3 GUI

GUI (användargränssnitt) är den del av applikationen som hämtar information från databasen och presenterar den på skärmen på ett användarvänligt sätt. Det är också via detta GUI som operatören hanterar sådana delar av diagnostikprocessen som kräver respons från honom/henne. Som ett komplement till det vanliga GUIt finns ett webGUI (webbanvändargränssnitt) som visar samma information som det vanliga gränssnittet fast på en webbsida. Detta webGUI drivs av en inbyggd webbserver i Windows CE, och möjliggör samma typ av interaktion med systemet som det vanliga gränssnittet.

3.4 Signalflöde

Signaler läses från CoDeSys in till diagnosblocken som bearbetar informationen och skickar vidare information till rätt actionblock, som skapar händelser och interagerar med databasen. (se figur 3)



Figur 3. Signalflödet genom DRE, styrsignaler från CoDeSys Runtime (RT) via diagnosblock i Diagnos Layer till Action Layer och in till databasen.

Ett diagnosblock är ett block med ett antal ingångar och utgångar Blocken innehåller logik för at kontrollera de inkommande värdena. Om dessa inte är normala skickas utgående värden som aktiverar händelseblocken som finns kopplade till det aktuella diagnosblocket. Ett händelseblock agerar efter att ha fått signaler från ett diagnostikblock. I vissa fall kan man överföra information från diagnostikblocket till händelseblocket för att styra mer precist hur händelseblocket skall agera.

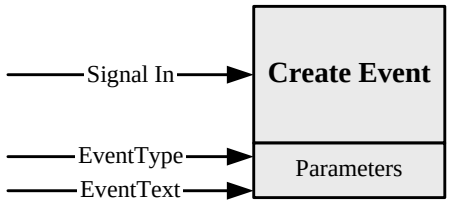
Följande händelseblock finns tillgängliga i diagnostiksystemet:

- Skapa händelse (event) (Create Event)
- Skapa trend (Create Trend)
- Skapa alarm (Create Alarm)
- Skapa statistik (Create Statistics)
- Sänd SMS (Send SMS)
- Sänd E-brev (Send Email)
- Sätt DWORD-signal i PLC (Set DWORD signal in PLC)
- Sätt BYTE-signal i PLC (Set BYTE signal in PLC)
- Sätt BOOL-signal i PLC (Set BOOL signal in PLC)
- Spara undan den svarta lådan (Store Blackbox)

I denna rapport beskrivs endast ett av dessa händelseblock, skapa händelse (Event). För information kring de övriga händelseblock hänvisas till CC Systems manual för diagnostiksystemet [10].

3.4.1 Skapa event

Ett event innehåller ett nummer, en text, en typ, en tidsstämpel och en ”subcode” (Figur 4). Skapa event är ett händelseblock som skapar ett event som sedan läggs in i databasen (se Figur 5).



Figur 4. Skapa event (Create Event).

I eventdatatabellen finns följande fem fält.
EVENTNUMBER – Löpande nummer som genereras automatiskt för varje nytt event.
TEXT – Inkommande parameter, olika vid olika typer av event.
TYPEOFEVENT – Inkommande parameter.
TIMESTAMP – Aktuell tid då eventet skapades. Genereras automatiskt.
SUBCODE - Inkommande signalvärde från diagnosblocket.

Eventdata

EVENTNR	TEXT	TYPE	TIMESTAMP	SUBCODE
1	Test Event 1	10	2007-08-20 12:11:10	10
2	Test Event 2	11	2007-08-20 12:11:11	11
3	Test Event 3	9	2007-08-20 12:11:16	9
4	Test Event 4	11	2007-08-20 12:11:16	11
. . .				

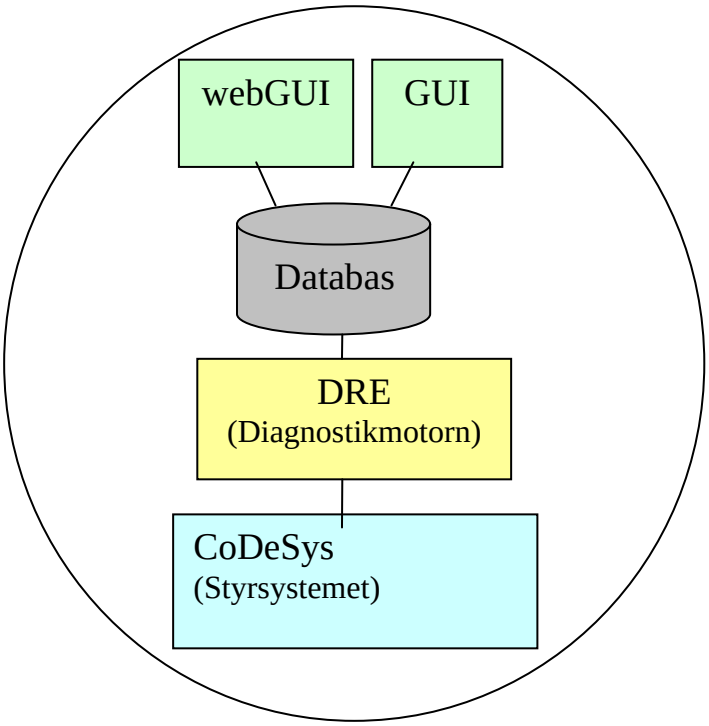
20 000 rader

Figur 5. Databastabellen Eventdata, med exempel på innehåll.

3.5 Databashanterare: Mimer SQL Mobile

Databashanteraren som systemet använder heter Mimer SQL Mobile och är ett databashanteringssystem (eng DBMS – Database Management System) som är speciellt utformad för mobila/inbyggda system. Mimer SQL Mobile är en relationsdatabas, och sådana beskrivs i Wikipedia [21] på detta sätt: ”En relationsdatabas består vanligtvis av en databas med flera tabeller, även kallade relationer. Informationen sparas i rader i tabellerna. Idag kan data av en mängd olika typer sparas, inte bara text och numeriska data utan också bilder, ljud och video.” Mer om detta i kommande kapitel 4.1.1 (Definitioner och förklaringar till databasbegrepp)

I databasen lagrar diagnostiksystemet all den information som skall finnas tillgänglig för GUIt. GUI hämtar data från databasen och presenterar det för användaren på skärmen och databasen är på detta sätt den enda kopplingen mellan GUI och DRE. (
Figur 6)



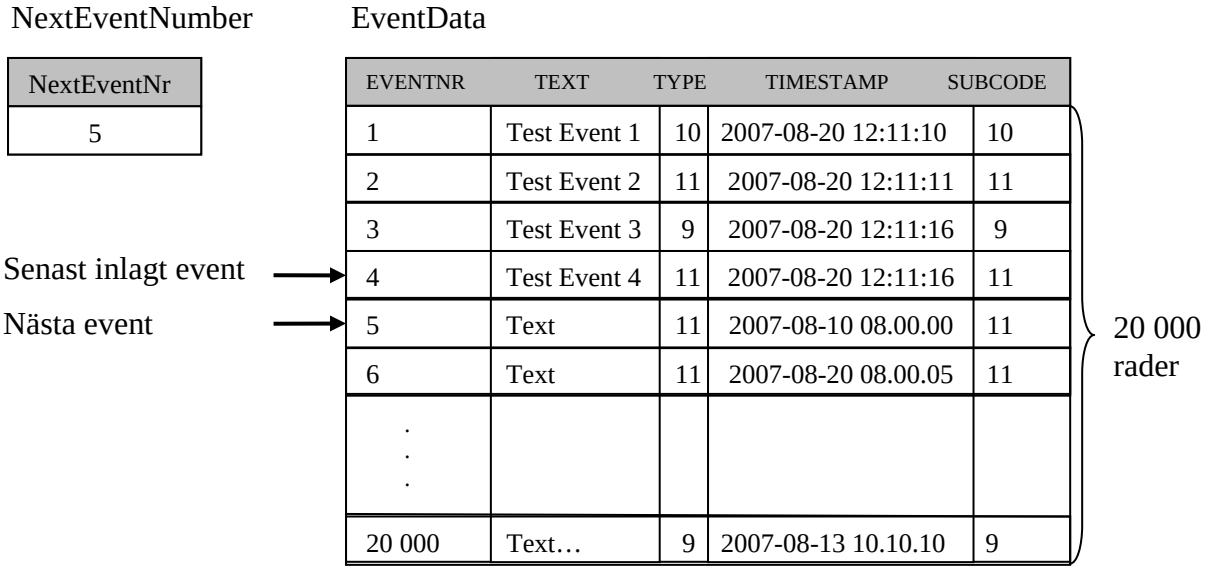
Figur 6**.** Databasens koppling till GUI och DRE .

Följande tabeller finns databasen (antalet rader inom parentes):

- eventdata (40 000)
- alarmlog (40 000)
- smsesSent (200)
- emailsSent (200)
- trenddata (100)
- machinedata
- ett antal enfältstabeller

De två tabeller som anpassats för realtidstillämpningen i denna rapport är (Figur 7):

- eventData: Storleken har minskats till 20 000 rader för att underlätta realtidsanpassningen. Eventen läggs in på ett cirkulärt sätt så att tabellen alltid innehåller de 20 000 senaste eventen i tidsordning.
- nextEventNumber: Enfältstabell som innehåller information om aktuell rad att lägga in nästa event på.



Figur 7. Databastabellen redo att uppdatera rad nr 5.

4 TEKNIKSTUDIE: DATABASSYSTEM

Marknadens behov av inbyggda, snabba och säkra datalagringssystem fortsätter att öka. De allt mer avancerade tillämpningarna gör att lagring av information i vanliga filsystem inte är tillräckligt kraftfulla. I en sådan lösning saknas bland annat säkerhet vad gäller hantering av data och åtkomstkontroll. För att slippa de bristerna använder man sig istället av databassystem (databaser med tillhörande databashanterare).

Generella databashanterare är ofta dåliga på att garantera tider för läsning och skrivning. De är utvecklade för att optimera genomströmningen av data och använder sig av parallellism mellan användare [8]. Tidsåtgången för databasens arbete så som lagring av data, sökning bland data och uppdatering av data varierar således mycket. Databashanteringssystemen är vanligen snabba i genomsnittsfallet men kan ha en mycket lång värstafallstid.

4.1.1 Definitioner och förklaringar till databasbegrepp

Här följer en genomgång av databasbegrepp som är aktuella i sammanhanget.

Databas [1]
En databas är en samling information som är organiserad på ett sådant sätt att det är lätt att söka efter och hämta enskilda bitar information, samt i regel även att ändra informationen.

Databashanterare [2]
En databashanterare (eng. Database Management System, DBMS) är en programvara för att hantera data i en databas. Den består av en samling data och program för att komma åt och hantera informationen. Hanteringen består i att lagra, söka och uppdatera posterna i databasen. Databashanterare är till för fortlöpande kontroll av att databasen är uppdaterad, fullständig och tillförlitlig. De är oftast serverbaserade och arbetar enligt det s.k. klient/serverkonceptet.

Databassystem [12]
Databashanterarens mjukvara tillsammans med den databas den hanterar. Ibland är också applikationen inkluderad.

Realtidssystem [13]
I datavetenskaplig bemärkelse datorsystem vars korrekthet inte bara är avhängigt de logiska resultaten utan också *när* dessa lämnas. Realtidssystem behöver inte vara effektiva och snabba (vilket inte hindrar att många är det), det kritiska kravet är istället tidsmässig förutsägbarhet i "worst case" scenarion. Realtidssystem brukar delas in två olika kategorier, hård och mjuk realtid. Ett exempel på där

hårda realtidskrav finns är styrsystemet i en bil medan korrigering av temperaturen i kupén är ett mjukt sådant.

Realtidsdatabas [14]
En realtidsdatabas är en databas som med funktionalitet som garanterar maximala svarstider för transaktioner. En realtidsdatabas är en databas som exekverar transaktioner med hänsyn till tidskrav. Hårda och mjuka tidskrav kan finnas.

Relationsdatabas [15]
En relationsdatabas består vanligtvis av en databas med flera tabeller, även kallade relationer. Informationen sparas i rader i tabellerna. Idag kan data av en mängd olika typer sparas, inte bara text och numeriska data utan också bilder, ljud och video

Transaktioner [16]
När man använder en databas, i synnerhet när man gör ändringar i databasen, hör ofta en följd av operationer ihop som en enhet. Ett exempel kan vara att flytta pengar från ett bankkonto till ett annat. Då drar man först bort beloppet från det ena kontot, och adderar det sen till det andra kontot. En sådan följd av operationer som hör i hop kallas en transaktion.

Man brukar tala om ACID-transaktioner. Bokstäverna i ACID anger fyra egenskaper som transaktioner bör ha, och som man vill att databashanteraren automatiskt ska garantera:

- **A** står för **atomicitet**, dvs. odelbarhet. En transaktion ska vara odelbar. Det betyder att antingen ska hela transaktionen genomföras, så att alla ändringar den gör förs in i databasen, eller så ska inga ändringar alls göras. Om en transaktion avbryts mitt i (för att användaren ångrar sig, eller för att strömmen bryts), måste databashanteraren alltså ta bort alla ändringar som den hunnit göra.
- **C** står för **consistency preserving**, dvs. "konsistensbevarande". Om databasen är konsistent, dvs. utan några inre motsägelser, före en transaktion, så ska den också vara konsistent efter transaktionen. Det här betyder att databashanteraren måste kontrollera alla integritetsvillkor, så att de är uppfyllda.
- **I** står för **isolering**. Transaktionerna ska hållas isolerade från varandra. Även om databashanteraren utför flera transaktioner samtidigt så får en transaktion aldrig se en annan transaktions halvfärdiga ändringar. (Något förenklat. I en del system är det lite krångligare än så.)
- **D** står för **durability**, dvs. "hållbarhet". När transaktionen är genomförd och avslutad ska de ändringar den gjort aldrig försvinna ur databasen, även om strömmen går, om datorn kraschar, och helst också om disken går sönder.

Samtidighetskontroll (eng: concurrency control) för transaktioner
Vid utveckling av databashanterare finns olika metoder för att göra genomströmningen av transaktioner snabb och effektiv. Att utföra transaktionerna seriellt (en efter en) är det enda garanterat säkra sättet för att undvika konflikter i form av att man vill komma åt resurser från mer än en transaktion åt gången, men seriell hantering av transaktioner skulle göra systemet otroligt långsamt. Man låter istället transaktioner exekvera parallellt och väljer en strategi för att kontrollera och hantera konflikterna. Dessa strategier kan i grova drag grupperas som pessimistisk, optimistisk och multiversion av transaktionshantering och nedan följer en kortare förklaring av de olika varianterna [1][2]

- **Pessimistisk transaktionshantering**
  Låser resurs, utför transaktionen och släpper resursen.
  Fördel: Transaktionerna är låsta så att bara en transaktion åt gången kan komma åt dataelementet. Garanterar konsitens då många uppdaterar data samtidigt.
  Nackdel: Kan ge upphov till baklås (*deadlock*). Kan bli långsamt för att man låser resurser i onödan.

- **Optimistisk transaktionshantering**
  Kör transaktionerna utan att låsa resurser, om resurskrock inträffar så backar man och kör sedan transaktionerna på nytt.
  Fördel: Det går mycket snabbare än pessimistiska varianten då lås inte behövs ifall mest olika dataelement uppdateras.
  Nackdel: Upprepningsloop krävs om flera transaktioner försöker uppdatera samma element samtidigt (kör om, krockar och får backa för att köra om igen ifall det inte upphör att krocka).

- **Multiversion av transaktionshantering**
  Upprätthåller flera kopior av data och låta transaktioner som annars skulle blockerats att läsa gamla versioner av dataelement.
  Fördel: Inga deadlocks vid full multiversionshantering.
  Nackdel: Mycket minneskrävande.

4.1.2 Realtidsdatabas i fordonsdator

Ett modernt fordon är idag i princip helt styrt av inbyggda datorer. I takt med att funktionaliteten i fordonen ökar, blir programvaran i dessa datorer mer och mer komplex. När det gäller behovet av realtid i ett styr/kontrollsystem är det inte alltid fallet att man i sitt system verkligen behöver *strikta realtidsegenskaper* och det kan i de fallen vara fullt tillräckligt med en lösning som är *nästan* realtid. Vid val av realtidsanpassad databas är det viktigt att undersöka att systemet verkligen har anpassningar för de specifika krav som ens system ämnar hantera, vare sig det gäller snabb åtkomst av data, hög säkerhet för

datahanteringen, stöd för att spara loggar, ett väl utvecklat distribuerat system, eller annat som kan vara aktuellt.

5 TEKNIKSTUDIE: MIMER SQL REAL-TIME EDITION [5]

De databashanterare man använder i inbyggda system har ofta krav på sig vad gäller snabbhet och säkerhet. I många fall har man också begränsat med minne att tillgå i systemet. Realtidsegenskaper är ofta ett ytterligare krav på systemet och detta har gjort att forskning inom realtidsområdet är aktiv på många håll i världen, både på akademisk nivå och i näringslivet i övrigt. Delar av denna forskning beskrivs senare i denna rapport (kapitel 5.6)

Realtidsanpassningar av databassystem innebär att vissa inskränkningar i användningen av databasen kan bli aktuella att införa. Ett exempel på en sådan anpassning kan vara att kräva att användaren i förväg skall definiera de databasfrågor som skall användas i systemet (eng: predefined queries). Ett annat kan vara att storleksbestämma de filer eller tabeller där data skall lagras vid skapandet av dessa, då en utökning i storlek annars gör att en högsta svarstid blir omöjlig att garantera.

I de flesta fall då man talar om realtidsdatabaser har man valt att realtidsanpassa delar av databassystemet. Det förekommer att man kallar system för realtidsdatabaser då de är relativt snabba, t.ex. en databas med lösningen att all lagring av data hanteras i primärminnet. Detta reducerar tidsåtgången rejält då diskaccess inte förekommer, vilket annars är en tidskrävande del av transaktionshanteringen. Dessa databaser blir snabba, men har ingen garanterad högsta svarstid, vilket krävs för att vara en ”äkta” realtidsdatabas.

Mimer använder optimistisk konflikthantering för sina databastransaktioner/operationer (se definitioner 4.1.1) vilket gör att databaserna blir tidseffektiva. Optimistisk transaktionskonflikthantering innebär att lås inte används, men för realtidsoperationerna används så kallad latching [17] vilket är en lågnivå-serialiseringsmekanism för att skydda delade datastrukturer. Latches är en enklare typ av kortvarigt lås som här används just då operationens skrivandet/läsandet sker [17]. Realtidsoperationerna är prioritetsbaserade och åtkomst till enstaka celler i bufferpoolen följer den prioritetsordning som används av operativsystemet i fråga.

Mimer SQL Real-Time Edition är en produkt som är tänkt att användas i realtidsapplikationer. Den är skapad för att underlätta konstruktion av applikationer där både hårda och mjuka realtidskrav finns. Exempel på användningsområden är processautomation, styrsystem för fordon och övervakningssystem och inbyggda kontrollsystem. Mimer SQL Real-Time Edition, så som den ser ut vid skrivandet av denna rapport är tänkt att användas för uppdatering av sensorvärden. Incrementella uppdateringar, d.v.s. transaktioner i betydelsen en kedja av händelser som sker atomärt (i detta fallet: läsa ett värde-uppdatera värdet-skriva tillbaka det nya värdet) stöds inte. De hårda

realtidstransaktioner som sker i Mimer Real-Time Edition benämns hädanefter som operationer alternativt realtidsoperationer.

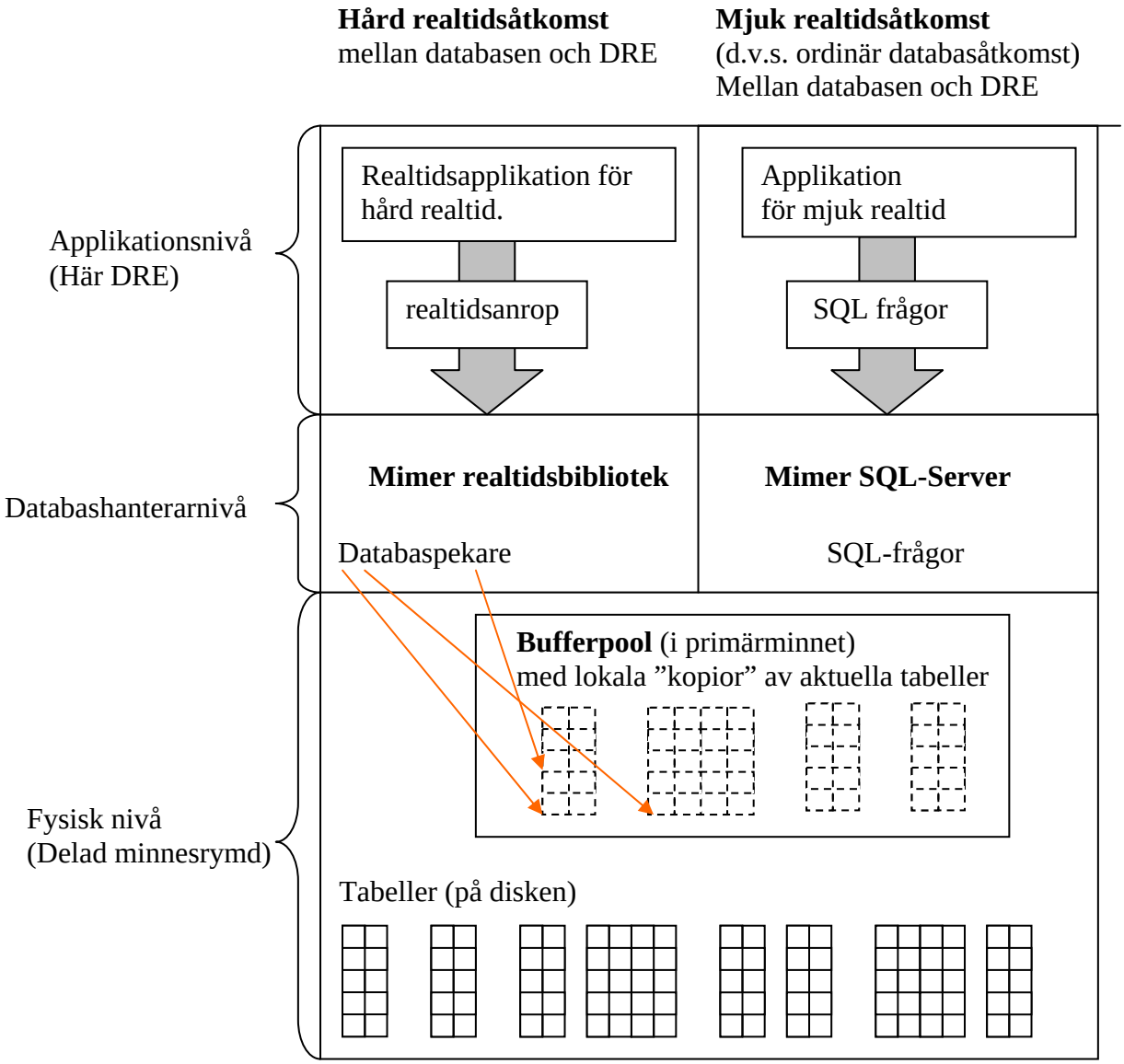
5.1.1 Intern hantering av databaspekarna

Vid hantering av databaspekare utförs endast de allra nödvändigaste kontrollerna. Vid skrivande via databaspekare som finns kopplad till en cell i tabell sker en enklare typkonvertering, kontroll av uppkopplingen och ”en latch”. Exempelvis ett värde av datatypen INT som skall skrivas via en databaspekare konverteras till intern Mimerdatatyp (MIMINT). Det utförs en kontroll att koppling till databasen finns och att bindning till aktuell cell existerar. På liknande sätt utförs en typkontroll av skrivvärdet och kontroll att värdet är inom domänen (värdeområdet) för den cell som databaspekaren är kopplad till. Då värdet skrivs används en så kallad latch. Den tid en latch tar i anspråk är mycket kort och predikterbar, och dess policy ärvs av operativsystemet. I fallet med Windows CE blir det immediate inheritance-protokollet som används.

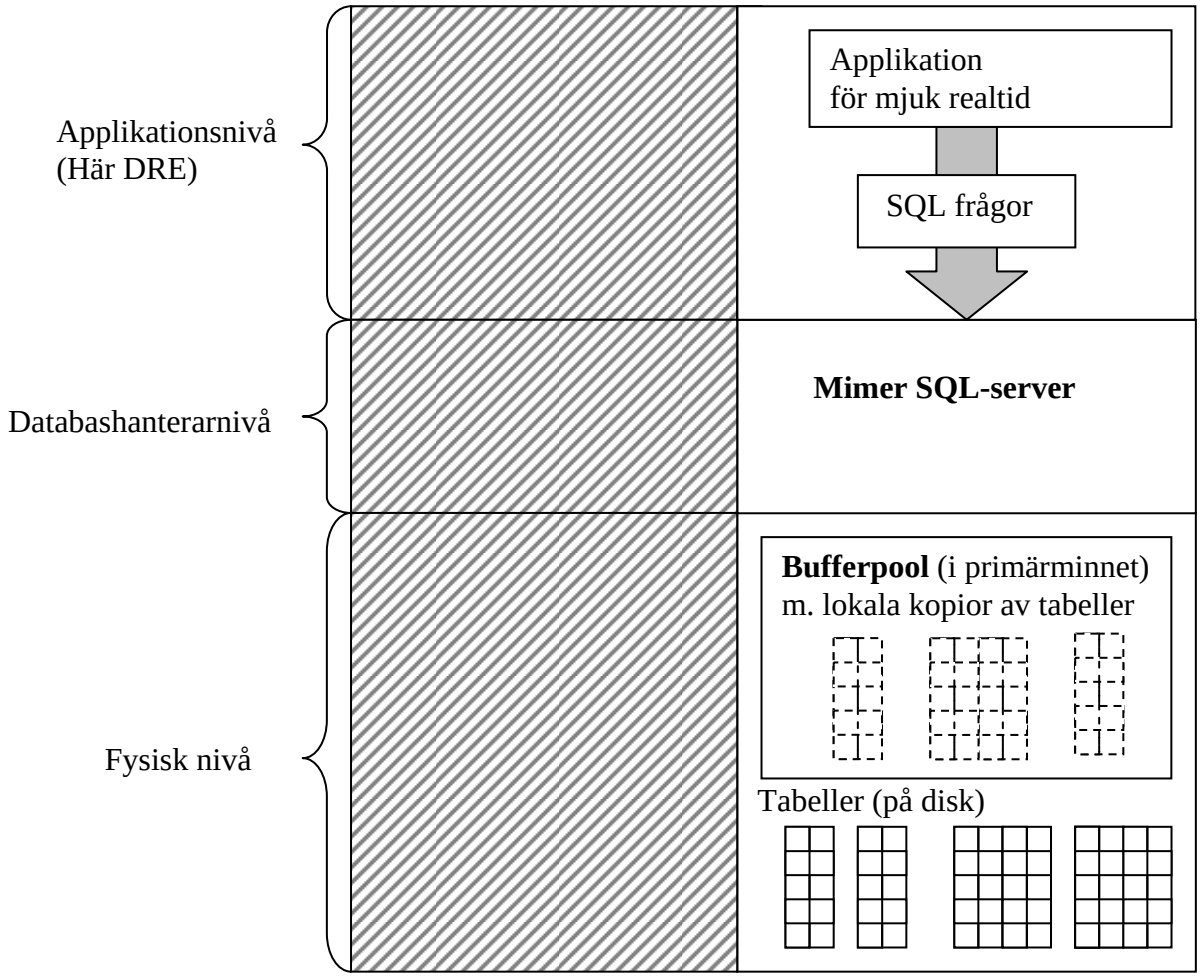
5.1.2 ACID-egenskaper i MimerRT

De operationer som i nuläget är de ”hårda transaktioner” som förekommer i Mimer Real-Time Adition garanterar ACID-egenskaperna men då transaktionshantering inte finns inplemanterat garanteras inte ACID-egenskaperna för transaktioner (Med transaktion menas här flera operationer i rad. Exempelvis: läs värde, ändra värdet och skriv värdet). ACID-egenskaperna finns mer utförligt beskrivna i kap 4.1.1 och i korthet betyder de att atomicitet, konsistensbevarande, isolering och hållbarhet (d.v.s. persistent sparande av data) skall gälla för transaktioner i en databas. Införandet av transaktioner kommer att förbättra databasen vad gäller säkerhet och korrekthet men förmodligen också medföra något längre tidsåtgång för att upprätthålla ACID-egenskaperna.

5.2 Gränssnitten för hård och mjuk realtid



Figur 8. Arkitekturen för Mimer SQL Real-Time Edition. Gränssnitten för hård och mjuk realtid kan användas parallellt.



Figur 9. Arkitekturen för Mimer SQL Mobile (d.v.s. utan realtidsanpassning).

Mimer SQL Real-Time Edition tillhandahåller två gränssnitt. Ett för hård och ett för mjuk realtid och dessa kan användas parallellt i applikationen (se figur 8 och figur 9).

5.2.1 Mjuk realtid

Den mjuka realtiden motsvarar all funktionalitet som en ordinär mimerdatabas representerar och erbjuder klassisk åtkomst till databasen där man kan manipulera och komma åt större mängder data åt gången men utan att kunna garantera svarstider för de olika operationerna. Denna åtkomst sker via vanliga SQL-kommandon (Figur 9). Detta förfarande omöjliggör den förutsägbarhet man kräver i en realtidslösning.

I fallet med mjuk realtid sker vid en SQL-fråga en bearbetning av frågan; rätt ställe i databasen skall lokaliseras och detta kräver genomsökning av en eller flera

tabeller inklusive eventuella jämförelser av olika celler i tabellerna. Detta gör att en SQL-fråga blir en förhållandevis långsam operation. Arbetsgången för SQL-frågan är inte förutsägbar då det från gång till gång kan ta olika lång tid att utföra sökning i databasen för att hitta rätt cell att uppdatera. Ett exempel på detta är bufferpoolen som innehåller aktuella datasidor (eng: data pages). Dessa datasidor flyttas till och från disken beroende på vilka delar som behövs för tillfället och det blir stor skillnad i åtkomsttid om de aktuella sidorna för just den efterfrågade tabellen befinner sig i bufferpoolen eller behöver hämtas från disken.

5.2.2 Hård realtid

MimerRTs hårda variant av realtid ger däremot tidsmässigt förutsägbar och snabb åtkomst till individuella element i databasen. Detta sker genom skapandet och användandet av databaspekare. En databaspekare binds till en enskild cell i databasen med hjälp av ett ordinärt SQL-kommando. De tidskrävande stegen med lokalisering av rätt cell sker endast vid ett tillfälle, då databaspekaren binds till cellen. När denna bindning skett placeras och låses hela tabellen i primärminnet vilket gör att efterföljande arbete med att läsa och skriva till cellen går väldigt snabbt. Inga data behöver flyttas till och från disken. Databaspekare skapas dynamiskt under programmets körning och kan också tas bort under pågående körning. Skapandet och borttagandet av databaspekare sker i mjuk realtid, medan användandet (läsande och skrivande) sker i hård realtid. Denna lösning gör att man, till skillnad från det mjuka realtidsfallet, kan garantera en svarstid för operationerna.

5.3 Arkitekturen

Mimer SQL Real-Time Edition kan beskrivas med hjälp av dessa delar:

- **Mimer SQL server**
  Databasservern hanterar den *mjuka* databasuppkopplingen. Detta sker med SQL-kommandon som ger upphov till transaktioner (se Figur 8).

- **Mimer realtidsbibliotek för hård realtid**
  Mimer SQL Real-Time Edition-gränssnittet (API = Application Program Interface) som används av programmeraren för att införa databaspekarna i sitt system (se Figur 8).

- **Bufferpool**
  Den delade minnesrymd där kopior av aktuella tabeller finns representerade då de har databaspekare kopplade till sig. Denna minnesrymd används för både den hårda och mjuka realtidsdelen av databasen (se Figur 8).

- **Databanker/datafiler**

De filer som innehåller all data (tabeller) i databasen (se Figur 8).

5.3.1 Algoritmer

Den algoritm som ligger som grund för realtidslösningen kallas 2V-DBP (two-version database pointer algoritm) och beskrivs i en avhandling; Database Pointers - Efficient and Predictable Data Access in Real-Time Control-Systems [4], och här följer en sammanfattning av algoritmen och avhandlingens jämförelse med en algoritm 2PL-HP (two-phase locking high priority) som tidigare används i liknande sammanhang. 2V-DBP-algoritmen tillåter hårda databasoperationer att exekvera utan att bli blockerade av mjuka transaktioner. De mjuka transaktionerna som använder realtionsdelen av databasen tillåts dock exekvera utan att blockeras eller avbrytas av de hårda operationerna. För att åstadkomma denna lösning låter man två versioner av alla realtiddata (d.v.s. data som har databaspekare kopplat till sig) existera i databasen. Införandet av 2V-DBP ökar effektiviteten hos systemet, och de prestandamätningar som utförts som en jämförelse med 2PL-HP visar att 2V-DBP har mycket lägre andel avbrutna (aborted) transaktioner/operationer. Det minskar från 75% till 25% i medel om man övergår till att använda 2V-DBP. Vidare konstaterades en minskning i högsta observerade svarstid för hårda operationer från 175 till 27 microsekunder.

Jämförelserna visar alltså att både mjuka transaktioner och hårda realtidsoperationer får bättre prestanda med den nya algoritmen. Den enda försämringen i prestanda som framkom i jämförelsen är att 2V-DBP tar något mer minnesutrymme i anspråk för sin hantering av den extra version av dataelement som används i hård realtid. Ökningen i minneskrav ligger på ca 6.8% vilket kan ses som en låg siffra i sammanhanget.

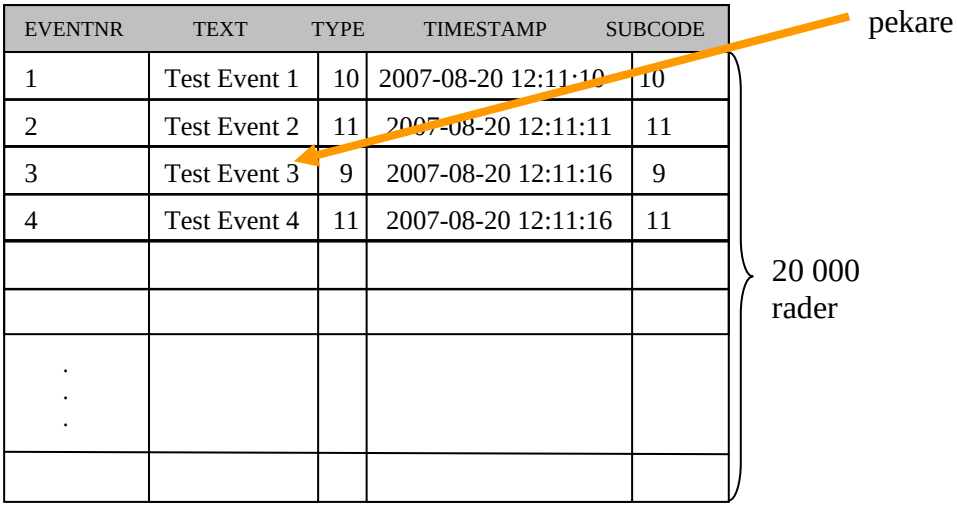
5.4 Realtidsbiblioteket

5.4.1 Databaspekare [5]

En databaspekare binds till en enskild cell i databasen med hjälp av ett ordinärt SQL-kommando enligt Figur 10 (För anropsdefinition se Figur 11)

```
MimerRTBindDbp(&pekare,"select TEXT from EVENTDATA where EVENTNR=3");
```

EVENTDATA



Figur 10. Exempel på hur databaspekare ”pekare” binds till cell i databastabell.

Följande kriterier måst vara uppfyllda för det SQL-kommando som skall identifiera det ställe i databastabell dit databaspekaren skall kopplas

- Peka ut exakt en cell.
- Inte peka på en primärnyckel.
- Inte peka ut ett indexerat attribut.
- Inte innehålla några “placeholders”. I detta exempel ät frågetecknen placeholders: SELECT type FROM Eventdata WHERE eventnumber=?
- Peka på en relation som inte är komprimerad.

5.4.2 Realtidssession [5]

För realtidsuppkoppling mot databasen skapas en *session* som är aktiv under hela den tid som realtidsaktiviteter pågår. När körningen skall avbytas avslutas sessionen och de delar av servern som hanterar realtid stängs av.

5.4.3 Realtidstråd

Realtidsaktiviteterna sker under körningen i ett *task*. Ett task körs i en delad adressrymd. Mimer SQL Real-Time Edition har ingen egenutvecklad trådhantering utan använder operativsystemets trådmodell som används på just den aktuella plattformen. Programmeraren av realtidsapplikationen skapar i den ett realtidstask per intern tråd som skall använda sig av realtidsgränssnittet. Flera task kan ha databaspekare till samma cell i databasen.

Anrop	Beskrivning
MimerRTBeginSession	Initialiserar realtidskärnan och servern.
MimerRTEndSession	Stänger realtidskärnan och realtidsdelarna av servern.
MimerRTInitializeTask	Tillverkar en realtidstråd för databasåtkomst.
MimerRTShutdownTask	Stoppar databasåtkomsten för en realtidstråd.
MimerRTBindDbp	Skapar en ny databaspekare och binder den till en specifik cell i databasen.
MimerRTDeleteDbp	Tar bort en databaspekare.
MimerRTReadInt MimerRTReadShort MimerRTReadDbl MimerRTReadString MimerRTReadTimeStamp	Läser från en databaspekare: 32-bit integer 16-bit short 64-bit floating point Null terminated string TIMESTAMP
MimerRTWriteInt MimerRTWriteShort MimerRTWriteDbl MimerRTWriteString MimerRTWriteTimeStamp	Skriver till en databaspekare: 32-bitars integer 16-bit short 64-bit floating point Null terminated string TIMESTAMP

Figur 11. Realtidsblioteket för Mimer SQL Real-Time Edition ([5]).

5.4.4 Exempel på anropssekvens för en realtidssession

MimerRTBeginSession
  MimerRTBeginTask
    MimerRTBindDbp
      [*Här utförs läs- och skrivoperationer med den/de databaspekare som bundits*]
    MimerRTDeleteDbp
  MimerRTShutdownTask
MimerRTEndSession

5.4.5 Returkoder

Vid realtidsanrop returneras returkoder och utifrån dessa kan man få information om ett anrop gått rätt eller fel. De returkoder som används är Mimers standardreturkoder med tillägg av flera nya returkoder som är specifika för realtidslösningen. Förklaringar till dessa finns i dokumentationen för produkten [5].

5.5 Bufferpoolen

I normalfallet för Mimers databaser så är bufferpoolen ett delat primärminne och innehåller datasidor (eng: pages) med kopior av aktuella databastabeller. Bufferpoolen ligger i processens virtuella adresseringsutrymme och konkurrerar

då på lika villkor med alla andra aktiva processer om det fysiska minnet. Det som inte får plats *i* RAM flyttar operativsystemet till disken [7].

I fallet med realtidsdatabasen sker all hantering i primärminnet och ingen skrivning till disken görs. Man *låser* hela bufferpoolen i primärminnet för att skriva till disken skulle göra det omöjlig att behålla realtidsegenskaperna. En längsta transaktionstid skulle då bli omöjlig att garantera. Alla tabeller med databaspekare kopplade till sig placeras i bufferpoolen och stannar där ända tills den sista av dess databaspekare tagits bort från tabellen i fråga.

5.6 Bakomliggande forskning

Utvecklingen av Mimer SQL Real-time Engine sker hos Mimer i Uppsala, i samarbete med Mälardalens högskolas avdelning för realtidsforskning; Mälardalens Real-Time Research Center (MRTC) och har ännu inte släppts som färdig produkt. I skrivande stund är en release av en första version planerad kring årets slut (2007). En första officiell presentation av den nya realtidsadatabasen har skett på Embedded Conference Scandinavia 2007, under tekniska mässan i Älvsjö den 16 oktober.

Bakomliggande arbete som lett fram till utvecklandet av Mimer SQL Real-Time Edition finns bland annat beskrivet i en avhandling om fordonssystem (*Data Management in Vehicle Control-systems* [4]) skriven av Dag Nyström, som också är med i utvecklingsgruppen för Mimer SQL Real-time Edition. Ett par algoritmer har utarbetats och dessa beskrivs nedan.

5.6.1 Avhandling om fordonsstyrsystem

Dags Nyströms avhandling om fordonssyrsystem presenterar koncept för utvecklandet av en realtidsdatabas med inbyggda kontroll/styrsystem med hårda realtidskrav i åtanke. I avhandlingen beskrivs arbetet med att utreda den då existerande marknaden för inbyggda styrsystem till fordon, både området realtidsforskningsplattformar och området kommersiella plattformar för inbyggda system undersöks. Avhandlingen beskriver ett urval av forskningsplattformar som är anpassade för realtid, och det visar sig att de är inriktade mot mer storskaliga applikationer. Detta gör att de inte är optimala för användning i inbyggda resursbegränsade miljöer. Den sammanställning och jämförelse av några utvalda kommersiella produkter visar att det inte heller där finns en väl anpassat produkt för en tillämpning i ett just så resursbegränsat och tidskritiskt (realtidskrävande) system som ett fordonsstyrningssystem kan sägas vara. De undersökta databaserna och forskningsplattformarna från ovan nämnda avhandling [4] finns beskrivna senare i detta kapitel men en sammanfattning av undersökningen följer redan här:

- **De kommersiella produkterna har inte tillräckligt strikta realtidsegenskaper för att användas i fordonsstyrsystem.**
  Inga av produkterna kan garantera realtidsegenskaperna och trots att flera av produkterna har väldigt hög accesshastighet för data så kan inte realtid garanteras. Oförutsägbara transaktionshanteringsfördröjningar kan inträffa, vilket inte är acceptabelt i riktiga realtidssystem.

- **De undersökta forskningsplattformarna för realtid är inte skapade för inbyggda system.**
  Målet med de plattformarna har inte varit att snåla på resurser på ett sådant sätt som ett inbyggt system kräver. Någon av dem kräver två processorer för uppdelning av hård och mjuk realtidstransaktioner, en annan kräver en spegelnod i sin lösning, och en tredje använder sig av semaforskyddad central databasfil. Alla dessa lösningar är resurskrävande.

Traditionella databashanterare passar inte i fordonsstyrningssystem där man i många fall kräver en garanterad högsta svarstid. Minneskapaciteten är i de flesta inbyggda styrsystem begränsad så att en traditionell databashanterare inte ryms.

Här följer en kort beskrivning av de databaser som undersökts i ovan nämnda avhandling om styrsystem för fordon med kommentar om varför de är intressant ur ett inbyggt fordonsstryningsperspektiv:

- **Pervasive SQL,** Pervasive Software Inc.
  Denna databas har tre olika versioner för inbyggda system. För smart-card, mobila system och inbyggda system. Alla tre är integrerade med varandra på ett bra sätt och fungerar också bra med den icke-inbyggda varianten med samma namn, Pervasive SQL. Denna databas finns med i undersökningen för att den har små minneskrav.

- **Polyhedra**, Enea AB.
  Denna databas undersöks av tre anledningar, den hävdar att den är en realtidsdatabas, det är en primärminnesdatabas och den har ett aktivt databashanteringssystem.

- **RDM**, Birdstep Technology Inc.
  På samma sätt som Polyhedra (ovan) kallar sig denna databas vara en realtidsdatabas. I övrigt skiljer den sig från Polyhedrasystemet, bland annat genom att vara uppbyggd som ett inbyggt bibliotek, och alltså inte använda sig av klient/server-modellen.

- **Berkeley DB**, Sleepycat Software Inc.
  Denna databas, som också är en inbyggd biblioteksdatabas distribueras som öppen källkod och är därför intressant ur forskningssynpunkt.

- **TimesTen**, TimesTen Performance Software.
  Detta är en relationsdatabas, som Polyhedrasystemet, och likaså en primärminnesdatabas.

Kort beskrivning av de undersökta forskningsplattformarna inom realtid:

- **DeeDS**, Skövde Högskola. Sverige
  DeeDS är ett distribuerat realtidsdatabassystem som hanterar både hårda och mjuka databastransaktioner. Den använder sig av ECA-regler (för event E, om egenskap/*condition* C agera A) för att tillhandahålla en aktivt realtidsfunktionalitet och är designad för en mulitprocessorsmiljö.

- **RODAIN**, Helsingfors, Finland
  Ett strikt realtidsdatabassystem som är huvudsakligen skapat för telekommunikation. Det är designat för hög tillgänglighet men är tolerant mot fel. Den är anpassad till att passa strukturen hos telekommunikationstransaktioner mer korta köer och snabba uppdateringar blandat med mer långsamma massiva uppdateringar.

- **ARTS-RTDB**, University of Virginia, Charlottesville.
  Produkten är byggd som en utökning av ARTS realtidsoperativsystem och hanterar både hårda och mjuka realtidstransaktioner. Den använder sig av en algoritm som på engelska kallas *imprecise computing* [18] för att garantera realtidskraven för databasen. (Grundidén för *imprecise computing* är att använda sig av det delresultat som finns färdigt vid tiden för transaktionens *deadline* och sedan låta det slutliga resultatet inkomma då beräkningen är helt genomförd)

I avhandlingen beskrivs också ytterligare forskning som utförts som att samarbete mellan Mälardalens Högskola och Linköpings Universitet och den beskrivs i korthet nedan.

5.6.2 Forskningsprojektet COMET (a Component-Based Real-Time Database for Automotive Systems) [4]

En komponentbaserad realtidsdatabas för fordonssystem. Denna databas skapas med hjälp av ett verktyg som möjliggör för utvecklare att designa och analysera systemet under utvecklingens fortskridande. Olika komponenter kombineras till en skräddarsydd lösning som på så sätt blir specialanpassad för just den tillämpningen. Varje skapad COMET-databas innehåller endast funktioner som krävs för just den nod den körs på.

COMET-projektet tar hänsyn till de ofta begränsade systemresurser som finns i inbyggda system, då specialanpassningen innebär att man utelämnar överflödig funktionalitet. COMET-projektet ligger till grund för utvecklandet av Mimer SQL Real-Time Edition.

6 DEMONSTRATORN

Demonstratorn skall visa skillnader i prestanda mellan DRE i originalutförande (demoStandard) och dess modifierade variant med databaspekare (demoRealtid). Realtidspekare införs inte i hela systemet utan endast i en av tabellerna i databasen. I demonstratorn mäts hur lång tid det tar att lägga in event i databasen med demoStandard och demoRealtid. Vidare undersöks hur mycket arbete i form av ändringar i DRE och databasscriptet som skulle vara nödvändiga för att lägga in realtidsdatabasen i hela systemet.

Utvecklingsmiljö: Visual Studio 2005
Databasverktyg: DB Visualizer, Mimer Controller, Mimer Administrator
Utvecklingsspråk: C++ och Mimer SQL
Dator: HP Compaq, Intel Pentium 4, 3.20GHz, 512 MB RAM

6.1 DRE med och utan MimerRT

6.1.1 DRE utan realtidsanpassning - demoStandard

Då ett nytt event genereras i systemet läggs det in i databasen enligt följande:

- Ett event inträffar och läggs in i den kö som finns för ändamålet (A i Figur 13*).* Den kö som används i DRE är Deque [20], vilken är en STL(Standard Template Library)-sekvens för hantering av element. Strukturen liknar en vektor fast med snabb åtkomst till enskilda element inuti strukturen och med möjlighet att effektivt lägga till och ta bort element både i början och slutet av sekvensen.

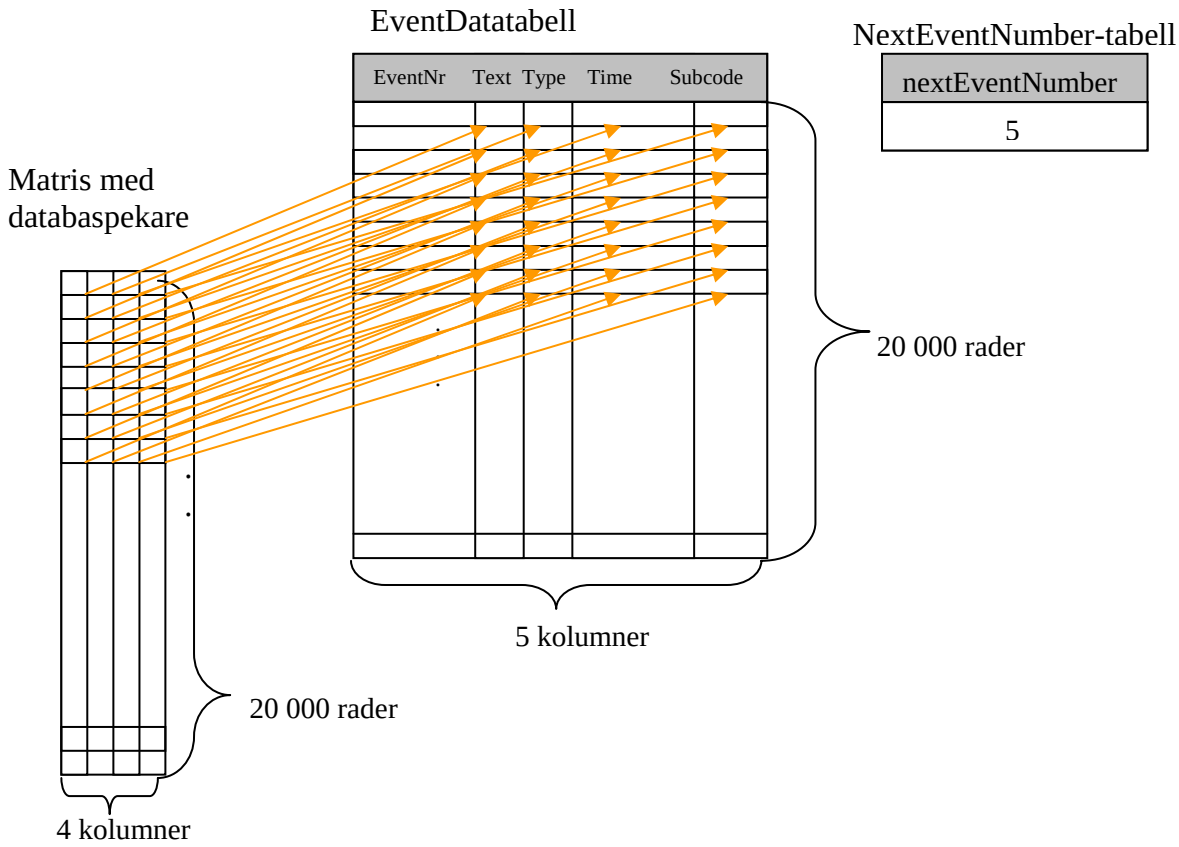
- Ett så kallat förkompilerat statement har tillverkats då systemet initierades (startades) och till ett sådant statement skriver man att antal parametrar (B i Figur 13*).* Att använda sig av förkompilerade statements innebär att man definierar vilka SQL-frågor man kommer att ställa till databasen. Detta ger den fördelen att SQL-servern slipper kompilera varje fråga på nytt vid varje anrop. I DRE fungerar denna lösning bra då det enbart förekommer i förväg uttänkta operationer för interaktion med systemet, det vill säga att användaren aldrig producerar några *egna frågor* till databasen.

- Köade statement lagras och skrivs in i databasen med jämna mellanrum. Att lägga in eventen kontinuerligt utan att använda sig av köer skulle göra systemet för långsamt då alltför frekvent diskskrivande behöver göras. Med nuvarande lösning köas eventen kontinuerligt och skrivning (eng: commit) av köade event görs var 100e millisekund.

6.1.2 DRE med realtidsanpassning - demoRealtid

Hela eventdatatabellen får databaspekare kopplade till sina rader. Alla reltidspekarbindningar görs vid uppstart och placeras i en datastruktur (matris) och pekar då på alla fält i fyra av de fem ingående kolumnerna i EventData-tabellen. Databaspekare kan inte peka till fält som är tabellens primärnyckel, i detta fall den första kolumnen EventNumber. Pekarna används under hela programmets körning, och först när det programmet avslutas tas de bort. Läs- och skrivdelen kommer att motsvara den i DRE förekommande metoden AddEvent som används för att lägga in händelser (Event) i databasen.

Realtidspekarna återanvänds under hela körningen på samma cirkulära sätt som sker för eventtabellen i grundsystemet. Den aktuella raden där nästa event skall läggas in hanteras med hjälp av databaspekare till en minitabell, nextEventData, i databasen. I den tabellen finns endast ett fält, och vi läser och skriver till det fältet kontinuerligt under körning (se Figur 12).



Figur 12. Matrisen med realtidspekarna och den databastabell, EventData, som pekarna är kopplade till.

6.2 Arbetsgång för införandet av MimerRT i DRE

6.2.1 Installera Mimer SQL Real-Time Edition

Installerar databasen jag fått från Mimer. Enligt *release notes* [6] från Mimer skall några justeringar göras, för att vissa funktioner inte finns implementerade och kan ske per automatik i MimerRT ännu. En kortare beskrivning av justeringarna följer här. För med detaljerad information hänvisas till release notes [6].

- En global miljövariabel skapas och skall ha samma namn som databasen.
- Instruktionerna för att sätta upp ett nytt projekt i Visual Studio följes, men i detta fall använder jag ett existerande projekt (DRE) så alla delsteg används inte. Bland annat lägges filerna MIMRTAPI.LIB och MIMRTAPI.H in i arbetskatalogen i Visual Studio och det länkas till libfilen. Varje fil som använder MimerRT inkluderar h-filen
- Regedit startas för att i register ändra antalet tillåtna databasepekare från defaultvärdet 2 000 till 100 000 i detta fall.
- Inloggningsnamn och lösenord skall vara samma för anropet MimerRTBeginSession(L”Sysadm”, L”Sysadm”, 3);

6.2.2 Skapa en ny databas

Lägger in vår DREdatabas genom att i databasens kommadofönster (BatchSQL) läsa in det databasscript som används för det ursprungliga systemet. Fyller databasens tabeller med dummydata från installationsfil. Vissa modifieringar av scriptet kommer att utföras för att motsvara speciella behov hos MimerRT. Tabeller får inte vara komprimerade och databashanteraren får inte utföra loggning i sina datafiler. Mer om detta i kapitlet om problem och lösningar på dessa (se 6.3)

6.2.3 Använda realtidsgränssnittet i DRE

Gör en testmiljö i form av ett litet projekt i Visual Studio. Provar de olika anropen för hård realtid först kopplat till liten testdatabas och senare till den ordinarie ovan (punkt ett) beskrivna databasen. När det fungerar för det lilla projektet övergår jag till att realtidsanpassa det större projektet (DRE). Nedan beskrivs i punktform de förändringar av DRE som gjorts för att införa MimerRT. Då endast en av tabellerna anpassas till att använda MimerRT låter vi alla övriga anrop och koppling via ordinär SQL-kommandon vara kvar som de är och dessa räknas som mjuka realtidsoperationer.

- **RtPointerHotel.cpp**
  Min egen klass som hanterar MimerRTanpassningen. Denna klass hanterar den matris (hotel) som skapas och upprätthålls samt skapar och avbryter kontakten med realtidsdatabasen. Följande funktioner finns i klassen:

RtPointerHotel()	Konstruktor
~RtPointerHotel()	Destruktor

PopulateHotel()	Binder databaspekare
AddEventIntoHotel(Event &e)	Lägger till ett event
SetupRtConnection()	Startar Session och Task
EndRtConnection()	Avslutar Session och Task

RtPointerHotel.h
Tillhörande h-fil inkluderar RTbiblioteket och definierar användbara

- **SystemManager.cpp**
  1) Init-funktionen är den som startar DREs olika delsystem och det är här jag lägger in anropen till RtPointerHotel

  2) Uppkopplingen mot databasen för vanlig SQL-åtkomst sker med hjälp av en sträng med information om uppkopplingen. Denna sträng ändras så att den kopplar upp sig mot den nyskapade realtidsdatabasen (men dess mjuka realtidsvariant). Det som behöver ändras är:
  - DATABASE, databasens namn till *RTdatabas*
  - UID, användarnamnet till *sysadm*
  - PWD, lösenordet till *sysadm*

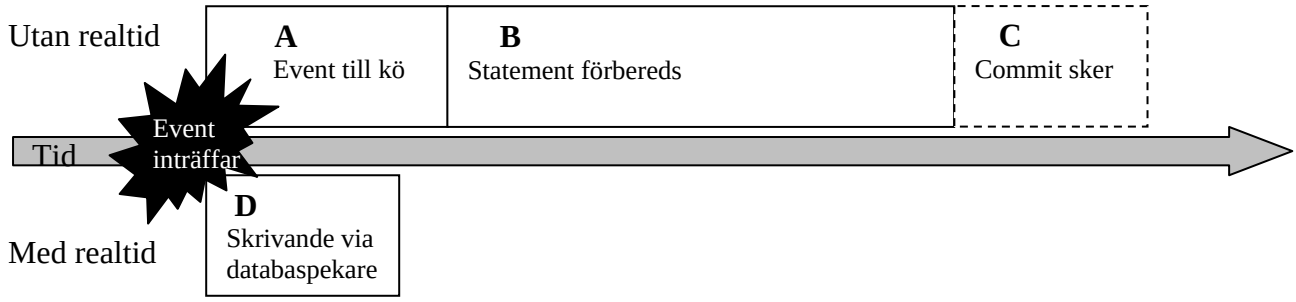
```
const STRING DATABASE_CONNECTSTRING =
_T("DRIVER=Mimer;DATABASE=RTdatabas;UID=SYSADM;PWD=SYSADM;PR
OTOCOL=LOCAL;");
```

  3) Jag lägger in NO_EVENTS_TO_ADD antal event via AddEventIntoHotel(Event) och förbereder tidsmätning av detta.

- **EventMgr.cpp**
  Letar upp det ställe där Event läggs till; AddEvent(Event). Ändrar så att AddEvent inte lägger Event i kö utan istället anropar AddEventIntoHotel(Event).

6.3 Prestandajämförelser

För att kontrollera skillnader i prestanda utförs två olika tester som båda utgår från operationen att lägga till Event i systemen:



Figur 13. De olika momenten för ett nytt event som inkommer och läggs till i systemet (med och utan realtidanpassning).

demoStandard (se Figur 13)
A – Event läggs i kö
B – Ett SQL-statement förbereds
C - Commit utförs var 100e millisekund, och de köade statementen skrivs då till disken.
Begränsning: Då commit saknas i demoRealtid räknas tiden för detta inte med i jämförelsen.

demoRealtid (se Figur 13)
(Initiering av pekare utförs vid uppstart av systemet)
D – Skrivande via databaspekare
Realtidslösningen innebär inget arbete med att lägga event i kö och förbereda ett statement för varje event.
Begränsning: Det tidskrävande arbetet med att lokalisera var data skall läggas in har utförts då databaspekaren skapades vid initieringen av applikationen och påverkar sedan inte tiden för varje addEvent. I resultatet tas denna tid inte med.

6.3.1 Prestandatester

1) **AddEvent**
   Mäter hur lång tid det tar att lägga in (logga) event, med och utan databaspekare. Jag testkör båda systemen vid skapandet av olika antal event (5 000, 10 000 och 15 000) och sparar medelvärdet av resultatet och visar det i tabell/diagramform
2) **Enskilda svarstider för AddEvent**
   Mäter hur lång tid det tar att lägga in varje enskilt event för att se hur stor spridning det är på tidsåtgången. Sparar resultaten och visar dem i tabell/diagramform.

6.4 Problem som uppkommit under arbetets gång och hur de hanterats.

Här beskrivs problem som uppkom i samband med arbetet med att införa MimerRT i DRE, samt hur dessa problem lösts.

Problem
Den bifogade filen MIMRTAPI.H var skriven för att fungera i ett C-projekt. För att den skulle fungera i vårt C++projekt behövdes en del ändringar göras.
Lösning
Handledare Carl-Magnus kände till problemet och kunde lösa det.

Problem
Felkoden ges för anropet MimerRTBeginSession men felkoden gav inte helt relevant information i det här fallet.
Lösning
Min skapade databas hade typen Engine (istället för Realtime) vilket Mimer hjälpte mig att hitta (och har därefter ändrat felkoden)

Problem
Felkod vid bindning av databaspekare;
MimerRTBindDbp(`&rt_pointer_read, L"SQLSTATEMENT");`
Lösning
Felet uppkommer på grund av att tabeller som skall ha databaspekare får inte använda sig av komprimering. Föjlande rad skrives in i databasscriptet för att förhindra komprimering:

```
ALTER TABLE EventData SET COMPRESS OFF;
```

Problem
Icke-deterministiskt beteende med att programmet kraschar varannan eller var tredje körning.
Lösning
Loggning skall vara avstängt för datafilerna (kallas databanker i Mimer). Man stänger av loggning på detta sätt och dessa rader finns inlagda i databasscriptet:

```
SET DATABANK develop OFFLINE;
SET DATABANK develop ONLINE PRESERVE LOG;
```

Problem
MimerRTWriteTimeStamp och MimerRTReadTimeStamp finns med som anrop i MIMRTAPI.H men är inte implementerade.
Lösning
Mimer får bakläxa på detta, löser det och skickar mig en ny version av MimerRT.

Problem
Jag önskar mig mer dokumentation som beskriver MimerRT och tekniken bakom denna, för att utöka avsnittet om teknikstudie i rappporten. MimerRT har knapphändigt med dokument eftersom den inte är släppt som färdig produkt ännu.
Lösning
Jag läser den HTML-hjälpfil som jag fått med MimerRT samt technical description för *vanliga* Mimer Engine, vilken är den databas som MimerRT är en utökning av. För ytterligare information och tekniska frågor inför rapportskrivandet mejlkorresponderar jag med Dag Nyström och Anders Eriksson på Mimer.

Problem
Databasen har inte samma lösenord som användarnamn, och jag har problem med att hitta hur det skall ändras.
Lösning
Lars, som tillverkat originalscriptet för databasen i DRE föreslår att jag istället raderar hela databasen och lägger in den på nytt för då väljer man nytt inloggningsnamn och lösenord.

Problem
Vill under projektets gång växla mellan två olika databaser (originalet och mindre testversion av den). Detta skapar bekymmer då jag för varje byte mellan databaserna behöver gå in och ändra miljövariabelns namn.
Lösning
För att underlätta databasbytet skapar jag en ny användare i originaldatabasen och låter den nya användaren skapa testversionen. På detta sätt kan jag växla mellan de olika databaserna utan att ändra miljövariabeln.

```
create ident PU as user using 'PU';
```

Problem
I demoStandard skapar databasen själv ett timestamp då ett nytt event läggs in. I demoRealtid skall vi själva skriva ett timestamp via databapekare.
Lösning
Hämtar tiden via C++strukturen SYSTEMTIME och bygger ihop ett eget timestamp. Ser till att det har rätt format och rätt antal decimaler för att kunna skrivas via pekaren till timestampcellen. (2007-10-05 13:47:14.734000)

Problem
Vid bindning av alla databaspekare är det en av de 80 000 pekarna som ger en felkod -16244.

```
BindDbp (31,1)   first try = -16244
BindDbp (31,1)  second try = 0
```

Lösning
Jag provar att binda om pekaren och detta fungerar i minst 99 fall av 100. Felet är oroväckande och bör åtgärdas. Möjligen kommer felet att byggas bort i och med att ny funktionalitet (loggning och atomär uppdatering) läggs in i MimerRT.

Problem
Uppstartstiden för systemet blir onödigt lång då alla 160 000 databaspekare (40 000 rader gånger 4 celler i varje rad) skall tillverkas. Tiden för att initiera alla dessa pekare tar ca en minut.
Lösning
Jag minskar antalet rader från 40 000 till 20 000 för att minska uppstartstiden. Detta påverkar inte prestandamätningarna men gör systemet lättare att hantera.

Problem
Eventdatatabellens kolumn eventNumber är nyckeln för tabellen och tilldelas unika nummer genom en sekvens. Dessa nummer förekommer alltid i stigande ordning, men ibland alltså med luckor i sekvensnummerföljden. Min lösning med databaspekarmatrisen förutsätter en kontinuerlig och ökande nummersekvens med 0 som startvärde. (i detta fall 0 till 19 999)
Lösning
I DB-visualizer kontrollerar jag manuellt att luckor i sekvensen inte finns. Om sådan smugit sig in så tas databasen bort och läggs in på nytt.

Problem
Tidsmätningen av addevent via RT ger väldigt olika tid vid mätning med exakt samma parametervärden.
Lösning
Initieringsdelen av DRE gör ibland att mina mätningar fördröjs. En *sleep* ger DRE möjlighet att bli klar med initieringen och mina mätvärden för RT blir betydligt jämnare.

7 RESULTAT OCH DISKUSSION KRING DESSA

7.1 Skillnader i prestanda

7.1.1 Tidsmätning – svarstider för AddEvent i genomsnitt

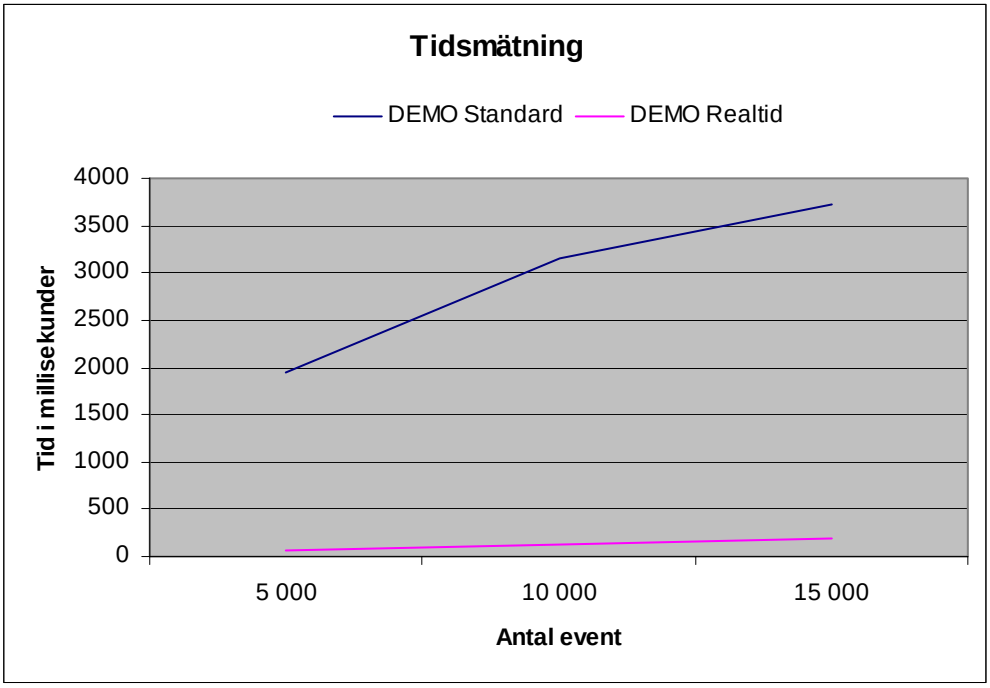
Mätningarna visar att demoRealtid är mycket snabbare än demoStandard vad gäller att lägga in event i databasen. Att lägga in ett event via databaspekare går snabbare än att lägga in samma event i kö i demoStandard (se
Figur **14**). Tidsåtgången för att lägga in event i databasen oavsett om det är med demoStandard eller demoRealtid är linjär, vilket framgår av grafen i Figur 15. Mätvärdena är medelvärden baserade på utförda tester med 5000, 10000 och 15000 event.

Tid I millisekunder

		DEMO Standard					DEMO Realtid
		A	B	C	Totalt	Totalt - C	D
1	5 000	278	1054	42	1982	1940	67
2	10 000	542	2607	84	3233	3149	126
3	15 000	856	2874	57	3787	3730	187

A, event till kö
B, statement skapas/läggs till
C, commit sker
D, Skrivande via databaspekare

Figur 14. Tidmätningens resultat för 5 000, 10 000 och 15 000 inlagda event i demoStandard och demoRealtid.

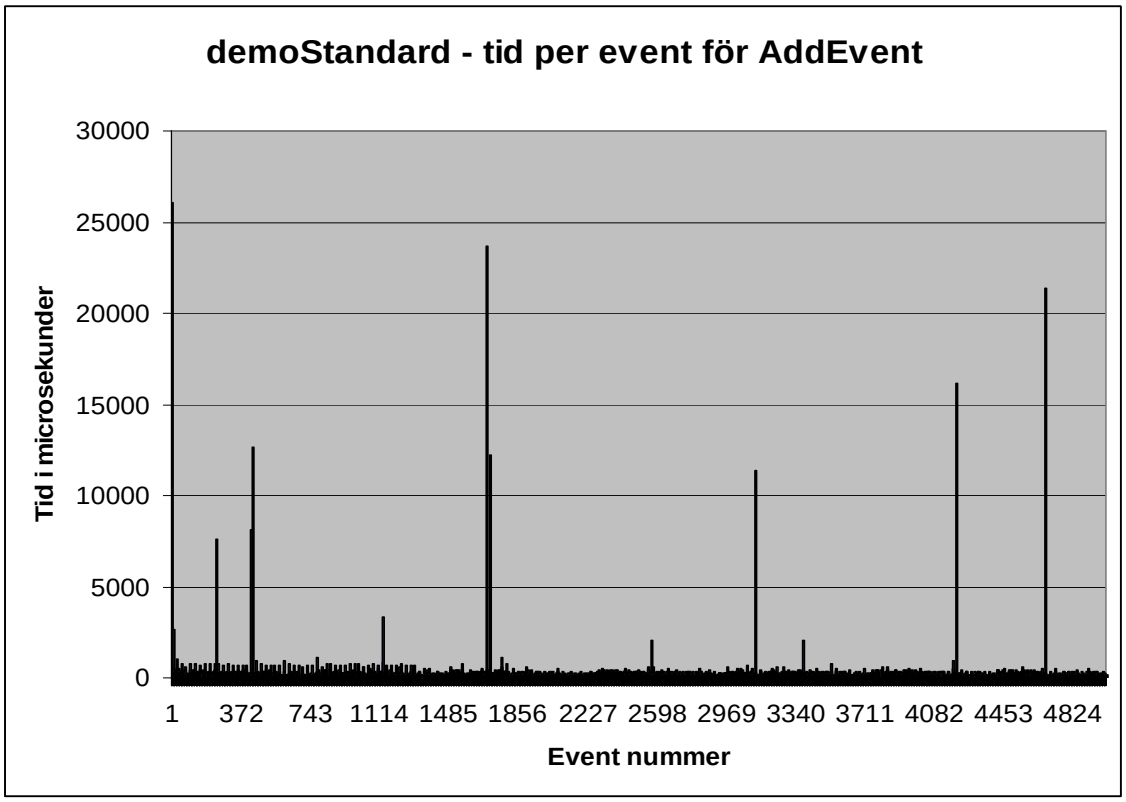


Figur 15. Tidmätningens resultat i diagramform.

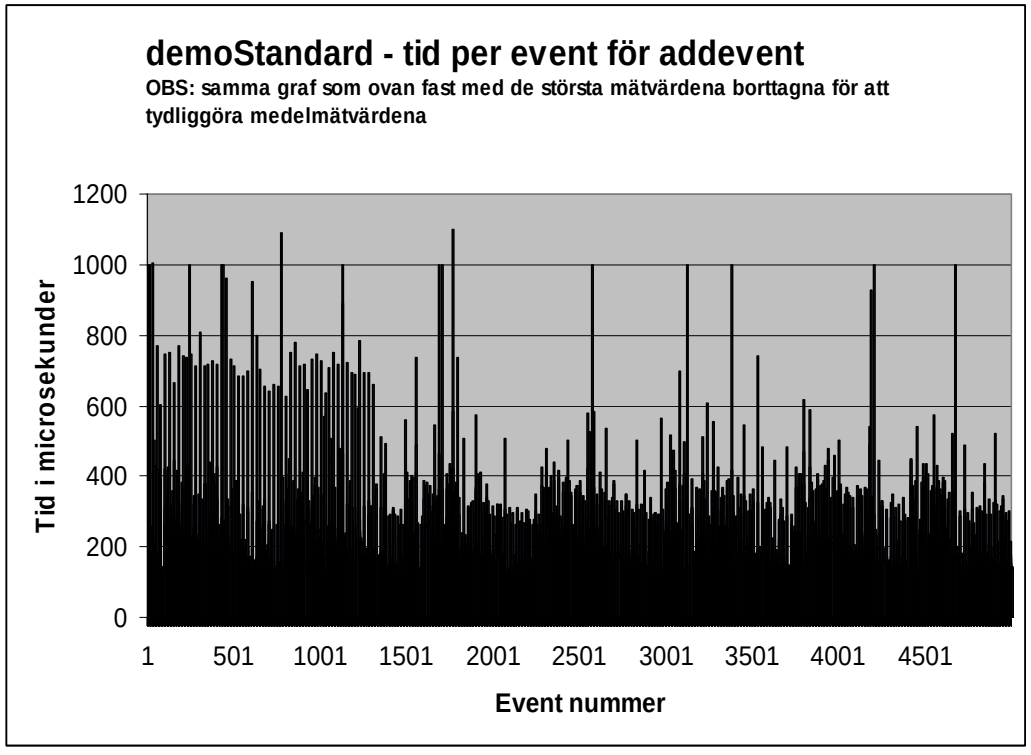
7.1.2 Tidsmätning - svarstider för enskilda AddEvent

demoStandard

Medianvärdet för att lägga in ett event via AddEvent är 240 microsekunder per AddEvent (se Figur 16). En del riktigt stora värden gör att grafen nedan är lite svår att tolka så i nästkommande figur (se Figur 17) visas samma mätvärden fast med de största mätvärdena reducerade för att öka läsbarheten.

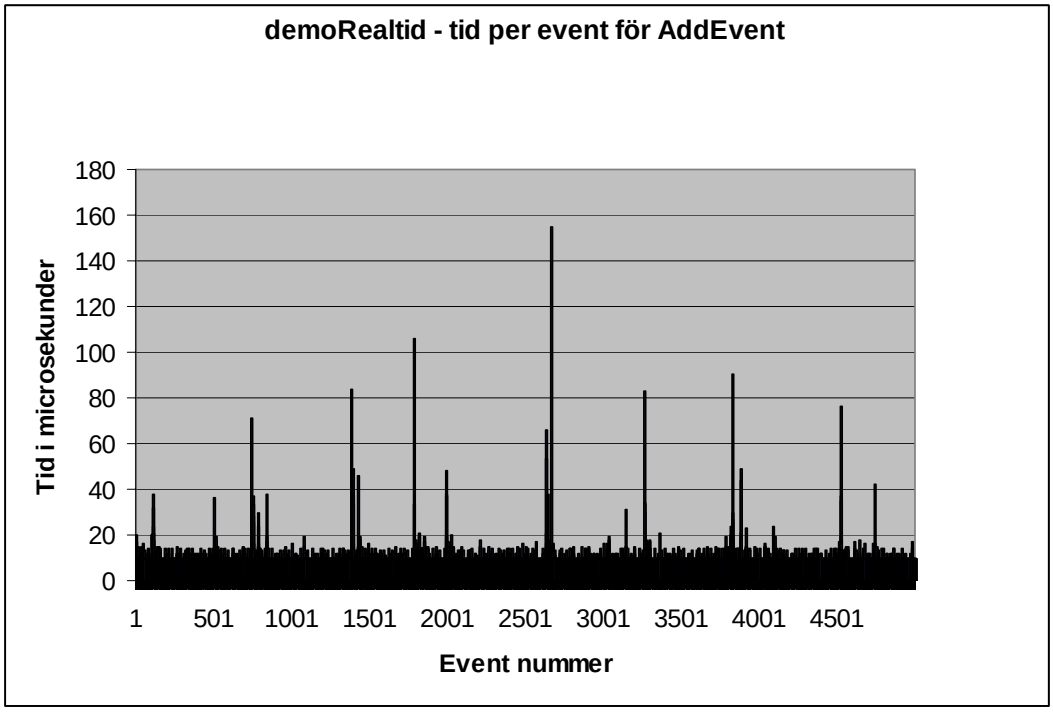


Figur 16. Diagram över tidsåtgång för 5000 utförda AddEvent i demoStandard.



Figur 17. Diagram över tidsåtgång för 5000 utförda AddEvent i demoStandard. De största mätvärdena är här borttagna för att öka läsbarheten av diagrammet.

demoRealtid
Medianvärdet för AddEvent är 10 microsekunder per event i demoRealtid. I testkörningen är det högsta värdet 155 microsekunder (se Figur 18). Anledningen till de fåtal högre mätvärdena som förekommer skulle kunna vara det faktum att mätningen utförts på ett operativsystem som inte är realtidsanpassat. I detta fall skulle fenomenet kunna uppkomma på grund av de prioriteringsproblem som kan förekomma (se 7.2.1 om prioriteringsproblem).



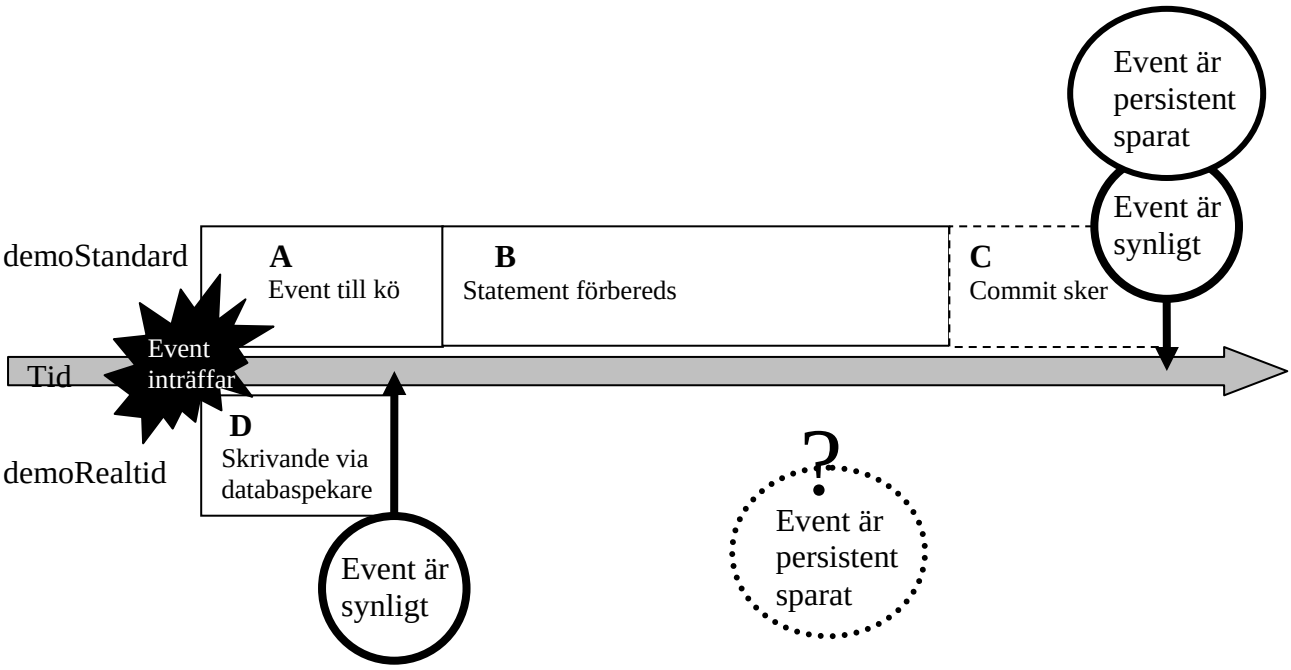
Figur 18. Diagram över tidsåtgång för 5000 utförda AddEvent i demoRealtid.

7.1.3 Persistent sparande av data

I demoStandard är data persistent sparat på disk i och med att commit skett (se Figur 19). Data som är persistent sparat försvinner inte även om strömmen skulle brytas. I fallet med demoRealtid ligger alla tabeller i primärminnet och är i det läget inte sparat till disk. En commitfunktionalitet behövs där för att göra data persistent men denna typ av funktionalitet saknas för närvarande i MimerRT.

7.1.4 Synligt data

I demoStandard blir data för det event som lagts in i systemet synligt i databasen då commit skett (se Figur 19). Det är först då som data kan nås från t.ex. andra applikationer. Då AddEvent sker i demoRealtidsfallet skrivs informationen direkt in i tabellkopian i bufferpoolen och blir då genast synligt (se Figur 19).



Figur 19. Data blir synligt snabbare i realtidslösningen, men icke sparat till disk.

7.2 Önskemål om utökad funktionalitet i Mimer SQL Real-Time Edition

Under examensarbetets gång, framför allt i samband med skapandet av demonstratorn, har det framkommit ett flertal funktionalitetskrav som inte ännu är uppfyllda i Mimer RT. Realtidslösningen så som den ser ut i nuläget lämpar sig bäst i ett system där enstaka celler i databasen skall nås och modifieras. Exempelvis vid datalagring där hela rader eller till och med flera rader i olika tabeller skall uppdateras kan Mimers nuvarande lösning med enstaka databaspekare inte utföra dessa som en transaktion. En transaktion skall garantera att flertalet egenskaper gäller i databasen, bland annat konsistens och andra ACID-egenskaper så som hållbarhet data (är persistent) och odelbarhet hos transaktionerna (se kapitel 4.1.1). Att sakna denna funktionalitet är en stor begränsning.

För att vara användbart i CC Systems DRE-system, eller liknande system behövs uppdateringar av Mimer RT på flera olika områden. Här beskrivs den funktionalitet som CC Systems önskar för sitt DRE-system och i förlängningen kommer dessa utökningar bidra till att produkten blir aktuell för liknande tillämpningar. En analys av hur en sådan anpassning skulle påverka resursanvändandet (framför allt tidsåtgången) samt Mimers eventuella planer på att införa just den anpassningen i produkten finns också med för varje punkt.

7.2.1 Portning till andra operativsystem

Resultatet i denna rapport utgår ifrån mätningar gjorda i Windows XP på en X86-processor, vilket prestandamässigt inte motsvarar Windows CE med ARM-processor som används i fordonsdatorerna.

Mimer planerar att göra en portning till Windows CE under senare delen av produktutvecklingen, troligen med start hösten 2007.

7.2.2 Prioriteringsproblem

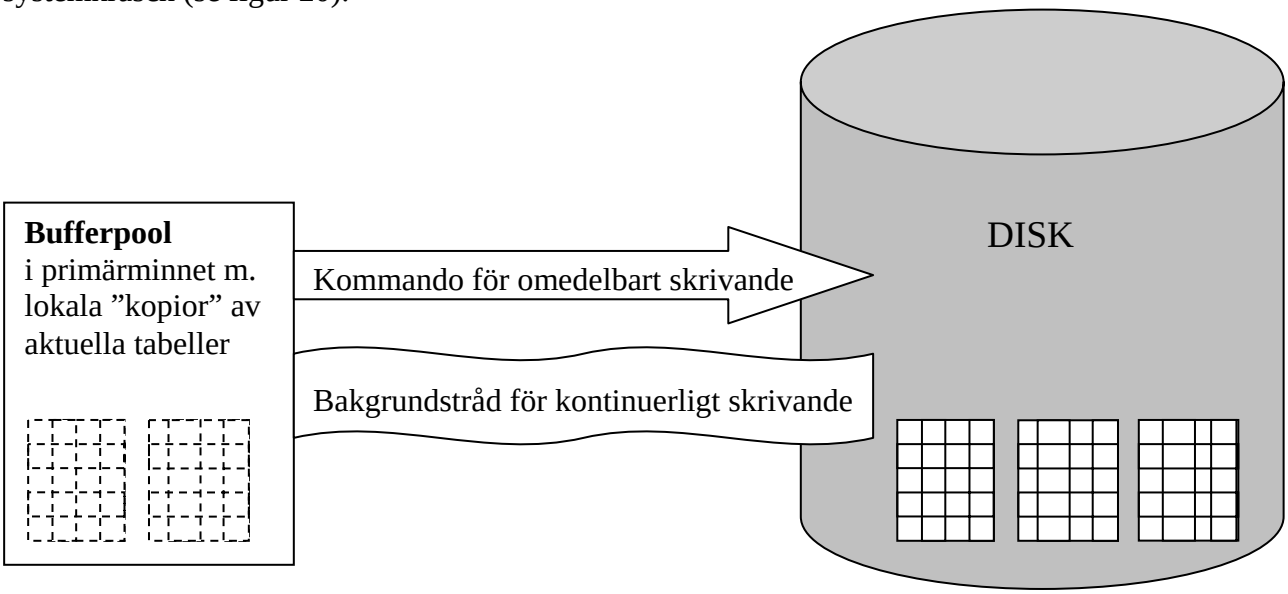
I utveckling av realtidssystem finns i Windows XP en *inbyggda problematik* som gäller prioriteringar för processer. Denna problematik gör att realtidsgarantierna för systemet kan *rubbas*. En kort beskrivning av problemet är att en högprioriterad process blockeras av en lågprioriterad som låser önskad resurs. Medelprioriterad process kan då få förtur och köra innan den lågprioriterade, (följd av den högprioriterade). I Windows CE är detta problem avhjälpt via immediate inheritance-protokollet [19] som används på den plattformen. Protokollet ser till att den lågprioriterade processen som låser resursen ärver den högprioriterades prioritet och därmed får köra före eventuell medelprioriterad process, vilket löser problemet.

7.2.3 Skrivning till disk

I nuläget fungerar realtidsdatabasen som en primärminnesdatabas där inga skrivningar av realtidsdata görs till disk. Detta beror på att databaspekarna pekar till, och därmed modifierar, enbart bufferpoolskopior (i primärminnet) av tabellerna. Om systemet kraschar försvinner all data.

CC Systems önskar att diskskrivningar sker kontinuerligt antingen i bakgrunden under pågående körning eller i form av ett kommando som man kan utföra regelbundet och kontrollerat. Detta skulle göra att vid en eventuell krasch förloras bara de senaste händelserna som ännu inte hunnit skrivas till disk.

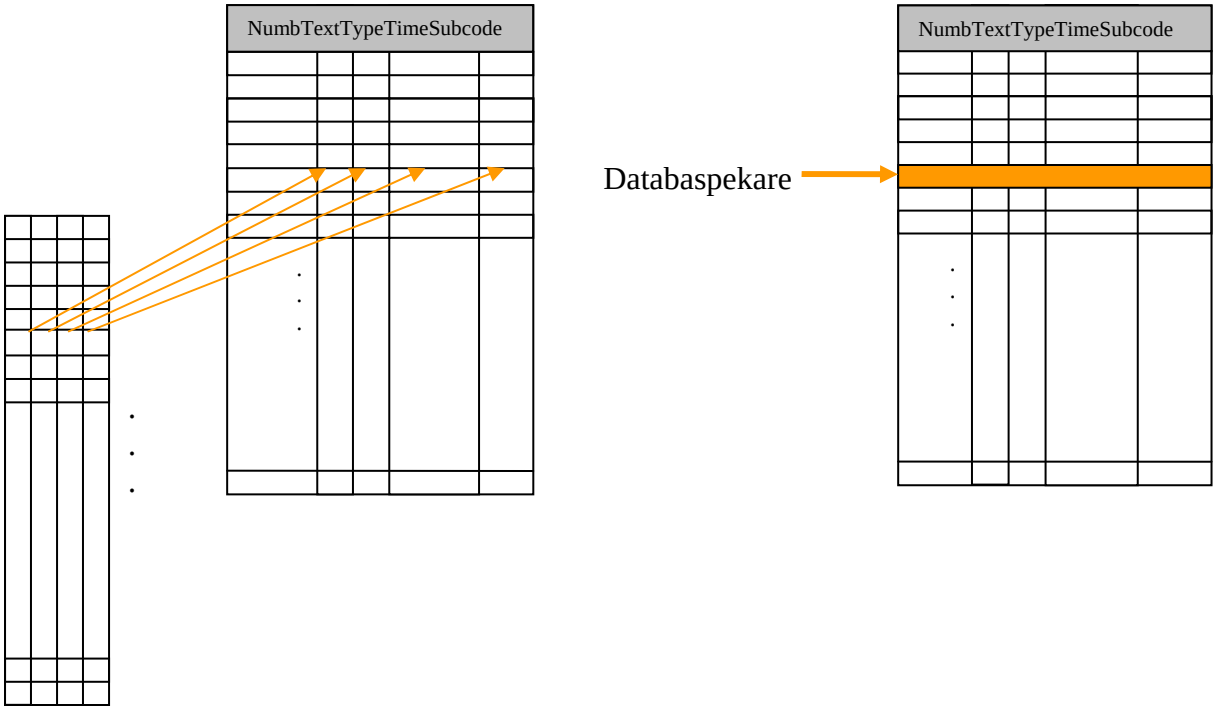
Hos Mimer pågår fortsatt utvecklande inom detta område och samtal med CC Systems pågår för att hitta en bra lösning för detta. Det finns många faktorer att ta hänsyn till kring detta. Exempelvis *vilka* data som skall sparas till disk. Man kan tänka sig att spara valda delar eller kategorier av data, vilket definieras då databaspekaren skapas eller på annat sätt via kommando under körning. En annan aspekt att ta hänsyn till är *när* skrivandet skall ske och det kan då röra sig om omedelbart, periodiskt eller ”on demand”. *Hur* data skall skrivas bör också betänkas. Synkront skrivande då systemet låses medan skrivandet sker, eller asynkront då det sker i bakgrunden. Oavsett vilken lösning det blir skall man ta hänsyn till hur mycket data som är godtagbart att förlora vid en eventuell systemkrasch (se figur 20).



Figur 20. Skrivande till disk, via kommando och/eller bakgrundstråd.

7.2.4 Uppdatering av hel rad

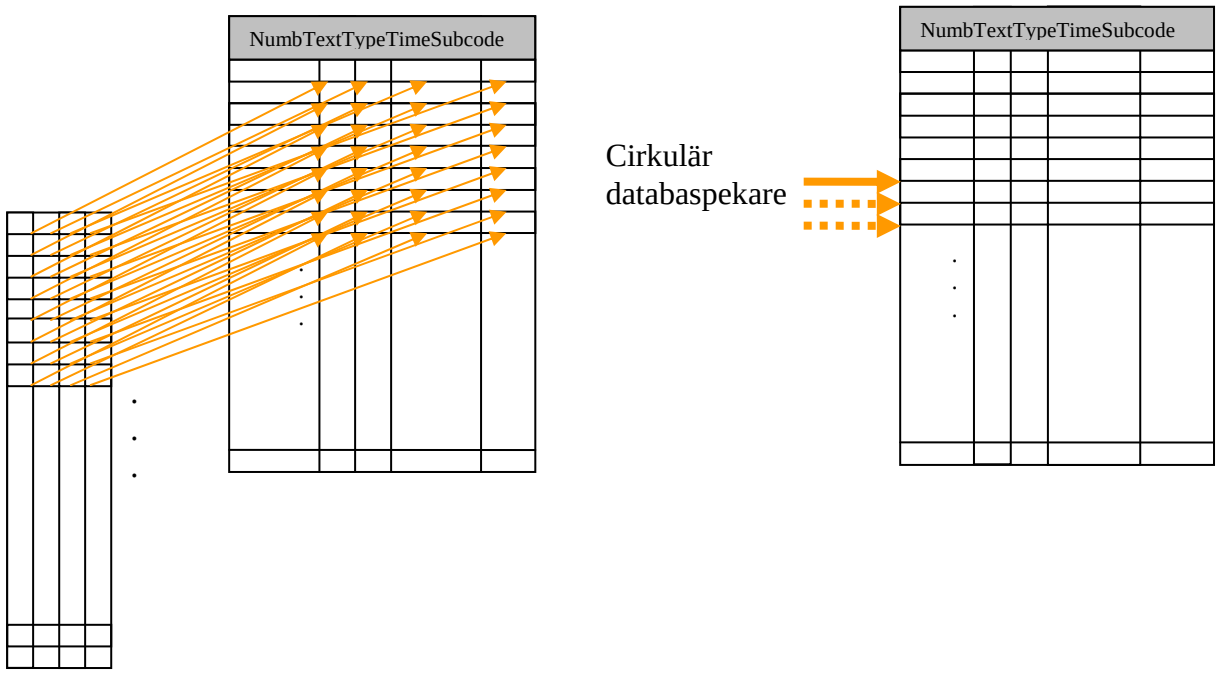
Atomär uppdatering av hel rad i en databastabell är ett nödvändigt tillägg för att undvika att bara delar av en rad blir uppdaterad och databasen således blir konsistent. I nuläget om systemet kraschar mitt i en skrivning kan det resultera i inkonsistens i databasen då enstaka celler i en rad är uppdaterade medan de övriga innehåller gammal information (se figur 21). Införandet av en rad i databasen via vanliga SQL-kommandon sker atomärt och så bör naturligtvis fallet med RT-pekarna också vara och anpassningen planeras komma i den release av Mimer RT som sker slutet av 2007.



Figur 21. Uppdatering av hel rad i databastabell.

7.2.5 Cirkulär databaspekare

En databaspekare som kan förflyttas till nästa rad i databastabell önskas. Detta är ett tillägg som bör införas för att det är vanligt att sådana system som är i behov av en realtidsdatabas också ofta använder delar av sin databas för olika typer av loggning av data. I de fallen vet man att nya data alltid kommer läggas in på *nästa rad* i aktuell tabell och tabellen bör då med fördel kunna göras cirkulär så att den transparent upplevs som en kontinuerlig *logg* som alltid innehåller de senaste lagrade värdena i tidsordning eftersom det senaste värdet skriver över det äldsta värdet (se figur 22). I vår DRE-applikation kommer vi ha stor användning av denna funktionalitet då vi redan nu använder oss av internt hanterade cirkulära tabeller för lagring av loggar. Denna anpassning kommer i den release av Mimer RT som sker i slutet av 2007



Figur 22. Skillnad i antalet databaspekare för en databastabell utan cirkulär databaspekare och med sådan.

7.2.6 Transaktionshantering

Ett exempel på en databastransaktion kan vara att via databaspekare läsa ett värde, öka värdet med ett och skriva det nya värdet. Detta förekommer relativt ofta i databassammanhang. Mimer RT har än så länge inget stöd för sådan typ av transaktionhantering utan operationerna sker som flera separata. Än så länge befinner det sig denna funktionalitet på planeringsstadiet men bör komma att ingå i produkten för att utöka dess användbarhet (för definition av *transaktion* se kap 4.1.1).

7.2.7 Databaspekare ”låser” tabeller.

I nuläget kan inte icke realtidsanpassade applikationer utföra modifieringar av tabell som har databaspekare kopplade till sig. Läsa går bra, men uppdatera är inte möjligt. Undersökningarna i denna rapport hanterar inte kopplingen mot GUIt så det berörs inte för närvarande men vid en implementation av MimerRT i hela systemet behöver detta beaktas. Exempelvis alla åtgärder som rensar hela eller delar av databasen skulle då behöva hanteras på annat sätt än förut. En möjlig lösning är att införa någon typ av serviceläge då man kopplar bort realtidsgränssnittet tillfälligt och låter GUIt utföra rensning av tabeller eller liknande modifieringar. Naturligtvis har Mimer som mål att undvika *låsningar* på detta sätt och vidare utvecklingsarbete angående detta är planerat.

7.2.8 Låsta storlekar på tabellerna.

Databaspekare kan inte användas till att utöka tabeller (insert) då de måste peka till redan existerande celler i tabeller.

7.3 Fortsatt undersökning av MimerRT

När MimerRT har utökats med ovanstående funktionalitet bör förnyade prestandatester göras, då dessa kommer att skilja sig markant från de i denna rapport angivna resultaten för tidsåtgång.

7.4 Utökningarnas förväntade ändring i prestanda för DRE

7.4.1 Tidsåtgång

Införandet av cirkulära pekare och atomär uppdatering kommer att ändra uppdateringshastigheten men det är oklart om det blir snabbare eller långsammare. Cirkulär pekare återanvänds och förflyttas vilket reducerar uppstartstiden betydligt jämfört med att tillverka pekare till varje cell i tabellen men fördröjer antagligen genomströmningstiden på grund av att arbete med att flytta pekaren skall göras för varje ny post som skall läggas till. Atomär uppdatering innebär skrivande till flera celler på samma rad istället för att skriva via flera pekare. Detta bör inte märkbart ändra tidsåtgången.

Ett byte till plattformen Windows CE påverkar inte skrivandet/läsandet via RT-pekare och påverkar således inte tidsmätningen i stor utsträckning.

Diskskrivandet är den viktigaste biten att tidsmäta och det bör vara denna som sedan kommer att vara avgörande ifall produkten kan/bör användas i DRE och/eller liknande tillämpningar i framtiden.

7.4.2 Minneskrav

Med tanke på de begränsade resurser rent minnesmässigt som fordonsdatorn har, jämfört med den PC som används i denna rapport, diskuteras här i ett troligt scenario för portningen av Mimer RT till Windows CE och därmed användning i fordonsdatorn:

Införandet av MimerRT ger något förändrat beteende för databasens resursutnyttjande, men i jämförelse med den tidigare använda databasen Mimer SQL Engine bör det inte bli alltför stora förändringar: Databasfilerna (databankerna) blir inte större, stackutrymmet bör vara oförändrat och ökningen i kodstorlek bör bli marginell. Det kommer krävas en del extra minne för de adminsitrativa aktiviteterna men det rör sig knappast om så stora förändringar att det skall skapa några problem, utrymmesmässigt. Ett tänkbart område, där man skulle kunna stöta på problem är det fysiska minnet i CE-datorn. Realtidslösningen låser ’bufferpoolen’ i det fysiska minnet, och om det är ont om fysiskt minne blir detta ett problem. Preliminära uträkningar och testkörningar visar att det bör gå bra att införa realtidsdatabasen i det existerande systemet DRE med bara smärre förändringar och ökning av storleken på bufferpoolen.

Diskussioner kring hur exempelvis för lite utrymme i bufferpoolen hanteras framgår inte i dokumentationen för produkten och mina preliminära tester kring detta visar att systemet kraschar utan att ge någon återkoppling till användaren. Vid ändring av värdet på bufferpoolens sidantal (2K, 16K och 64K stora sidor finns) ges felmeddelande vid för litet antal, med undantaget för om man väljer värdet 0 (noll) då inget meddelande visas alls.

7.5 Förmodad omarbetning av DRE vid övergång till MimerRT

Om man väljer att använda MimerRT i DRE så krävs en del ändringar i systemet. Här följer en sammanfattning av det arbete som behöver utföras.

- **MimerRTklass**
  För alla anrop via realtidsgränssnittet behöver returvärdeskontroller göras, och lämpligen bör en klass tillverkas för detta ändamål.

- **Ta bort deque-köerna**
  All köhantering (av t.ex. event, alarm och trender) kan tas bort. Flertalet ställen, bland annat Event- Alarm och Trendmgr i DRE, behöver justeras för att åstadkomma detta.

- **Commit i egen tråd eller ej**
  Beroende på vilken lösning som Mimer väljer för att utföra commit kan detta innebära olika mycket förändringar i systemet. Ett av förslagen var

att utföra commit i bakgrundstråd och då behöver vi skapa en sådan. I annat fall kan commit utföras med jämna intervall eller på annat sätt men måste hanteras av programmet.

- **Databasscriptet**
  En del operationer så som att stänga av loggning av databanker, och förhindra komprimering av tabeller skall införas.

  Nuvarande lösning med encellstabeller som håller reda på var i de stora tabellerna vi befinner oss är införda för att bespara en massa sökningsarbete för GUIt. En annan fördel med denna lösning är att då systemet försätts i viloläge kan vi vid återstart återgå till samma tillstånd som innan viloläget. Denna hantering kan komma att ändras något i realtidsfallet i och med att databaspekarna där sköter förflyttandet till nästa rad och därmed skapar loggarna på annat sätt än tidigare.

- **Tömma databasen/tabeller**
  Funktionen med att rensa hela eller delar av databasen behöver omarbetas. Förmodligen kan man lättast lösa detta med att inte tillåta rensning annat än i specifikt administratörsläge, t.ex. då maskinen inte körs och man då kan stänga av realtidsdelarna och låta rensningsfunktionerna få tillgång till de annars realtidslåsta tabellerna.

8 SLUTSATS

Användning av den undersökta databasen, Mimer SQL Real-Time Edition, ger en snabbare genomströmning av data i systemet. Köerna kan tas bort och på så sätt minska hanteringstiden för data samt göra systemet enklare. Produkten är inte färdigutvecklad ännu, men med den utökade funktionalitet som beskrivits ovan blir realtidsdatabasen en väl fungerande komponent för denna och liknande applikationer.

9 ORDLISTA

Förklaringar till begrepp som används i rapporten.

CAN	(Eng: Controller Area Netvork) Nätverksstandard för kommunikation mellan enheter I fordon.
CCP-XS	CC Systems Windows CE-baserad dator, en fordonsdator.
CC Systems	Företag, med utvecklingskontor ibland annat i Uppsala (för webbsida se referenser [9]) där examesarbetet utförs.
CoDeSys	Maskinstyrsystem. En mjukvaru-PLC.
Crosstalk	CC Systems produktserie med styrsystemlösningar.
DBMS	(Eng: Database Management Systems) Databashanterare.
DLL	(Eng: Dynamically Linked Library) Dynamiskt länkat bibliotek, består av en eller flera programfunktioner som kan delas mellan olika program för att tjäna ett visst syfte. (se referenser)
DRE	(Eng: Diagnostic Runtime Engine). Diagnostikmotor. Ett program för att diagnostisera t.ex. ett styrsystem.
ECA-regler	(Eng: On Event E, if Condition C, do Action A). Vid händelse E om egenskap C gäller så utför A.
GUI	(Eng: Graphical User Interface). Användargränssnitt. Finns även som Web-variant och är då kallad webGUI
Mimer	Mimer Information Technology. Ett företag som utvecklar databaser. Lokaliserat i Uppsala (för webbsida se referenser [3])
Mimer RT	Mimer SQL Real-Time Edition. En realtidsanpassad databas.
ODBC-gränssnitt	(Eng: Open Database Connectivity) Standardiserat gränssnitt för åtkomst av data i databashanterare.

Persistent	Data som är persistent överlever avstängning av det program som skapade det. I fallet med Mimerdatabaser är data som fullständigt skrivits till disk persistent.
PLC	(Eng: Programmable Logic Controller). Digital dator som används vid styrning av maskiner. T.ex. monteringsrobotar inom industrin.
RTDBMS	(Eng: Real Time Database Management Systems) Realtidsdatabashanterare.
WebGUI	se GUI
XML	(Eng: Extensible Markup Language). En fri öppen standard för ett generellt textspråk. Ett så kallat “markup language” som är text med tillhörande information om hur texten skall vara formaterad.

10 REFERENSER

Böcker

[1] Database Systems, 4th edition 2004, by Elmasri, Navathe.

[2] Database System Concepts, 4th edition 2002, by Silberschatz, Korth, Sudarshan.

Webbsidor och företagskontakter

[3] Mimers hemsida, 2007-09-18, kl 09.14, http://www.mimer.com/

[4] Data Management in Vehicle Control-Systems, Dag Nyström, 2007-09-14 kl 09.47, http://www.idt.mdh.se/~dnm/thesis/nystrom05PhDThesis.pdf

[5] Dokumentation av Mimer RT, Mimer SQL Real-Time Edition C API, 9.3.7D.

[6] Internt dokument som följde med betaversionen av Mimer SQL Real-Time Edition, Release Notes Mimer RT 9.pdf

[7] Samtal med Anders Eriksson, Mimer. Sep 2007

[8] Samtal med Dag Nyström, Mimer, Sep 2007.

[9] CC Systems hemsida, 2007-09-18, kl 09.15, http://www.cc-systems.com/

[10] CC Systems interna dokumentation, DREmanualen

[11] ODBC, 2007-10-22, kl 09.52, http://support.microsoft.com/kb/110093

[12] Databassystem, 2007-10-11, kl 13.15, http://www.ida.liu.se/~TDDB48/timetable/tddb48-fo1-introduktion.pdf

[13] Realtidssystem 2007-09-13, kl 11.12 http://sv.wikipedia.org/wiki/Realtidssystem

[14] Realtidsdatabas, 2007-10-10, kl 11.12, http://www.his.se/upload/18396/grunder%20i%20db.pdf

[15] Databas, 2007-10-10, kl 13.36, http://sv.wikipedia.org/wiki/Databas

[16] Transaktioner 2007-09-13 kl 11.40 http://www.databasteknik.se/webbkursen/transaktioner/index.html

[17] Webbsida om latching http://www.peacetech.com/flipper/oracle9i/901_doc/rac.901/a89867/coord.htm

[18] Imprecise computing, 2007-10-08, kl 11.05 http://ieeexplore.ieee.org/Xplore/login.jsp?url=/iel5/6460/17368/00801762.pdf

[19] Immediate inheritance protocol, 2007-09-19, kl 13.22 http://64.233.183.104/search?q=cache:rMmfinHGXbMJ:www.it.uu.se/edu/course/homepage/realtid/H04/ass4/fpscalc_assignment.ps+immediate+inheritance&hl=sv&ct=clnk&cd=1&gl=se&client=firefox-a

[20] Deque, 2007-10-23, kl 14.25, http://msdn2.microsoft.com/en-us/library/22a9t119(VS.80).aspx

[21] Relationsdatabas 2007-10-10, kl 12.14 http://sv.wikipedia.org/wiki/Relationsdatabas