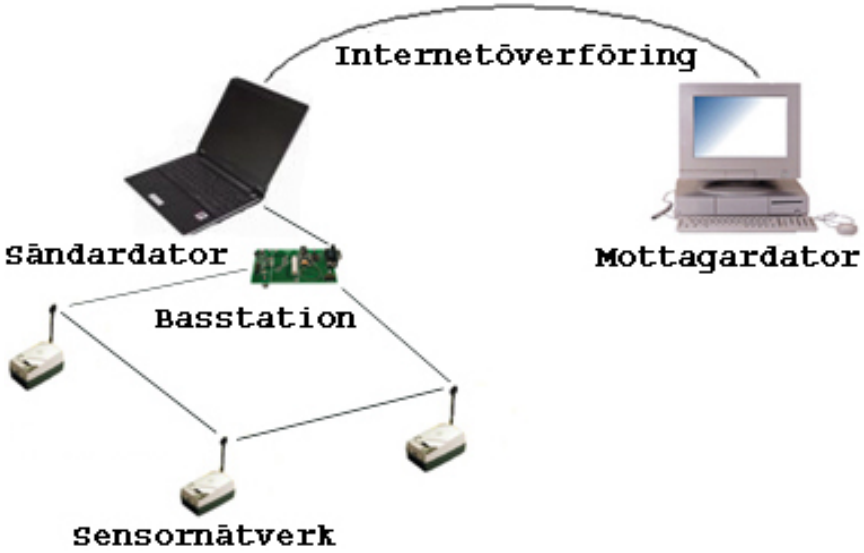
EXAMENSARBETE CERTEC, LTH NUMMER 6:2005

Markus Bylesjö

Trygghets-kommunikation med motes



Certec

Avdelningen för rehabiliteringsteknik
Lunds tekniska högskola

Certecs rapporter

Ett urval av rapporter från CERTEC

Enquist Henrik
Mina medicinska bilder
Certecrapport, 2004
http://www.certec.lth.se/dok/minamedicinskabilder/

Gustafsson Jörgen
Optik för synsvaga människor
Doktorsavhandling, 2004
http://www.certec.lth.se/dok/optikforsynsvaga/

Mandre Eve
Vårdmiljö eller lärandemiljö?
Doktorsavhandling, 2002
http://www.certec.lth.se/dok/franvardmiljotill/

Sjöström Calle
Icke-visuell haptisk interaktionsdesign
Doktorsavhandling, 2002
http://www.certec.lth.se/dok/ickevisuellhaptiskdesign/

Svensk Arne
Design av kognitiv assistans
Licentiatuppsats, 2001
http://www.certec.lth.se/dok/designavkognitiv/

Sammanfattning

Många situationer i vårt dagliga liv kan bidra till att skapa en känsla av otrygghet. Situationen är ofta sådan att möjlighet att dämpa oron finns genom olika former av kommunikation, men de känns ofta onödigt krångliga och används därför inte. Känslan av att inte vilja tränga sig på eller störa den andra personen är också ofta starkare än viljan att dämpa sin egen oro.

Arbetet baseras kring motes, en liten dator som drar minimalt med ström, kan övervaka ett flertal sensorer och kommunicera med omvärlden med hjälp av en radiosändare. Målet med examensarbetet var att använda en befintlig uppsättning av sådana motes, som tillsammans bildar ett trådlöst sensornätverk, för att ta fram en kommunikationslösning som syftar till att öka känslan av trygghet i vardagen.

När arbetet startades upp fanns motsen att tillgå, i det skick de levererades från återförsäljaren, varvid den första delen av projektet tillbringades med att genomföra grundläggande installationer på motsen för att systemet överhuvudtaget skulle gå att använda.

Efter detta inledande installationsarbete studerades ett flertal befintliga applikationer, för att se om någon av dem kunde användas som grund i den tänkta lösningen på problemet. Jag valde här att använda TinyDB, en befintlig applikation för insamling av sensorvärden, och genom att modifiera det togs ett lösningsförslag fram där kommunikation kan ske mellan motes och en internetansluten dator.

Lösningsförslaget testades sedan av två testpersoner, Ann och Håkan. Sensornätverket installerades i deras gemensamma hem och mottagare av informationen var Håkans arbetsdator. Efter en dryg veckas tester utvärderades systemet och jag fick mycket bra synpunkter på vilka förbättringar som kan göras, men också en bekräftelse på att systemet faktiskt bidragit till att ge en ökad trygghetskänsla hos Håkan, vilket var huvudmålet med arbetet.

Nyckelord

Trådlösa sensornätverk
Motes
Smart Dust
Trygghetskommunikation

Abstract

Many situations in our daily lives contribute to a feeling of insecurity. In many of these situations it is possible to reduce the anxiety through various forms of communication, but they often feel a bit too complicated and are therefore not used. The feeling of not wanting to disturb the other person is often stronger than the will to increase ones own anxiety.

The project focuses on motes, a small computer with minimal energy comsumption, that can monitor a number of sensors and comminicate with the outside world using a transmitter. This master thesis aims to, with an available set of motes that forms a wireless senor network, yield a way of communication that increases the sensation of security in everyday life.

The first task of the project was to install the motes fundamental software to get the basic applications of the system up and running.

After these opening installations a number of existing applications were studied, to see if any of them could be used as a base for the final solution. I chose TinyDB and by modifying it, a possible solution to the problem was presented with communication between motes and a computer connected to the internet.

Ann and Håkan then tested the first proposal of a solution. The sensor network was installed in their home, and the receiver of the information was Håkan's computer at work. After one week of extensive testing the system was evaluated, which rendered a lot of ideas of how to improve the system. Moreover, I got confirmation from Håkan that the system actually contributed to reduce his anxiety, which was the main goal of the project.

Keywords

Wireless Sensor Networks (WSN)
Motes
Smart Dust
Communication to reduce anxiety

Förord

Detta examensarbete, på 20 poäng, ingår i utbildningen Civilingenjör Datateknik 180p vid Lund Tekniska Högskola.

Anledningen till att jag valde att göra mitt examensarbete på Certec är att jag, efter att ha läst en kurs i rehabiliteringsteknik där, känt att institutionen har tankar och idéer kring samspelet mellan teknik och människa som tilltalar mig mycket.

Valet blev inte svårare när jag erbjöds att använda en mycket spännande teknologi där utvecklingen de närmaste åren troligen kommer att ge ett stort genomslag av nya produkter på marknaden. Det finns dock mycket kvar att göra, framför allt för att skapa mer användarvänliga system, men också i skapandet av applikationer till annat än ren övervakning. Det här examensarbetet är en bit på vägen mot en sådan applikation och kan förhoppningsvis väcka ett intresse som kan ligga till grund för vidare utveckling av tekniken.

Jag vill rikta ett stort tack till Ann och Håkan som testat mitt system, men också kommit med mycket tankar och funderingar kring applikationen som fick mig att vilja skapa en bättre slutprodukt. Jag vill även tacka mina handledare, Peter och Henrik, för idéer och inspiration när jag kört fast i något problem.

Lund juni 2005

Innehållsförteckning

1 Inledning

Jag har i det här examensarbetet undersökt möjligheten att använda trådlösa sensornätverk för att skapa en enkel och snabb form av trygghetskommunikation.

1.1 Problembeskrivning

Många situationer i vår vardag kan ge upphov till en känsla av otrygghet. Det kan vara saker som rädsla för att utsättas för brott eller annan kränkning, men också mer vardagliga situationer.

- Hur mår min gamla mamma som kanske inte orkar bo kvar hemma längre?
- Tog sig barnen hem som de skulle och dom har väl inte lämnat spisen på igen efter att ha kokat sin mjölkchoklad?
- Maken har väl inte ramlat ner från stegen när han skulle rensa hängrännorna? Jag skulle vilja ringa och kolla men då blir han ju bara arg och tycker jag kollar upp honom som om han vore ett litet barn.
- Gick bilresan hem bra?

Alla dessa mer eller mindre vanliga händelser kan bidra till att skapa en otrygghetskänsla i vår vardag. Situationerna är ofta sådana att möjlighet till kommunikation finns, men de känns onödigt krångliga och blir därför ett lite för stort steg att ta. Känslan av att inte vilja tränga sig på eller störa den andra personen blir starkare än viljan att dämpa sig egen oro.

1.2 Syfte

Arbetet syftade till att:

- Med hjälp av befintligt sensornätverk utveckla en produkt som kan användas för en enkel och tillförlitlig trygghetskommunikation.
- Genomföra tester och i samråd med testpersoner genomföra ändringar och förbättringar som i slutändan kan leda fram till en användvärd produkt.[Eftring,1999]

1.3 Förutsättningar

Arbetet baseras kring en teknik där enskilda trådlösa sensorer, *motes*, tillsammans bildar ett nätverk. Tekniken som motes grundas på ingår i det som även kallas *smart dust* och under examensarbetet fanns en uppsättning av motes att tillgå. Motes används i dagsläget nästan uteslutande till applikationer som övervakning av temperatur, tryck och till att tända och släcka ljus i

byggnader. Nya, mer innovativa applikationer utvecklas dock hela tiden. Föreställ dig system som helt automatiskt övervakar inomhusmiljön i ditt hem, kontrollerar hållfastheten i en bro, talar om när din bil behöver lagas och dessutom bokar tid på verkstaden, övervakar en hjärtsjuk persons värden i hemmet och automatiskt skickar dessa till läkaren på sjukhuset. Många av systemen befinner sig ännu i ett utvecklingsskede, men färdiga applikationer utvecklas ständigt och utvecklingspotentialen är oändlig.

Arbetets centrala del var att undersöka och testa möjligheterna med trådlösa nätverk, vilket innebar att möjliga lösningar med annan teknologi inte undersöktes.

2 Metod

Då arbetet påbörjades fanns en uppsättning motes, inköpta från Crossbow, på institutionen där examensarbetet skulle genomföras. Vid uppstarten var motsen enbart testade för att se att de fungerade men aldrig programmerade eller på annat sätt modifierade från fabriksinställningarna. Med det befintliga systemet genomfördes arbetet i stora drag på följande sätt:

- Inhämtning av information kring tekniken och de specifika motsen. Detta rörde allt som hade med motsens funktionalitet att göra, från hur de skulle programmeras till hur dockning mellan mote-basstation-dator fungerade.
- Grundinstallation av mjukvara. Detta rörde den mjukvara som behövdes för att koppla upp och testa systemet, programmera motsen och köra enkla testapplikationer (se kapitel 4).
- Egna tester och genomgång av befintliga applikationer. Här ändrades de befintliga applikationerna, bland annat för att möjliggöra programmering via radiolänk.
- Framtagning av lösningsförslag på system för trygghetskommunikation genom modifiering av vald mjukvara (se kapitel 5).
- Anpassning till testsituationen. Motsen fick ett skyddande hölje och på en bärbar dator installerades all behövlig mjukvara. Även överföringen av information via internet löstes här (se kapitel 5.2).
- Användartester. Systemet kopplades upp i användarnas hem och tester genomfördes under en dryg veckas tid. Modifiering av systemet genomfördes ett flertal gånger under testperioden, både av testpersonerna själva, men även av mig.
- Utvärdering av tester. Intervjuer genomfördes med testpersonerna.

3 Teknik

3.1 Motes

En mote är en liten och billig dator som drar minimalt med ström. Datorn övervakar en eller ett flertal sensorer, exempelvis sensorer för temperatur, ljus, ljud, position, acceleration och fuktighet. Med hjälp av en radiosändare kan datorn kommunicera med omvärlden. All denna teknologi trycks ihop på så liten yta som möjligt.

Ad hoc-nät är ett datornät som är decentraliserat och självorganiserat av autonoma noder.

Möjligheten att bygga upp ad hoc-nätverk med dessa mikroelektromekaniska sensorer (MEMS) är den enskilt största fördelen med multisensorerna. Ett ad hoc-nätverk består av hundratals eller tusentals noder som själva inordnar sig i nätverket och kommunicerar och skickar vidare information mellan varandra. Detta ger ett extremt kraftfullt nätverk som kan spridas över en stor yta och med mycket liten energiåtgång överföra lokalspecifik information till en övervakningsenhet. [Intel, Ad Hoc Networks]

3.1.1 Specifikation av projektets motes

De motes som användes under projektet ingick i ett paket inköpt från Crossbow (se bild 3.1).

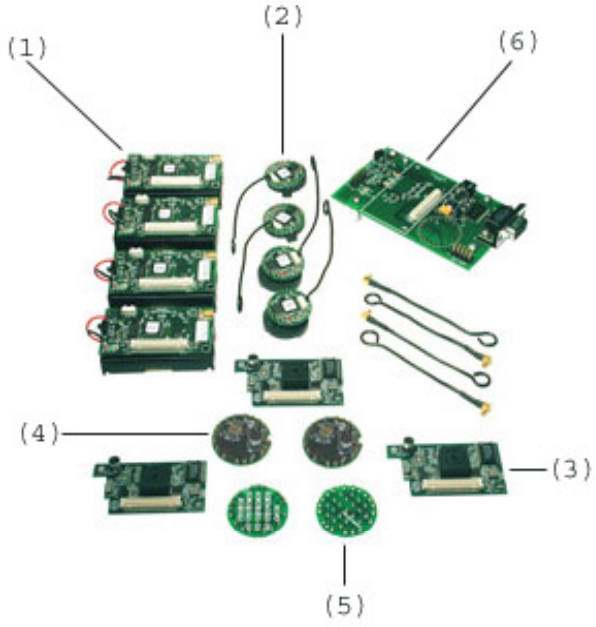


Bild 3.1 Mote-kit 5040 från Crossbow [1]

I paketet ingick:

1. 4 stycken MPR400 mica2 processor/radiokort. MPR400 har storleken $58 \times 32 \times 7$ mm, exklusive batterihållare.
2. 4 stycken MPR500 mica2dot processor/radiokort. MPR500 har måtten 25×6 mm, exklusive batterihållare.
3. 3 stycken sensorkort, MTS310, till mica2. Kortet innehåller sensorer för acceleration, magnetism, temperatur, ljus och ljud. Dessutom finns en buzzer som avger en signal med frekvensen 1KHz.
4. 2 stycken sensorkort, MTS510, till mica2dot innehållande sensorer för acceleration, ljus och ljud.
5. 2 stycken MDA500 kort. Kortet används som en dockningsplatta mellan mica2dot och basstationen MIB510.
6. Ett MIB510 kort vilket används för programmering av noderna och för att sammankoppla nätverket, via parallellport, till en dator. [Crossbow Mote-kit 5040]

De mica2 som användes arbetar inom frekvensbandet 868/915 MHz och är kompatibla med MPR500, vilket gör att mica2 och mica2dot kan kommunicera med varandra. Möjlighet att variera arbetsfrekvensen inom detta område finns och radion har en räckvidd på 30-50 meter beroende på yttre faktorer. Mica motes använder en Atmel ATmega 128L processor som arbetar vid 4MHz. Processorn i Mica motes är ungefär lika kraftfull som den som fanns i IBM: s original PC från 1982, men kräver bara 8 mA vid drift och 15μA vid vila. För att lagra program har processorn ett 128kB flashminne och för lagring av data finns även ett 512kB flashminne. En 10-bitars A/D-omvandlare används för att digitalisera informationen från sensorerna.[How Stuff Works]

3.1.2 Exempel på alternativa motes

Det finns ett flertal olika sensorkort och kort för programmering av motes på marknaden förutom de som användes under projektet. Exempel på sådana är:

- MicaZ. Är Crossbows första mote som använder 2.4GHz bandet och är kompatibelt med Crossbows övriga sensorkort och programmeringskort. Kan användas till ljud, video och andra applikationer som kräver hög hastighet i överföringen av data.
- TelosB Mote plattform har inbyggda sensorer för temperatur och fuktighet och även antennen finns integrerad på kortet. Programmering av kortet och insamling av data sker via en USB port.[Crossbow, New Products]

Bild 3.2 Crossbows MicaZ [2]

Bild 3.3 TelosB från Crossbow [2]

Bild 3.4 Tmote Sky från Moteiv [3]

Bild 3.5 MTS420CA sensorkort [2]

Bild 3.6 Pluto Mote [4]

- Tmote Sky. Är den senaste generationens mote plattform. Tmote Sky har integrerad antenn som ger en räckvidd på upp till 125 meter och även integrerade sensorer för fuktighet, temperatur och ljus. Tmote Sky kräver dessutom extremt lite ström och har en uppvakningstid från vila på under 6 μs .[Moteiv]
- MTS420CA är sensorkort, kompatibla med Mica2 och MicaZ med sensorer för temperatur, fuktighet, tryck, ljus, acceleration och avkänning av GPS signaler.
- Pluto Mote. Moten är designad vid Harvard University och är i grunden en Telos mote som har gjorts mindre för att kunna bäras på kroppen. Pluto moten innehåller bland annat ett litet uppladdningsbart Li-jon batteri, en USB kontakt och en accelerometer. Pluto används för att mäta fysisk aktivitet och motoriska funktioner. [CodeBlue]

3.2 Mjukvara

3.2.1 TinyOS

TinyOS är ett händelsedrivet operativsystem designat för noder i ett sensornätverk vilka har mycket begränsade resurser. Noderna måste organisera sig själva i ett nätverk genom att lyssna på andra noders signaler och lista ut vem det är de hör. Svårigheten med ett sådant system är att det kostar energi att bara lyssna och energitillgången är mycket begränsad. För att möjliggöra ett system där motes oftast befinner sig i viloläge men ändå låta information passera från nod till nod i nätverket skapades ett kompakt operativsystem, TinyOS. För att minimera energiåtgången och säkert sända information kodar TinyOS datapaketen, mycket likt den typ av datahanteringsprotokoll som används till internet, och startar bara radiodelen vid behov.

TinyOS har en öppen källkod skriven i nesC, ett programmeringsspråk som liknar C, vilket gör att användare själva kan modifiera systemet efter specifika önskemål. [Getting Started Guide]

3.2.2 NesC

NesC är en utveckling av programmeringsspråket C och är designat för att klara av strukturen och exekveringen av TinyOS. De grundläggande koncepten bakom nesC är:

- Program byggs av komponenter som sedan sammanfogas för att bilda hela program.
- Komponenterna är statiskt sammanlänkade gränssnitt.

- Interfacet möjliggör en komplex interaktion mellan komponenter, till exempel anmäla intresse för en viss händelse och skicka tillbaka information om den inträffar. [NesC]

3.2.3 Cygwin

Cygwin är en Linux-liknande miljö till Windows.
När Cygwin finns installerat är det möjligt att använda många Unixprogram i Windowsmiljö, utan att göra större ändringar i källkoden. Ett exempel på ett sådant program är de huvudsakliga delarna av GNU mjukvaran. Till denna mjukvaran följer även en del hjälpverktyg som kan användas både från kommandoprompten och från Cygwinfönstret. [Cygwin]

3.2.4 SerialForwarder

SerialForwarder är ett program skrivet i Java och används för att läsa information från datorns serieport och vidarebefordra den så att andra program kan kommunicera med sensornätverket. Väl uppkopplat lyssnar SerialForwarder efter tillkopplade nätverk på specificerad TCP port och sänder vidare TinyOS meddelanden från serieporten.

3.2.5 TinyDB – applikation till motes

TinyDB är ett system för att underlätta insamling av sensorinformation från motes. Programmet följer med i TinyOS nyare programpaket. Den stora fördelen med TinyDB är att ett mycket enkelt grafiskt användargränssnitt tillhandahålls, där användaren själv kan specificera vilken information som skall hämtas från motsen. Informationen kan sedan bearbetas på olika sätt, exempelvis medelvärdesbildas eller filtreras bort. All sensorinformation kan också sparas i en databas för vidare bearbetning.

Den huvudsakliga uppgiften för TinyDB är att underlätta för utveckling av applikationer. Genom att använda TinyDB slipper man skriva applikationer direkt till sensorerna, en uppgift som kan vara mycket komplicerad. TinyDB innefattar bland annat:

- High Level Queries: Användargränssnittet låter användaren själv bestämma vilken typ av data som skall hämtas utan att exakt deklarera hur detta skall göras.
- Network Topology: TinyDB sköter själv hanteringen av nätverket genom att spåra sina grannar och se till att sensorvärden från noderna kan levereras på ett säkert sätt.
- Multiple Queries: TinyDB tillåter att flera förfrågningar körs samtidigt på en uppsättning motes.

TinyDB ingår även i ett större programpaket, TASK (Tiny Application Sensor Kit), med ytterliggare hjälpmedel för insamling av sensorvärden (se kapitel 3.2.6). [TinyDB]

Efter installation av TinyDB, både på dator och motes, kan programmet initieras. Detta resulterar i att två olika fönster öppnas (se bild 3.7).

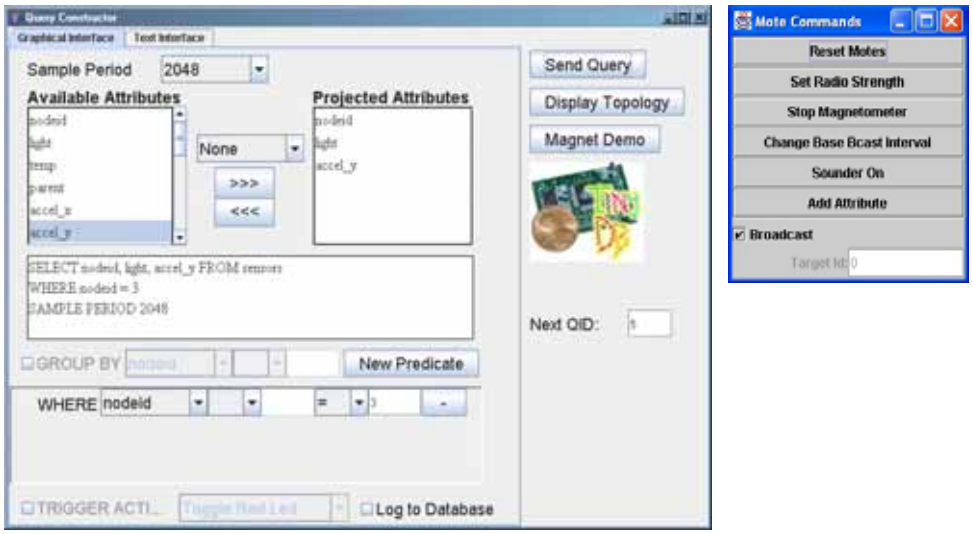


Bild 3.7 Grafiskt användargränssnitt till TinyDB

I Query Constructorfönstret skapas de förfrågningar som sedan skickas till noderna i nätverket. De viktigaste funktionerna är:

- Available Attributes: Här väljs vilka sensorvärden som skall hämtas från noderna. När de valts flyttas de till Projected Attributes.
- Sample Period: Bestämmer hur ofta valda noder skall skicka sensorvärden. Anges i millisekunder.
- New Predicate: Ger möjligheten att studera sensorvärden från enskilda, eller grupper av noder.
- Trigger Action: Om ett valt värde över/understigs kommer noden i fråga att utföra vald handling. Exempel på sådan handling är att en röd lysdiod börjar blinka eller att noden avger en pipsignal.
- Send Query: Sänder förfrågan till noderna och startar insamling av mättvärden.

Fönstret Mote Commands används för att direkt adressera noder i nätverket. Möjligheten att, exempelvis, nollställa noderna i nätet eller få en enskild nod att pipa finns här.

När förfrågan skickas till noderna öppnas ytterliggare ett fönster som visar de insamlade värdena i en tabell. Vid många former av förfrågningar ritas även de insamlade värdena i en graf för att bättre åskådliggöra resultaten (se bild 3.8).

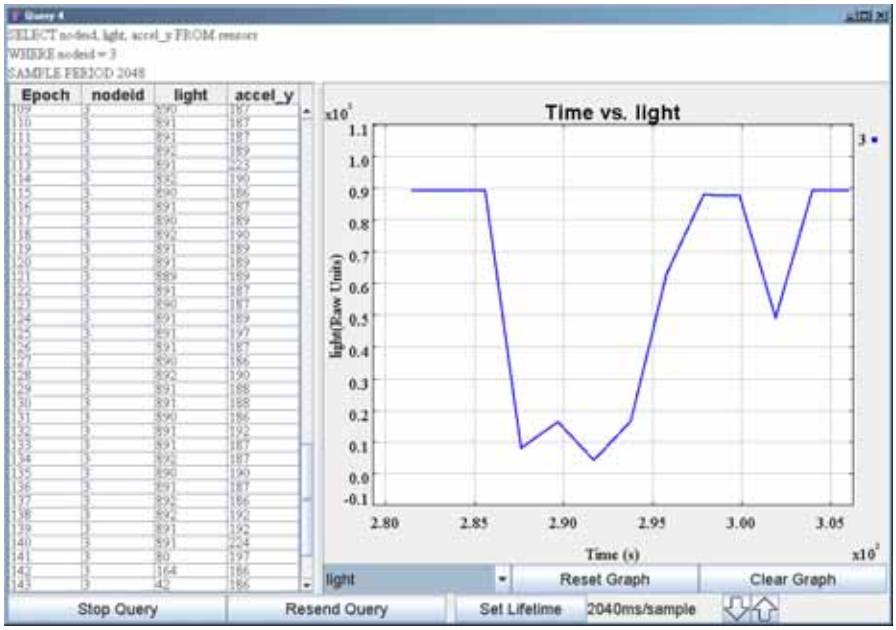


Bild 3.8 Förfrågningens resultatfönster

Möjlighet att spara undan förfrågningens sensorvärden i en databas finns också. Detta görs genom att kryssa i rutan ”Log to Database” i Query Constructorfönstret (se bild 3.7). När förfrågan är slutförd kan dessa värden på ett enkelt sätt exporteras till exempelvis ett excelark för att sparas undan i en mer lätthanterlig form än vad databasen erbjuder.

3.2.6 TASK

TASK (Tiny Application Sensor Kit) är en samling verktyg som används för att minska svårigheterna för icke experter att skapa egna applikationer till sensornätverk, framtaget av Intel Research, Berkeley. TASK innehåller följande komponenter:

- TinyDB. Låter användaren kommunicera med sensornätverket genom ett SQL-liknande användargränssnitt.
- TASK server. Fungerar som en proxy för sensornätverk via internet.
- TASK DBMS. Databas som sparar sensorvärden. Nuvarande versioner av TASK stöder bara PostgreSQL.
- TASK client tools. Samling av verktyg som bland annat ger stöd för användaren att samla in data och filtrera denna.
- TASK Field Tool möjliggör problemlösning inom vissa delar av nätverket på distans.

TASK är dessutom mycket enkelt att integrera med hjälpmedel för analys av data. Exempelvis Excel, Matlab och ArcGis. TASK används ofta med applikationer där sensorvärden skall samlas in

under en lång tid eftersom systemet är mycket stabilt och fel ofta kan korrigeras via nätverket.

3.2.7 NetMeeting

Microsoft NetMeeting är ett program för att kommunicera online med andra personer över internet. Med NetMeeting går det kommunicera med andra användare på ett flertal olika sätt, bland annat genom att skriva, använda mikrofon och rita. Dessutom går det att dela med sig av applikationer och skicka filer till andra användare.

NetMeeting har de senaste åren ersatts av Microsoft Messenger. NetMeeting finns dock inbyggt i Windows 2000 och XP. För att kunna använda programmet måste oftast en installation göras. Denna görs enklast genom att klicka på Start > Kör och sedan skriva conf. Följ sedan installationsinstruktionerna.

3.2.8 MSN Messenger

MSN Messenger har ersatt NetMeeting på senare tid. Programmet gör det möjligt att chatta online i realtid med text eller videokonversationer. Med MSN Messenger finns också möjligheten att via internet dela ut kontrollen över egna program och applikationer till andra användare.

4 Grundinstallationer

Det praktiska arbetet inleddes med installation av mjukvara för att få det helt oanvända systemet att fungera. Detta innefattade installation av programmen som används för att kontrollera motsen, men även ändringar i den befintliga källkoden för att modifiera motesapplikationer så att de fungerade på ett mer tillfredställande sätt. Bland annat installerades DelugeC på motsen som gör det möjligt att programmera motes via radiolänk. Även frekvensen som noderna använder för att kommunicera med varandra ändrades. De färdiga applikationssystemen, TinyDB och TASK installerades och testades grundligt.

Grundinstallationerna kom att utgöra en betydande del av det totala arbetet och dokumenteras mer specifikt i bilagorna A-E.

4.1 Inledande installationer och programmering av motes

Den mest betydelsefulla programvaran som måste installeras för att systemet överhuvudtaget skall fungera är TinyOS. Installationerna inleddes således med att TinyOS installerades tillsammans med Cygwin (se bilaga A).

När detta var gjort kunde motsen programmeras för första gången. Basstationen (MIB510) sammanlänkades med datorns com-port och önskad mote kopplades till basstationen.

Efter de inledande installationerna genomfördes tester med enklare applikationer för att kontrollera att allt fungerade som det skulle, det vill säga att noderna klarade av att koppla upp sig i ett nätverk och kommunicera med basstationen.

4.2 Ändring av frekvens

I sitt grundutförande använder motsen 915MHz som standardfrekvens. Då detta är en olaglig frekvens att använda för privatpersoner i Sverige, behövde frekvensen ändras (se bilaga B). Alla frekvenser mellan 868MHz och 915MHz skall gå att använda med den uppsättning motes som användes. Det bör dock tas i beaktning att om flera sändningsband används samtidigt skall dessa separeras med minst 500Hz för att undvika störningar.

Frekvensen sattes inledningsvis till 868,202MHz, men det visade sig att den frekvensen inte fungerade över huvud taget. Trots ett flertal kontakter med Crossbows support, så hittades aldrig någon orsak till varför den frekvensen inte fungerade. Enligt Crossbows support så ska den frekvensen fungera alldeles utmärkt.

Möjligen kan orsaken vara att störningarna är för stora vid den frekvensen för att systemet skall fungera. Efter ett antal försök hittades dock en fungerande frekvens, 869,614MHz, som i fortsättningen användes som standardfrekvens för mica2 såväl som mica2dot.

4.3 Programmering via radiolänk

Det inledande arbetet med otaliga installationer och omprogrammeringar av motsen gav upphov till tankar om att det borde finnas ett bättre sätt att programmera om noderna i nätet än att koppla dem en åt gången till basstationen. Efter vidare undersökning i ämnet visade det sig att med version 1.1.8 och senare av TinyOS finns möjlighet att via motens radiolänk simultant programmera om alla noder i nätverket. Denna möjlighet tillhandahålls med programtillägget DelugeC.

Då den senast installerade versionen av TinyOS var 1.1.7 gjordes en CVS-snapshot uppdatering av TinyOS. Även nesC uppdaterades till version 1.1.1.2 (se bilaga C).

När alla nödvändiga ändringar och installationer gjorts fungerade radioprogrammeringen till synes alldeles utmärkt. Resultatet blev ett nätverk där endast nod 0 behövde programmeras om och nollställas. Vid nollställningen kommer alla noder i nätverket att programmeras om med samma innehåll som nod 0.

Här uppstod dock ett problem. När mica2 respektive mica2dot omprogrammeras skapas först en plattformsberoende fil, som nästan är identisk för mica2 och mica2dot, innan själva installationen genomförs. Just det faktum att de kompilerade filerna nästan är identiska för mica2 och mica2dot är källan till problemen. Vid radioprogrammeringen innebär detta att systemen är tillräckligt lika för att kommunicera och därigenom försöker programmera om varandra. Resultatet av detta blir att om en mica2 upptäcker en mica2dot i nätverket, ser den att programmet som körs på mica2dot inte är identiskt med det som körs på mica2 och försöker programmera om mica2dot noden. Omprogrammeringen kommer att lyckas, då de är tillräckligt lika för att kommunicera, men filen som skickas över är skapad för mica2 plattformen vilket resulterar i att mica2dot helt slutar fungera.

Utvecklarna av DelugeC, samma som gruppen bakom TinyOS, jobbar på att lösa problemet men i nuläget fungerar programmering via radiolänk enbart på nätverk bestående av noder med samma plattform.

4.4 TinyDB

TinyDB är ett system (se kapitel 3.2.5), skapat av TinyOS, för att underlätta insamling av sensorinformation från motes. Programmet följer med i TinyOS programpaket nyare än 1.1.0. För att testa applikationen och få en djupare förståelse för dess möjligheter och tillkortakommanden installerades systemet och testkördes på sensornätverket (se bilaga D).

4.5 TASK

TASK (se kapitel 3.2.6) är en samling verktyg för att underlätta för icke experter att ta fram egna applikationer till sensornätverk och underhålla dessa på ett enkelt sätt. För att testa systemet och få en djupare förståelse för dess möjligheter och tillkortakommanden installerades systemet och testkördes på sensornätverket (se bilaga E).

5 Lösningsförslag

Jag föreslår här en form av trygghetskommunikation där en person använder motes för att sända meddelanden och en andra person kan ta emot dessa meddelanden på en dator. Användaren av mottagardatorn skall också ha möjlighet att från sin dator sända meddelanden till motsen, exempelvis i form av pip eller blinkande lysdioder.

Den tänkta lösningen (se bild 5.1) kräver att informationen från motens sensorer kan visas för distansanvändaren på ett lämpligt sätt. Dessutom måste en tvåvägskommunikation möjliggöras så att distansanvändaren kan sända information till motsen.

Försöken med att skapa ett sådant system inleddes med att ett flertal befintliga system för insamling av sensordata undersöktes, exempelvis SurgeView, TinyDB och TASK för att se om någon av dessa skulle kunna fungera som en grund till det tänkta systemet. TinyDB och TASK utkristalliserade sig snabbt som de bästa kandidaterna och undersöktes därför mycket djupgående. Valet föll slutligen på TinyDB som visade sig bättre anpassat till den tänkta lösningen än TASK och användes därför som grund till lösningens mjukvara.

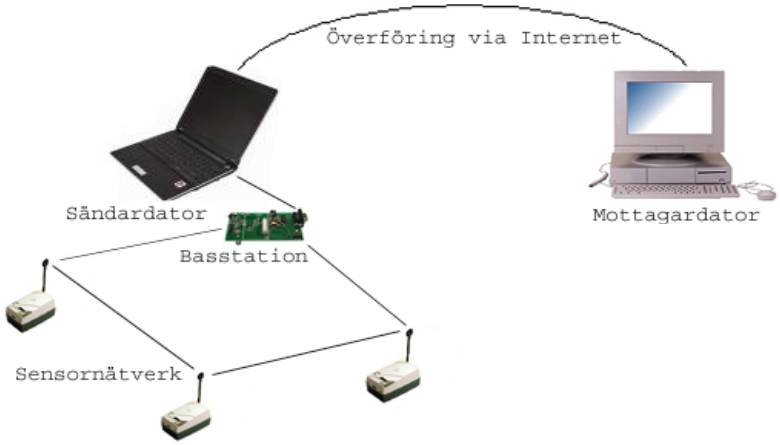


Bild 5.1 Schematisk bild av den tänkta lösningen

5.1 Modifierat TinyDB

TinyDB är i sitt originalutförande ett system för att samla in sensordata och åskådliggöra detta för användaren på ett tydligt sätt (se kapitel 3.2.5). Dessutom finns möjlighet att via dator, med knapptryckningar, kommunicera med motsen i nätverket, antingen alla på en gång eller en åt gången. Trots detta krävs en hel del förändringar av källkoden för att systemet skall gå använda till den tänkta applikationen.

De klasser som styr TinyDB och framförallt de som styr det grafiska användargränssnittet ligger i mappen C:\Program Files\tinyos\cygwin\opt\tinyos-1.x\tools\java\net\tinyos\tinydb.

De klasser som modifierades var (se bilaga F):

- CmdFrame, styr utseende och funktion för Mote Kommunikationsfönstret. Ändringarna här gick i huvudsak ut på att forma ett lättförståeligt fönster med enbart de mest nödvändiga knapparna för att kommunicera med motsen. En textruta, med varierande bakgrundsfärg, där informationen från motsen skrevs ut skapades också.
- CommandMsgs, skapar de meddelanden som sedan skickas till noderna i nätet. Ändringar gjordes så att motsen kunde pipa eller tända lysdioder vid knapptryckning.
- ResultFrame, skapar det fönster som visar efterfrågade sensorvärden. Ändringar gjordes här så att insamlade sensorvärden sparades undan i en vektor. Vektorvärdena användes sedan för att avgöra om önskad händelse inträffat eller inte.

Ändringarna resulterade i ett användargränssnitt med ett något förändrat utseende (se bild 5.2) i jämförelse med originalet (se bild 3.7).

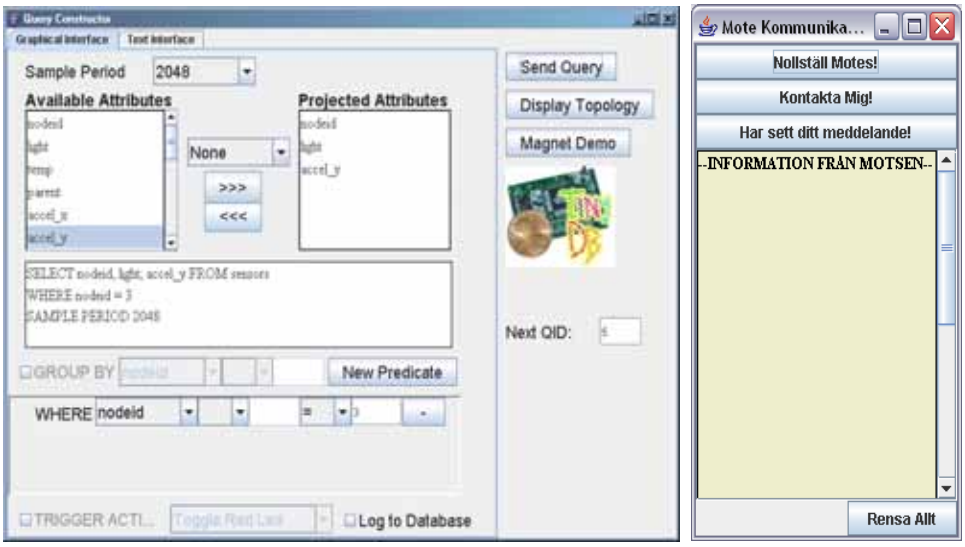


Bild 5.2 Visar Query Constructor fönstret och Mote Kommunikationsfönstret i den ändrade versionen

Query Constructorfönstrets funktion ändrades inte alls utan förändringarna är gjorda i Mote Kommunikationsfönstret.

5.2 Funktionalitet hos tänkt användarsystem

Ändringarna i TinyDB innebär att ett system där sensorinformationen kan användas som signalindikatorer har realiserats. Förutom detta finns även möjligheten att med en enkel knapptryckning sända meddelanden till motsen, det vill säga att en tänkt datoranvändare kan kommunicera med sensornätverket. Förutsättningar fanns således för att skapa det önskade systemet för trygghetskommunikation och det kom i huvudsak att bestå av två delar.

- TinyDB med insamling av sensorvärden. Förfrågning skapas i TinyDB och om önskad händelse inträffar sänds informationen till Mote Kommunikationsfönstret.
- Överföring av informationen till annan dator. På denna dator skall det förutom att passivt övervaka sensorvärdena även gå att sända meddelanden till motsen, det vill säga kunna använda sig av Mote Kommunikationsfönstrets knappar.

5.2.1 Så fungerar användarsystemet

Bild 5.3 Skakning av mote

Bild 5.4 Ändring av ljusvärde

Kärnan i hela användarsystemet är den förfrågan som skapas i TinyDB och skickas ut till noderna i nätet (se bilaga G). TinyDB kommer sedan att samla in sensorinformationen och visa den i ett resultatfönster. I det realiserade systemet frågades efter tre saker: nodeid, light och accel_y. Om något av värdena light eller accel_y under- eller översteg ett visst värde sändes denna information till Mote Kommunikationsfönstret. Gränsvärdena för light och accel_y ändras i metoderna varnaljus och varnaaccel i klassen CmdFrame.java.

När förfrågan är skickad kommer händelsen att ljuset, eller accelerationen i y-led, passerar visst gränsvärde innebära att bakgrundfärgen hos Mote Kommunikationsfönstret ändras och ett valt meddelande skrivs ut tillsammans med tid och datum för händelsen (se bild 5.5).

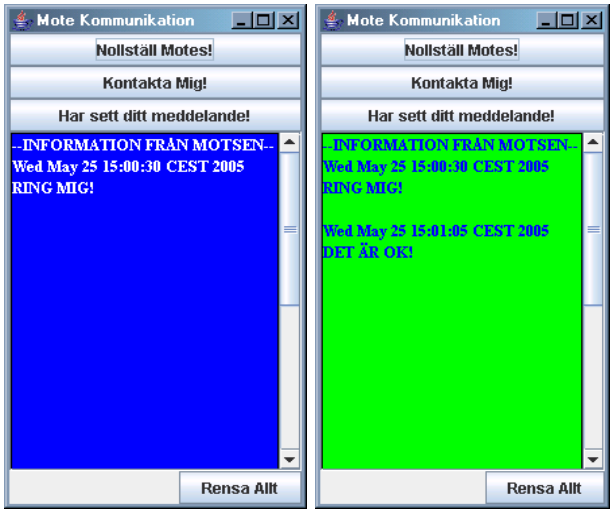


Bild 5.5 Visar informationen i Mote Kommunikationsfönstret då signalhändelse inträffat

I det tänkta användarfallet motsvarar en blå bakgrund och meddelandet ”RING MIG!” att ljuset gått under ett visst värde. Grön bakgrund och meddelandet ”DET ÄR OK!” motsvarar en skakning vilken ger utslag större eller mindre än visst värde.

Antalet meddelandenivåer var begränsade till två stycken och det styrde valet av meddelanden. Valet föll på två meddelanden som tydligt var separerade från varandra för att täcka in ytterligheterna i användarens tillstånd. Vilka meddelanden som var lämpliga att använda diskuterades även med Håkan, som senare testade systemet (se kapitel 6.2).

Bakgrundfärgerna till textfönstret valdes även de så att skillnaden mellan olika meddelanden skulle vara så tydlig som möjligt. Den gröna färgen kändes mycket naturlig att koppla samman med signalen ”ALLT ÄR OK!”, men bakgrundsfärgen till meddelandet ”RING MIG!” var svårare att välja. Först testades röd bakgrundsfärg, men den kändes lite för alarmerande då den ofta förknippas med fara och olycka. Därför valdes en blå färg, som tydligt skiljer sig från den gröna, men inte väcker samma oroskänslor som den röda färgen.

Mote Kommunikationsfönstret innehåller även 4 stycken knappar.

- Nollställ Motes! Om fel uppstår och motsen slutar samla in sensorinformation kan denna knapp användas för att ta bort nuvarande förfrågan. Efter att detta kommando givits måste förfrågan skickas igen.
- Kontakta mig! Kommer resultera i att alla motes i nätverket piper.
- Har sett ditt meddelande! Kommer resultera i att gul lysdiod på motsen tänds.

- Rensa Allt. Rensar textfönstret och återställer bakgrundsfärgen till den ursprungliga (se bild 5.2).

5.2.2 Överföring av information via internet

Användarsystemet bygger på att användaren av mottagardatorn kan se och använda sig av Mote Kommunikationsfönstret. För att möjliggöra detta användes programmen NetMeeting och MSN Messenger (se kapitel 3.2.7-3.2.8) och deras funktion att dela med sig av applikationer till andra personer över internet. Anledningen till att båda programmen behövde användas var att sändardatorn använde operativsystemet Windows 2000 och mottagardatorn Windows XP. Om båda datorerna använt sig av Windows XP hade det räckt med att enbart använda MSN Messenger. Då detta kopplats upp (se bilaga H) kunde användaren på mottagarsidan både se och använda Mote Kommunikationsfönstrets funktioner.

6 Användartester

Anledningen till att tester genomförs är att hitta och rätta till användbarhetsproblem. Testerna hjälper utvecklaren att skapa en produkt som är lätt att använda och det finns ett behov av. Resultatet av väl genomförda tester kan bli en efterfrågad produkt med få problem. [Rubin, 1994]

6.1 Utrustning

Utrustningen som användes till testerna bestod av:

- Internetansluten bärbar dator, med all behövlig mjukvara installerad (se kapitel 3.2).
- 3 stycken mica2 med modifierat TinyDB och DelugeC installerat (se kapitel 5.2).
- 3 stycken mica2dot med modifierat TinyDB utan DelugeC installerat.
- Basstation bestående av en MIB510 och kopplad till den en mica2 med nodeid 0.
- Internetansluten stationär dator för att visa informationen från motsen och möjliggöra återkoppling. Förutom internetanslutning är enda kravet på denna dator att NetMeeting och MSN Messenger finns installerat.

6.2 Testpersoner

De praktiska testerna av systemet genomfördes av testpersonerna Håkan och Ann. När Håkan är på arbetet skulle han vilja minska sin oro för hur hans hustru Ann, som är hemma, mår med hjälp av en enkel, snabb och tyst trygghetskommunikation. Ann har den kroniska sjukdomen SLE (systemisk lupus erythematosus) som tillhör de reumatiska sjukdomarna. Vid SLE angriper immunförsvaret den egna kroppen och kan ge feber, illamående och trötthet, men också inflammationer på inre organ. Sjukdomen kännetecknas även av att den går i skov. Detta innebär att perioder där sjukdomen ger få symptom blandas med perioder där sjukdomen är mycket aktiv.[Infomedica]

Att sjukdomen går i skov där tillståndet varierar snabbt och kraftigt mellan bra och dåliga perioder bidrar till att skapa en osäkerhet hos anhöriga till personer med SLE. För Ann kan en dålig dag betyda att hon är orkeslös, har feber, mår illa, har kramper och mycket smärta. Detta kan resultera i att hon blir sängliggande eller i värsta fall måste åka till sjukhuset. Som

kontrast till detta kan Ann de dagar hon känner sig bra, vara aktiv och göra mycket av det som aktiva personer utan SLE gör.

Situationen, att ständigt vara beredd på nödsituationer, skapar en känsla av otrygghet och anspänning hos Håkan, som när han befinner sig på jobbet gärna vill veta hur Ann mår. Problemet är att Håkan inte vill ringa och störa när Ann ligger och sover. Håkan vill också gärna veta när Ann mår bra eftersom de då kan få möjlighet att göra saker tillsammans som de inte kan när Ann mår dåligt.

6.3 Genomförda tester

Bild 6.1 Utplacering av motes i Ann och Håkans hem

Testerna inleddes med att systemet (se kapitel 5.2.1-5.2.2) installerades i Håkan och Anns hem. Motsen placerades på lämpliga ställen i huset så att noderna med säkerhet skulle få kontakt med basstationen. På Håkans jobbdator installerades nödvändiga program så att han kunde se informationsfönstret (Mote Kommunikation) och genom knapptryckningar sända signaler direkt till motsen.

En kort genomgång av tekniken och det aktuella systemet och dess funktioner hölls innan det startades upp. Ann och Håkan fick också testa systemets huvudfunktioner så som att skaka motsen, mörklägga ljussensorn och trycka på knapparna i mote kommunikationsfönstret för att se hur systemet reagerade på detta. En kort introduktion till hur uppkopplingen för överföring av information till Håkans dator på jobbet går till hölls också. Den delen av systemet startades upp först morgonen efter (se bilaga H) och vid uppkopplingstillfället fanns även jag med som support.

En enkel instruktion lämnades även till testpersonerna för att underlätta en omstart om systemet av någon anledning skulle haverera (se bilaga G). När allt fungerade som det skulle fick Ann och Håkan genomföra testerna på egen hand, med möjlighet till teknisk support från mig.

Under de första dagarnas tester uppstod en del problem. Det visade sig att systemet skickade ganska många falska skaklarm, vilket motsvarade utsignalen ”Allt är OK!” (se bild 6.2).

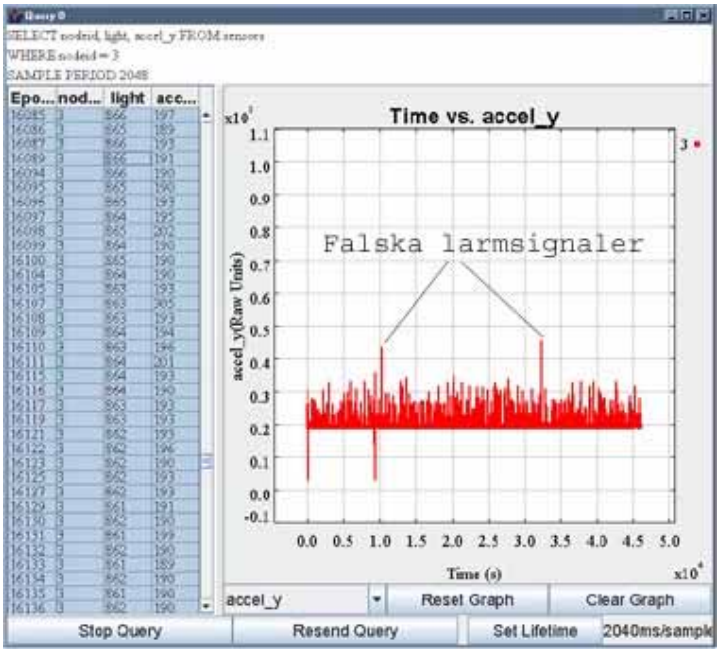


Bild 6.2 Visar falsklarmen vid de inledande testerna

Som första försök att lösa detta byttes signalindikatormote och gränsvärdena för larmning justerades. Systemet fungerade sedan bättre, men fortfarande skickades falska signaler då och då.

För att helt lösa problemet ändrades gränsvärdena så att enbart negativa accelerometervärden resulterade i att larmsignal skickades, då det visade sig att de falska larmsignalerna enbart gav positiva utslag. Efter denna justering fungerade systemet bra och skickade inte en enda falsk signal under resten av testperioden.

Ytterliggare ett problem uppstod efter några dagar av tester, noderna slutade helt sända information till basstationen. Efter flertaliga, misslyckade, försök att nollställa noderna insåg jag att problemet krävde vidare undersökning. Det första som testades var att plocka bort alla mica2dots i nätverket för att eliminera den möjliga felkällan att noderna felaktigt programmerat om varandra. Felet visade sig dock vara att batteriet tagit slut i alla mica2. Batterierna ersattes med nya och alla mica2 programmerades om. Efter detta fungerade systemet igen och testerna kunde återupptas.

6.4 Utvärdering av tester

Efter en dryg veckas tester utvärderades det använda systemet. Utvärdering genomfördes i intervjuform. Ann överlämnade även ett dokument med sina tankar och funderingar kring det testade systemet.

6.4.1 Utvärdering med Håkan

Håkan tyckte att systemet skapade en tydligt ökad närvarokänsla, där han genom att titta på dataskärmen när han arbetade vid den, eller bara passerade sitt kontor i korridoren, fick en känsla av hur Ann just då mådde. Detta minskade också hans oro avsevärt.

Håkan tyckte att de största svårigheterna med systemet var uppkopplingen av förbindelsen så att han, på distans, kunde se information från motsen och skicka meddelanden till dem. När detta skulle göras var Håkan och Ann tvungna att befinna sig framför vardera datorn samtidigt, vilket kändes lite krångligt och resulterade i att Håkan och Ann inte använde systemet varje dag. Håkan var också tvungen att vänta tills Ann vaknade, innan uppkopplingen gjordes.

Ibland kunde också systemet, på grund av överföringen av information via internet och användandet av en gammal dator, vara något långsamt och Håkan visste inte alltid säkert om de knapptryckningar han gjorde hade registrerats av systemet.

Håkan tyckte att det kommandofönster han använde i huvudsak fungerade bra. Det enskilt bästa med fönstret var ändringen av bakgrundsfärg som motsvarande det senast sända meddelandet från motsen. Det skulle dock vara en fördel om fönstret alltid låg överst på datorns skrivbord, vilket skulle underlätta användandet ännu mer. Han ansåg även att klockslaget då ett meddelande tagits emot kunde ha varit större och tydligare. Ibland glömde han att titta på klockslaget då de mottagna meddelandena skickats, innan han rensade kommandofönstret och när det väl var gjort gick det inte att ångra. Håkan kände att någon form av ytterligare loggning av de meddelanden som tagits emot, förutom meddelandena i kommandofönstret hade varit bra.

Håkans möjligheter till kommunikation bestod av signalerna ”Kontakta Mig!” vilket renderade ett pip på alla motes. Under testerna kom Håkan att mer använda den signalen som en fråga om hur det är, att han sett Anns meddelande, men också som en signal för att säga ”- Jag tänker på dig”. Den enda funderingen Håkan hade kring pipet var signalstyrkan. Om Ann låg och sov ville han inte skicka en pipsignal som möjligen kunde väcka henne. Ann tyckte däremot att ljudnivån var tydlig och bra men inte så hög att hon skulle bli väckt av den.

Håkan hade även möjligheten att rensa mote kommunikationsfönstret och använde den signalen även för att meddela Ann att han sett hennes meddelande. Knappen ”Har sett ditt meddelande” använde han nästan inte alls eftersom han visste att det var svårt för Ann att se den signalen som bestod av att en liten gul lysdiod tändes eller släcktes.

6.4.2 Utvärdering med Ann

Ann tyckte att systemet var snabbt och enkelt att använda. Hon saknade dock feedback från systemet i form av signal som indikerade att det meddelande hon ville skicka hade registrerats. Saknaden av sådan signal gjorde att hon ofta tittade på dataskärmen för att vara säker på att meddelandet gått fram. Ytterligare en form av feedback som hon kände behöver vidareutvecklas är de signaler Håkan kunde sända till motsen. Möjligheten att veta när Håkan sett hennes meddelande och även andra enkla meddelanden kan öka användbarheten avsevärt.

I det testade systemet fanns två möjligheter till kommunikation mellan dator och mote. Den ena var ett pip på alla befintliga motes och den andra att den orange lysdioden tändes. Ann ansåg att pipet fungerade bra, men att lysdioden syntes mycket dåligt.

Ann tyckte att motsen (mica2) var något stora för att bära på sig. Storleken på mica2dot hade nog varit bättre anpassad till den sortens bruk.

Hon reagerade även negativt på programdelningsmetoden då det kändes lite otäckt att en annan person kunde ta kontroll över hennes dator.

Under testperioden kände sig Ann också något begränsad eftersom hon hindrades från att använda internet när systemet var uppkopplat.

Överlag tyckte hon att tekniken är mycket intressant och med lite vidareutveckling kan hon mycket väl tänka sig att använda systemet för mer långvarigt bruk.

6.4.3 Ann och Håkans allmänna åsikter kring det testade systemet

Sammantaget så tyckte Ann och Håkan att systemet var kul att använda och att det, framförallt för Håkan, ökat känslan av närvaro dem emellan. Det kunde ha varit bra med några fler meddelandenivåer, men de som fanns i det testade systemet är fullt tillräckligt för att det skall vara värt att använda och trots de nuvarande begränsningarna skulle Håkan och Ann kunna tänka sig att använda nätverket för mer långvarigt bruk. De ansåg att antalet nivåer inte bör överstiga 4-5 eftersom systemet då riskerar att bli krångligt att använda. Fler meddelandenivåer kan skapas genom ytterliggare sensorer, exempelvis fuktighetsmätare och vibratorplatta, eller genom kombinationer av befintliga signalmöjligheter. Vidare hade möjligheten att ändra signalernas innebörd beroende på hur aktiv Anns sjukdom är, varit mycket positivt.

Håkan lämnade följande önskemål till ett system där en tyst förfrågan från honom kan ge svar i fyra nivåer. Han gav också

förslag till vad dessa fyra svarsalternativ i praktiken skulle kunna innebära för honom.

Svar på förfrågan		Konkret handling av Håkan
- Bättre än vanligt	⇒	- Åk hem och ta vara på det
- Som vanligt	⇒	- Inget speciellt
- Lite sämre än vanligt	⇒	- Underlätta genom att t.ex. handla hem mat
- Mycket sämre än vanligt	⇒	- Vill åka hem, även om Ann tycker att han inte behöver

En fråga som absolut måste lösas är ett bättre system för att överföra informationen från motsen till annan användare, som kändes både krångligt att koppla upp och blev ett orosmoment då datorns kontroll överlämnades till någon annan.

7 Diskussion och slutsatser

Projektets syfte var att, av ett befintligt system motes, skapa en ny kommunikationsmöjlighet för att ge människor en ökad trygghet i deras vardag.

Jag tycker att jag till viss del lyckades skapa ett sådant system, men att det fortfarande finns saker som måste förbättras innan systemet fungerar tillräckligt bra för att på ett smidigt sätt gå använda i det dagliga livet.

En av de saker som måste lösas är överföringen av information via internet, vilken i det testade fallet fick en lite hastig lösning som inte är hållbar i längden.

Mycket av projekttiden gick åt till att installera och modifiera grundkomponenterna i systemet. Den delen av arbetet fick en mycket större del än det från början var tänkt, men var tvunget att lösas för att överhuvudtaget kunna utveckla något system. De många problemen, dåliga hjälpinstruktioner från tillverkare och mycket bristfällig support var mycket frustrerande och gjorde att projektet ibland kändes helt hopplöst att genomföra. Den enda fördelen med alla inledande bekymmer var att jag var tvungen att lära mig från grunden hur systemet fungerade, vilket underlättade det vidare utvecklingsarbetet. Att systemet inte är i formen ”plug-and-play” är helt förståeligt men att det skulle vara så stora uppstartsvårigheter förvånade mig mycket, särskilt som det är ett uttalat mål hos tillverkarna att minska svårigheterna för icke avancerade användare.

När de inledande problemen lösts fungerade modifieringen av TinyDB mjukvaran, som låg till grund för testapplikationen, bra trots att det även här var svårt att hitta bra information om systemets uppbyggnad.

Fungerade då tekniken med trådlösa sensornätverk för att lösa uppgiften?

Svaret på det är nog både ja och nej. Det befintliga systemets motes har kanske lite för stora begränsningar, särskilt storleksmässigt, för att vara optimala. Detta märktes också tydligt under användartesterna. En av de viktigaste frågorna som måste lösas är energiförsörjningen. Idag drivs många motes med AA batterier och trots de stora batterierna är livslängden mycket begränsad. De motes som nu börjar komma på marknaden har förbättrats avsevärt med mycket små Li-jon batterier som dessutom är uppladdningsbara. Detta är ett mycket stort steg mot att skapa motes som utan obehag går att bära på kroppen.

Tekniken som koncept med trådlösa nätverk bestående av självkonfigurerande noder känns däremot som en mycket bra lösning på problemet. Ett alternativ för att minska det använda systemets begränsningar kunde ha varit att addera ytterligare finesser till sensorkorten, exempelvis högtalare, sensor för luftfuktighet eller fler lysdioder. Detta skulle ha ökat antalet meddelandenivåer, något som efterfrågades av testpersonerna Ann och Håkan. En annan väg att gå kunde ha varit att, istället för att direkt använda sensorerna, använda tryckknappar för att sända meddelanden och enbart utnyttja nätverksdelen av systemet.

Trots svagheterna i systemet kändes testerna som genomfördes och responsen från testpersonerna mycket positiva och sporrade till att vidareutveckla systemet ytterligare. Trots att uppkopplingen var krånglig och att meddelandenivåerna kunde ha varit fler, så fick testpersonerna och då främst Håkan en ökad känsla av trygghet vilket var huvudmålet med systemet.

Som jag tidigare uttryckt så tror jag mycket på tekniken och att fler och fler ’mjuka’ applikationer kommer att skapas inom den närmaste tiden. Detta både för att utvecklingen går mot mindre och energisnålare motes, vilket i sig ökar möjliga användningsområden, men också för att utvecklare börjar få upp ögonen för att motes kan användas till annat än statisk övervakning.

Referenser

Internet

Crossbow Getting Started Guide:
http://www.xbow.com/Support/Support_pdf_files/Getting_Started_Guide.pdf ,(2005-01-20)

Crossbow Mote-kit 5040:
http://www.xbow.com/Products/productsdetails.aspx?sid=69, (2005-02-05)

Crossbow New Products:
http://www.xbow.com/Products/new_product_overview.htm (2005-06-02)

CodeBlue:
http://www.eecs.harvard.edu/~mdw/proj/codeblue/ (2005-05-30)

Cygwin:
http://www.cygwin.com/, (2005-05-25)

Moteiv:
http://www.moteiv.com/ (2005-05-30)

NesC:
http://nescc.sourceforge.net/ , (2005-05-25)

How Stuff Works:
http://computer.howstuffworks.com/mote.htm (2005-01-28)

Infomedica:
http://www.infomedica.se/artikel.asp?CategoryID=15010 , (2005-05-23)

Intel, Ad Hoc networks:
ftp://download.intel.com/technology/silicon/mooreslaw/download/EML_adhoc.pdf , (2005-01-28)

TinyDB:
http://telegraph.cs.berkeley.edu/tinydb/overview.html, (2005-02-23)

Tryckt Material

Eftring, Håkan.(1999). The Useworthiness of Robots for People with Physical Disabilities. Doctoral Dissertion. ISBN 91-628-3711-7

Rubin, Jeffrey.(1994). Handbook of Usability Tesing. John Wiley & Sons, ISBN 0471594032.

Bilder

1. http://www.xbow.com/Products/productsdetails.aspx?sid=69 (2005-06-07)
2. http://www.xbow.com/Products/new_product_overview.htm (2005-06-07)
3. http://www.moteiv.com/products.php (2005-06-07)
4. http://www.eecs.harvard.edu/~mdw/proj/codeblue/ (2005-06-07)

Bilaga A, Installation av programvara och programmering av motes

Installation av programvara

- Kopiera mappen Surge-View (från installations cd:n) och lägg den i mappen C:\Program Files. Öppna sedan mappen (i Program Files), skapa en genväg till Serialforwarder.exe och lägg denna på skrivbordet.
- Installera TinyOS 1.1.0 genom att på installations cd:n öppna mappen TinyOS Install. I den mappen finns en fil tinyos-1.1.0-1is, klicka på den. Installationen startar nu. Detta kan ta ganska lång tid! Installera till mappen C:\Program Files\tinyos.
- Uppdatera till TinyOS 1.1.7. Kopiera filen tinyos-1.1.7July2004cvs-1.cygwin.noarch till mappen som du installerat TinyOS i. Installera genom att starta Cygwin (skall nu var installerat). Ställ dig i den katalog som du lagt filen i och skriv: rpm --force --ignoreos –Uvh tinyos-1.1.7July2004cvs-1.cygwin.noarch.rpm. Installationen startar.
- Kopiera xbowfiler. Gå till mappen TinyOS updates på installations cd:n. Kopiera filen Xbow.rar (packad fil) och lägg denna i mappen C:\Program Files\tinyos\cygwin\opt\tinyos-1.x\contrib. Packa upp filen här, när detta är klart skall det finnas en mapp xbow direkt under contrib mappen.

Du är nu redo att börja använda motsen.

Programmera mica2 och mica2dot

Steg ett är att koppla ihop mib510 med datorns com-port. Exemplen förutsätter att du har den kopplad till com1. Efter att detta är gjort skall den av mica2 eller mica2dot som skall programmeras sättas fast på mib510. Sätt på strömmen till mib510. Strömförsörjningen till mica2 skall vara av. För mica2dot måste batteriet tas ur.

Testapplikationer

Blink är den enklaste testapplikationen för att se att programmeringen av motsen fungerar som den ska. Installera Blink genom att:

1.Starta Cygwin

2.Skriv cd c:\Program Files\tinyos\cygwin\opt\tinyos-1.x\contrib\xbow\apps\Blink

3. Skriv make mica2 install mib510,com1

Klart! Du kan nu ta bort mica2 eller mica2dot från mib510 och slå igång den. Den röda lysdioden skall nu blinka.

Testa applikation för att sända och ta emot signaler mellan noder:

- Starta Cygwin.
- Sätt den mica2 som du vill ska skicka informationen på mib510.
- Skriv cd c:\Program Files\tinyos\cygwin\opt\tinyos-1.x\contrib\xbow\apps\CntToLedsAndRfm
- Skriv make mica2 install mib510,com1
- Ta bort den första mica2 och sätt fast den mica2 som du vill ska ta emot meddelanden på mib510.
- Skriv cd c:\Program Files\tinyos\cygwin\opt\tinyos-1.x\contrib\xbow\apps\RfmToLeds
- Skriv make mica2 install mib510,com1

Klart! Ta bort den andra mica2 från mib510 och slå på de båda programmerade mica2. Signalen skall nu skickas mellan dessa två.

Bilaga B, Ändring av frekvens

Ändra frekvens och kör programmet Surge

1. I mappen C:\Program Files\tinyos\cygwin\opt\tinyos-1.x\contrib\xbow\apps finns en fil MakeXbowlocal. Öppna den med lämpligt program, exempelvis Programmers Notepad eller Emacs. Se till att alla CFLAGS =…… kommateras ut. EX: CFLAGS=… blir #CFLAGS=….. Spara filen.
   Se också till att det på raden DEFAULT_LOCAL_GROUP=125 står just 125. Annars ändra detta. Spara filen.
2. Öppna Cygwin.
   Börja med att programmera alla mica2. Skriv cd c:\Program Files\tinyos\cygwin\opt\tinyos-1.x\contrib\xbow\apps\Surge_Reliable\
3. Sätt fast den första mica2 du vill programmera i mib510 och sätt på mib510's ström.
4. Skriv CFLAGS=-DCC1K_DEF_FREQ=869614000 make mica2. Detta programmerar mica2 till att använda frekvensen 869,614Mhz.
5. Skriv make mica2 reinstall,<nodens nummer> mib510,com1
   Upprepa detta för alla mica2.
6. Skriv cd c:\Program Files\tinyos\cygwin\opt\tinyos-1.x\contrib\xbow\apps\Surge_Reliable_Dot\
7. Sätt fast den första mica2dot du vill programmera i mib510 och sätt på mib510's ström.
8. Skriv CFLAGS=-DCC1K_DEF_FREQ=869614000 make mica2dot
9. Skriv make mica2dot reinstall,<nodens nummer> mib510,com1
   Upprepa för alla mica2dot.
10. Sätt fast mica2 med nodeid 0 på mib510.
11. Starta programmet SerialForwarder. Tryck på Stop server. I rutan Mote Communication skriv serial@COM1:mica2. Tryck på Start server.
12. Kör kommadoprompten, CMD och skriv cd \Program Files\Surge-View
13. Skriv Surge 125

Programmen startas nu och efter en stund skall alla noder hittas.

Alternativt sätt att ändra frekvensen

- I mappen C:\ Program Files\tinyos\cygwin\opt\tinyos-1.x\contrib\xbow\apps finns en fil MakeXbowlocal. Öppna den med lämpligt program. Se till att alla CFLAGS =…… kommateras ut. EX: CFLAGS=… blir #CFLAGS=….. Spara filen. Lägg till en rad med: PFLAGS += -DCC1K_DEF_FREQ=869614000
- Se också till att det på raden DEFAULT_LOCAL_GROUP=125 står just 125. Annars ändra detta. Spara filen.

Klart! Man behöver nu INTE skriva CFLAGS=…. när man ska omprogrammera mica2/mica2dot!

Bilaga C, Programmering via radiolänk

Programmera Mica2 och Mica2Dot via Radiolänk (DelugeC)

För att kunna göra detta måste man först uppdatera till TinyOS 1.1.8 eller högre. Efter att detta är gjort skall följande göras:

1. Funktionen make har gjorts om vilket innebär att följande skall läggas till i min egen Makefile (OBS!!! inte i Makerules). Antag att ändringen skall göras på applikationen Blink i mappen C:\Program Files\tinyos\…..\opt\apps\Blink. I Makefilefilen i mappen Blink skall då läggas till: TOSMAKE_PATH += $(TOSDIR)/../apps/Blink/make. Detta skall skrivas före texten include ../makerules. SPARA!
2. Gå till mappen …./tinyos-1.x/tools/make/. I denna mapp skall det finnas en fil som heter Makelocal. Gör det inte det så skapa en sådan fil. I den filen skriver du sedan: TINYOS_NP = BNP. SPARA!
3. Testa att allt funkar som det ska genom att prova göra make på Blinkapplikationen. Gå till mappen tinyos-1.x/apps/Blink och skriv: make mica2 install mib510,com1.Funkar detta är det bara att fortsätta!
4. Gå till mappen tinyos-1.x/apps/TestDeluge/GoldenImage. Skriv: make mica2 install,nodnummer mib510,com1.
5. När detta är klart, starta upp Serialforwarder och skriv serial@COM1:mica2. Starta servern (skall hållas öppen under resten av installationen).
6. Skriv java net.tinyos.tools.Deluge --ping. Detta skall ge resultatet att nodens information visas.
7. Nu måste även koden som skall köra programmet ändras. Öppna Blink.nc och på raden components läggs DelugeC till. Man skriver också till Main.StdControl -> DelugeC; Spara!
8. I Cygwin, gå till mappen Blink och skriv make mica2. När detta är klart gå till mappen Blink/build/mica2 och skriv java net.tinyos.tools.DelugeC --ihexfile main.ihex --imgnum 0. Detta installerar programmet på noden.
9. När detta är klart skriv: java net.tinyos.tools.DelugeC --reboot --imgnum 0. Detta startar om noden och startar applikationen.

Upprepa detta på mica2dot med enda skillnaden att servern skall startas med serial@COM1:mica2dot.

Kända problem och dess lösningar

Första gången man installerar programmen måste man göra detta på alla noder genom att sätta dom på mib510 och gå igenom alla ovanstående steg, annars kommer inte noderna att vilja uppdatera sig när nollan omprogrammeras.

Ytterliggare ett bekymmer är att mica2dot måste programmeras genom att först sätta en mica2dot som 0:a och via den programmera om alla mica2dots. OBS!!! När detta görs måste alla mica2 vara avslagna annars kommer signalen även att gå till mica2 som då kommer sluta fungera. Skulle detta ändå hända får man helt enkelt starta om och programmera om sina mica2 igen. Ett smart sätt kan dock vara att se om man får kontakt med sina mica2, genom att pinga dom, om detta fungerar kan man bara programmera om 0:an och göra reboot på denna. Detta gör då att alla mica2 skall rebootas. Om man inte kan pinga sina mica2 alls, så kan man installera exempelvis Blinkapplikationen (med standardinstallationen) sedan se om man kan pinga mica2, och börja om igen.

Man kan också köra olika program på mica2 med nodeid 0 och resten av noderna. Detta kan göras genom att först radioprogrammera ut det program man vill att nätets noder skall använda och sedan installera 0:ans program med standardinstallationen. (EX. CntToLedsAndRfm och RfmToLEds).

Uppdatera till TinyOS 1.1.8-1.1.10

Detta gav stora problem då fel hela tiden uppstod eftersom installationen kräver att nesC >=1.1.1 är installerat. Detta kan hämtas på: http://www.tinyos.net/dist-1.1.0/tinyos/windows/

1. Välj att hämta hem: nesc-1.1.1-2.cygwin.i386.rpm
   Även filen
   tinyos-1.1.8Nov2004cvs-1.cygwin.noarch.rpm eller annan version hämtas härifrån.
2. Lägg filerna i c:\Program Files\tinyos
3. Starta cygwin. Ställ dig i c:/Program Files/tinyos och skriv:
   rpm --force --ignoreos -Uvh nesc-1.1.1-2.cygwin.i386.rpm
4. När denna installation är färdig skriv:
   rpm --force --ignoreos -Uvh tinyos-1.1.8Nov2004cvs-1.cygwin.noarch.rpm

Klart!

Bilaga D, Installation av TinyDB

Installation av TinyDB

TinyDB finns som en del av TinyOS senare än 1.1.0. För att installera detta gör följande:

1. Börja med att verifiera att CLASSPATH innehåller JLex.jar, cup.jar och plot.jar. Filerna ska finnas i mappen …/tinyos-1.x/tools/java/jars. Finns dom inte där, öppna ett Cygwin fönster och skriv:

   export CLASSPATH='C:/Program Files/tinyos/tinyos-1.x/tools/java/javapath'
2. cd tinyos-1.x/tools/java/net/tinyos/tinydb

   skriv make
3. När detta är klart kan du gå tillbaka och installera TinyDB på dina motes. Om du vill använda motsen med DelugeC måste du göra ändringarna i den här ordningen och inte som det står i orginalinstruktionerna för då kommer det inte att fungera alls. För att göra detta tillsammans med DelugeC (möjliggör omprogrammering via radiolänk) gör följande:

   - Ändra filen C:\Program Files\tinyos\cygwin\opt\tinyos-1.x\apps\TinyDBApp\Makefile så att den får följande utseende:

     ```
     COMPONENT=TinyDBApp
     SENSORBOARD=micasb
     #Under detta är tillagt
     XBOWROOT=%T/../opt/tinyos-1.x/tos
     PFLAGS=-I %T/lib/Util -
     I%T/lib/Attributes -I%T/lib/Commands -
     I%T/lib/TinyDB -I%T/lib/Route -
     I%T/lib/Queue  -I%T/lib/FS -
     I%T/lib/TinyDB/Aggregates -fno-strict-
     aliasing
     #Under detta är tillagt
     TOSMAKE_PATH +=
     $(TOSDIR)/../CONTRIB/eyes/make
     MSG_SIZE=49
     #Under detta är tillagt
     include ../Makelocal
     include $(TOSROOT)/tools/make/Makerules
     #include ../Makerules
     ```

   - Ändra i filen C:\Program Files\tinyos\cygwin\opt\tinyos-

1.x\apps\TinyDBApp\TinyDBApp så att den får följande utseende.

```
implementation {
components Main, TupleRouter, DelugeC;
Main.StdControl -> DelugeC;
Main.StdControl -> TupleRouter;
}
```

- Om man ändrar ofta mellan att använda mica2 och mica2dot som basstation kan det vara smidigt att gör följande ändring. I filen C:\Program Files\tinyos\cygwin\opt\tinyos-1.x\tools\java\net\tinyos\tinydb\tinydb.conf gör följande ändring:

```
%comm-string:serial@COM1$57600
%comm-string:serial@COM1$19200
comm-string:sf@localhost$9001
```

Anledning att göra detta är att mica2 och mica2dot använder olika bauderate och ska man variera mellan dessa två ofta är det enklare att låta SerialForwarder styra vilken av de två man använder, istället för att vara tvungen att ändra i koden varje gång. Nackdelen är att man måste starta SerialForwarder innan man kan köra TinyDBMain.

- Installera TinyDBApp till valfri image. För mer info se programmering med DelugeC.

4. Starta SerialForwarder. Skriv serial@COM1:mica2 eller serial@COM1:mica2dot beroende på vilken av dessa du använder som nod 0.
5. Du är nu redo att testa programmet. Skriv cd …./tinyos-1.x/tools/java
6. java net.tinyos.tinydb.TinyDBMain. För att få information om hur programmet fungerar se http://telegraph.cs.berkeley.edu/tinydb/tutorial/tinydb.html

Att använda databas tillsammans med TinyDB

Först måste databasen startas upp. Börja med att i Cygwin skriva:

1. cd c:\Program Files\tinyos\cygwin\opt\tinyos-1.x\
2. ipc-daemon &
3. export PGDATA=/pgdata
4. mkdir /pgdata
5. initdb
6. Starta postmastern genom att skriva pg_ctl start
7. Öppna filen c:\Program Files\tinyos\cygwin\pgdata\postgresql.conf och byt ut raden #tcpip_socket = false mot tcpip_socket =true. Det ska alltså INTE vara något tecken # före den ändrade raden.
8. Starta om postgresql med pg_ctl restart
9. Skriv: export CLASSPATH=$CLASSPATH:tinyos-1.x/tools/java/jars/pgjdbc2.jar

10. Starta databasen med: createdb task
11. Skapa en användare med: createuser tele
12. Skapa en tabell där värden kan sparas undan med: psql task
13. create table queries (query_table varchar(10), query_time timestamp, query_string varchar(500));
14. Stäng detta med \q
15. Starta upp TinyDBMain som vanligt och kryssa i rutan log to database när du gör din query, så sparas värdena undan. Filen kommer att döpas till qX. Där X är numret på den query som du just kört.

När du är färdig med dina queries kan du enkelt konvertera de lagrade databasvärdena till ett excellark eller annat format genom att:

1. Stäng först ner TinyDB.
2. I cygwin skriv: psql task.
3. copy <query_table> to '<filename>.csv' delimiter ',';
4. Ett riktigt exempel kan då vara: copy q1 to '/tmp/q1.csv' delimiter ','; Detta lägger arket i mappen tinyos-1.x/tmp.

Allmänna tips kring TinyDB och databasen task.

Varje gång Cygwin eller datorn startas om måste följande göras för att enkelt starta upp och använda den befintliga databasen:

1. Starta cygwin, skriv: ipc-daemon &
2. export PGDATA=/pgdata
3. Ta bort filen C:\Program Files\tinyos\cygwin\pgdata\postmaster.pid. Det skapas en ny sådan när du startar om postmaster. Tar man inte bort den uppstår en konflikt när man försöker lagra värden till databasen i TinyDB.
4. Starta postmaster med pg_ctl start. Klart!
5. Starta upp TinyDBMain som vanligt.

Om du tidigare använt TASK och databas kan det vara lämpligt att ta bort allt mappen /pgdata mha rm –fr /pgdata annars kan koflikter uppstå.

Anledningen till att jag kallar databasen för task istället för tinydb är för att slippa ändra i configfilen som bestämmer viken databas den letar efter då jag växlar mycket mellan TinyDB och TASK.

När querien startas kan det ofta stå ERROR: Relation 'seqno' already exists ERROR: table "q3" does not exist. Detta är dock ingenting att oroa sig för, värdena sparas ändå undan i tabellen q3.

Bilaga E, Installation av TASK

Installation av TASK

För att kunna använda TASK måste en lokal databas skapas. Jag fick testa ett antal olika varianter innan det fungerade för mig. Jag tror att det absolut enklaste sättet att starta upp denna är att:

1. I Cygwin skriv ipc-daemon &
2. chmod a+rwx /tmp
3. chmod a+rx /usr/bin /usr/bin/*
4. Du är nu redo att installera TASK på dina motes. Ändra filen C:\Program Files\tinyos\cygwin\opt\tinyos-1.x\apps\TASKApp\Makefile så att den får följande utseende:

```
COMPONENT=TASKApp
SENSORBOARD=micasb
PFLAGS= -I%T/lib/Util -I%T/lib/Attributes -
I%T/lib/Commands -I%T/lib/TinyDB -
I%T/lib/Route -I%T/lib/Queue  -I%T/lib/FS -
I%T/lib/TinyDB/Aggregates -fno-strict-
aliasing
MSG_SIZE=49
#Under detta är tillagt
XBOWROOT=%T/../opt/tinyos-1.x/tos
#PFLAGS=-I %T/lib/Util -I%T/lib/Attributes
-I%T/lib/Commands -I%T/lib/TinyDB -
I%T/lib/Route -I%T/lib/Queue  -I%T/lib/FS -
I%T/lib/TinyDB/Aggregates -fno-strict-
aliasing
#Under detta är tillagt
#PFLAGS= -I$(XBOWROOT)/platform/mica2
#Under detta är tillagt
TOSMAKE_PATH +=
$(TOSDIR)/../CONTRIB/eyes/make
MSG_SIZE=49
#Under detta är tillagt
include ../Makelocal
include $(TOSROOT)/tools/make/Makerules
#include ../Makerules
```

5. Ändra filen C:\Program Files\tinyos\cygwin\opt\tinyos-1.x\apps\TASKApp\TaskApp så att den får följande utseende:

```
implementation {
components Main, TupleRouter, FieldApp,
DelugeC;
Main.StdControl -> DelugeC;
```

```
    Main.StdControl -> TupleRouter;
    Main.StdControl -> FieldApp;
  }
```

6. Installera TASKApp till valfri image. För mer info se programmering med DelugeC.
7. Starta SerialForwarder och använd som vanligt: serial@COM1:mica2
8. cd …./tinyos-1.x/tools/java/net/tinyos/task/taskserver
   Skriv sh ./setup-task-db.sh. Detta startar upp databasen task.
9. cd …./tinyos-1.x/tools/java
   Skriv java net.tinyos.task.tasksvr.TASKServer &
10. Öppna ett nytt Cygwinfönster och skriv cd …/tinyos-1.x/tools/java
11. java net.tinyos.task.taskviz.TASKVisualizer localhost. Detta startar upp TaskVisualizer. För mer information om hur det fungerar se: http://www.xbow.com/support/Support_pdf_files/TASK_Getting_Started_Guide_v0.1.pdf

Allmänna tips kring TASK och framförallt då databasen task.
När jag har använt detta verkar det som att funktionen som ritar upp grafer över sensorernas värden enbart fungerar för den första query som görs sedan programmet har startat. Efterföljande queries klarar inte detta, men värdena sparas ändå undan till databasen. Vill man grafiskt följa sina värden får man helt enkelt starta om programmet.
Det kan även vara lite problem att starta upp taskserver. Det enklaste sättet att komma runt detta verkar vara att stänga databasen genom:
pg_ctl stop –D /pgdata. Starta sedan upp databasen igen genom att skriva:
pg_ctl start –D /pgdata –l /pgdata/logfile. Detta kan exempelvis behöva göras efter omstart av datorn.
Skulle inte detta fungera kan man vara tvungen att ta bort de lagrade filerna och starta om databasen. OBS! Detta raderar tidigare sparade sensorvärden. Skriv:
Rm –fr /pgdata. Efter detta görs installationen om från steg 8 ovan.

Bilaga F, Modifierade klasser

CmdFrame.java
Endast de ändringar som är gjorda i klasserna redovisas här. För fullständig dokumentation av de klasser som användes, kontakta författaren.

```
// $Id: CmdFrame.java,v 1.5.12.2 2003/08/18 22:09:44 cssharp Exp
$/*tab:4
```

Command frame presents a simple UI for sending a variety of TinyDB control messages into the network. See CommandMsgs for more info. @author smadden

```
public class CmdFrame extends JFrame {
   static final short BCAST_ADDR = (short)-1;
   MoteIF mif;
   public CmdFrame(MoteIF mif) {
   super("Mote Kommunikation");
   this.mif=mif;

/* Här definieras de nya knapparna*/
   JButton sounderOnButton = new
   JButton("Kontakta Mig!");
   JButton SetLedYAllButton = new JButton("Har
   sett ditt meddelande!");
```

```
        JButton clearButton = new JButton("Rensa
        Allt");

    /* Här under definieras de nya textfältet och
    textpanelen*/
        JPanel textPanel = new JPanel();
        JPanel moteIdPanel = new JPanel();

    //Har kommaterat ut onödiga knappar
        buttonPanel.setLayout(new GridLayout(3,1));
        buttonPanel.add(resetButton);
        buttonPanel.add(sounderOnButton);

    //Ändring gjord här under. Har lagt till knappar
        buttonPanel.add(SetLedYAllButton);

    //Definierar panelen där clearknappen finns
        clearPanel.setLayout(new BorderLayout());
        clearPanel.add(clearButton,
        BorderLayout.EAST);

    //Definierar textrutan
        ScrollPanel = new JScrollPane(textArea);
        textPanel.setLayout(new GridLayout(1,1));
        textArea.setRows(30);
        textArea.setLineWrap(true);
        textArea.setEditable(false);
        textArea.setBorder(new LineBorder(new
        java.awt.Color(0, 0, 0)));
        textArea.setBackground(new Color(0xeeeecc));
        Font font = new Font("Serif", Font.BOLD, 12);
        textArea.setFont(font);
        textArea.setForeground(Color.black);
        textPanel.add(ScrollPanel);
        textArea.insert("--INFORMATION FRÅN MOTSEN--",0);
        moteID.setText("0");
        moteID.setEnabled(false);
        moteIdLabel.setEnabled(false);
        moteIdLabel.setHorizontalAlignment(SwingConsta
        nts.RIGHT);
        broadcastBox.setSelected(true);
        broadcastBox.addActionListener( new
        ActionListener() {

    public void actionPerformed(ActionEvent evt) {
        boolean en = broadcastBox.isSelected();
```

```
   moteIdLabel.setEnabled(!en);
   moteID.setEnabled(!en);
   }
});
   box.add(buttonPanel);
   box.add(Box.createGlue());
   box.add(textPanel);
   box.add(clearPanel);
   getContentPane().add(box);
   setSize(225,400);

/* Här under ligger alla actionlisteners till
knapparna*/
resetButton.addActionListener( new
ActionListener() {
   public void actionPerformed(ActionEvent evt) {
       /*Date date = new Date();
       textArea.setBackground(new
       Color(0x99ff00));
       textArea.setForeground(Color.blue);
       textArea.append("\n");
       textArea.append(date+"\n");
       textArea.append("Du tryckte på
       resetknappen");
       textArea.append("\n");*/
       sendMsg(CommandMsgs.resetCmd(rcptId()));
       }
   });
   clearButton.addActionListener( new
   ActionListener() {
   public void actionPerformed(ActionEvent evt) {
       textArea.setText("");
       textArea.setBackground(new
       Color(0xeeeecc));
   textArea.setForeground(Color.black);
   textArea.append("--INFORMATION FRÅN MOTSEN--
");
sounderOnButton.addActionListener( new
ActionListener() {
public void actionPerformed(ActionEvent evt) {
   /*Date date = new Date();
   textArea.setBackground(new Color(0xff3333));
   textArea.setForeground(Color.white);
   textArea.append("\n");
   textArea.append(date+"\n");
```

```
   textArea.append("Du tryckte på Sounder On
   knappen!");
   textArea.append("\n");*/
   sendMsg(CommandMsgs.setSounderCmd(rcptId()));
   }
});

/* Metoder som skriver ut meddelnaden i CmdFrame
då vissa krav är uppfyllda*/
public static void varnaljus(String ljus, int
ljus2){
   if(ljus2<70 && ljus2!=0){
       Date date = new Date();
       textArea.setBackground(Color.blue);
       textArea.setForeground(Color.white);
       textArea.append("\n");
       textArea.append(date+"\n");
       textArea.append("RING MIG!");
       textArea.append("\n");
       try{
       Thread.currentThread().sleep(2000);
       }
       catch (InterruptedException e) {
           //Do nothing
             }
       }
   }
public static void varnatemp(String temp, int
temp2){
   if(temp2==510){
   Date date = new Date();
   textArea.setBackground(Color.red);
   textArea.setForeground(Color.white);
   textArea.append("\n");
   textArea.append(date+"\n");
       textArea.append(temp2+"grader -- Smoking hot!");
       textArea.append("\n");
       try{
       Thread.currentThread().sleep(2000);
       }
       catch (InterruptedException e) {
           //Do nothing
       }
   }
   }
```

```
public static void varnaaccel(String accel,int
accel2){
   if(accel2<=140){
       Date date = new Date();
       textArea.setBackground(Color.green);
       textArea.setForeground(Color.blue);
       textArea.append("\n");
       textArea.append(date+"\n");
       textArea.append("DET ÄR OK!");
       textArea.append("\n");
             }
             }
public static void skrivut(String s){
   Date date = new Date();
   textArea.append("\n");
   textArea.append(date+"\n");
   textArea.append(s);
   textArea.append("\n");
             }
```

CmdMsgs.java

```
//Allt här under är tillagt
public static Message YellowLedAllCmd(short
targetId) {
   CommandMsg cmdMessage = new
   CommandMsg(MSG_LEN);
   cmdMessage.set_nodeid(targetId);
   cmdMessage.set_seqNo(getNextSeqNo());
   //int red =
   setCommandName(cmdMessage,"SetLedR");
   //int green =
   setCommandName(cmdMessage,"SetLedG");
   int yellow =
   setCommandName(cmdMessage,"SetLedY");
   //cmdMessage.setElement_data(red++,(byte)2);
   //cmdMessage.setElement_data(green++,(byte)2);
   cmdMessage.setElement_data(yellow++,(byte)2);
   return cmdMessage;
      }
public static Message setSounder(short
owntargetId) {
   CommandMsg cmdMessage = new
   CommandMsg(MSG_LEN);
   owntargetId =1;
   cmdMessage.set_nodeid(owntargetId);
   cmdMessage.set_seqNo(getNextSeqNo());
```

```
   int pos = setCommandName(cmdMessage,
   "SetSnd");
   cmdMessage.setElement_data(pos++,(byte)0);
   cmdMessage.setElement_data(pos++,(byte)2);
   return cmdMessage;
      }
```

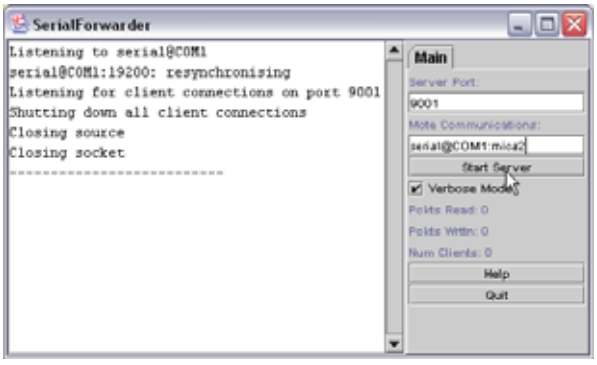
ResultFrame.java

```
Har ändrat här så att resultaten adderas till
tabellen och dessutom checkas mot valda
gränsvärden.
Varning sker vid under/överskridande av
gränsvärde
*/
public void addResult(QueryResult qr) {
   String ljus = "";
   int ljus2 = 0;
   String temp ="";
   int temp2 = 0;
   String accel ="";
   int accel2 = 0;
   Vector resultVector = qr.resultVector();
      if (resultVector == null ||
      resultVector.size() <= 1) return;
      if (qr.epochNo() > epochNo)
      epochNo = qr.epochNo(); //monotonically
      increasing
   tableModel.addRow(qr.resultVector());
   ljus += resultVector.get(2);
   ljus2 += Integer.parseInt(ljus);
   CmdFrame.varnaljus(ljus,ljus2);
   /*temp += resultVector.get(3);
   temp2 += Integer.parseInt(temp);
   CmdFrame.varnatemp(temp,temp2);*/
   accel += resultVector.get(3);
   accel2 += Integer.parseInt(accel);
CmdFrame.varnaaccel(accel,accel2);
resultTable.setRowSelectionInterval(tableModel.ge
tRowCount()-1,tableModel.getRowCount()-1);
JScrollBar bar = scroller.getVerticalScrollBar();
   if (bar != null) {
      bar.setValue(bar.getMaximum() + 128);
         }
```

Bilaga G, Uppstartinstruktioner; Förfrågan i testsystemet

1. Dubbelklicka på ikonen SerialForwarder. När detta startar tryck på knappen Stop Server.

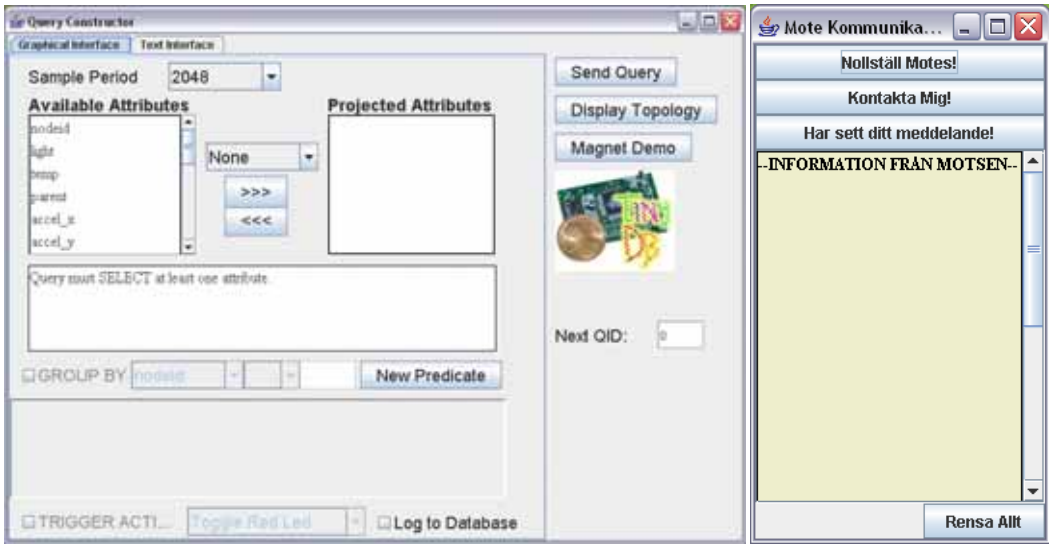
I rutan Mote Communications lägg till text så att det som står där är: serial@COM1:mica2



Tryck på knappen Start Server.
Klart! Du kan nu minimera detta fönster då du inte ska använda det igen.

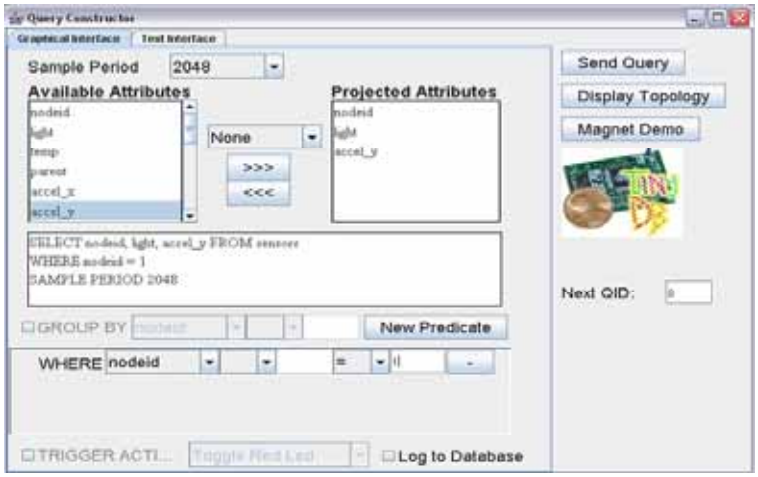
1. Klicka på ikonen Cygwin. I detta fönster, skriv:
    1. cd c:
    2. cd Program/tinyos/cygwin/opt/tinyos-1.x/tools/java
    3. java net.tinyos.tinydb.TinyDBMain

2. Nu startas TinyDB upp, med 2 fönster.

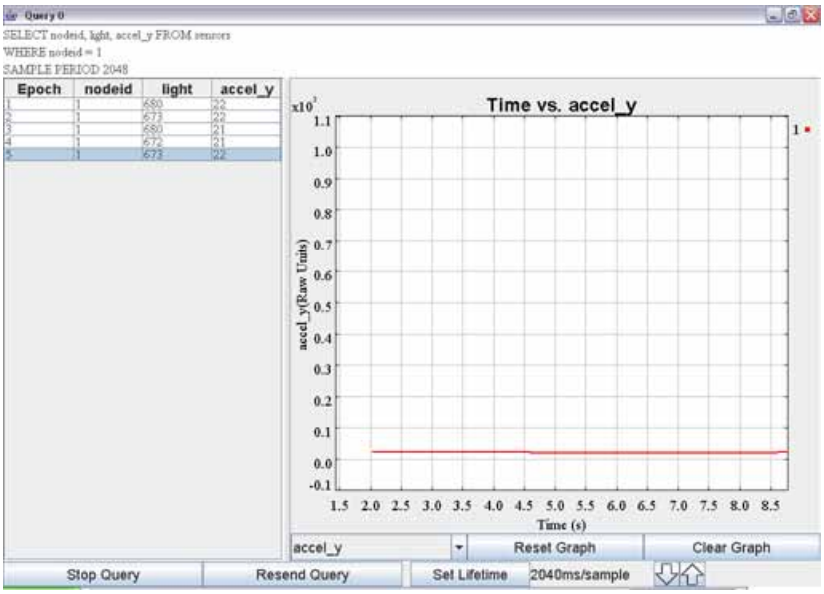


3. För att nu starta upp själva förfrågningen gör följande i Query Constructor fönstret:
    1. I menyn Available Attributes klicka på nodeid och sedan på knappen >>>
    2. I menyn Available Attributes klicka på light och sedan på knappen >>>
    3. I menyn Available Attributes klicka på accel_y och sedan på knappen >>>
    4. Klicka på knappen New Predicate. Det kommer nu upp en undermeny. I den skrivbara rutan längst till höger skriv en etta (1).

Query Constructor skall nu ha följande utseende:



Klicka på knappen Send Query. Nu öppnas ytterliggare ett fönster med en tabell och en graf. Om värden börjar fyllas på i dessa så fungerar allt som det ska. Om det inte gör det, prova trycka på knappen Stop Query och sedan Resend Query.



KLART!! Alla fönster utom Mote Kommunikation kan nu minimeras.

Bilaga H, Uppstartinstruktioner; Överföring av information via internet

De två programmen som kommer att användas är:

- MSNMessenger
- NetMeeting

1. Dubbelklicka på ikonen NetMeeting. Detta öppnar nedanstående fönster.

2. Klicka på ikonen MSN Messenger. Detta öppnar ett inloggningsfönster.

3. Klicka på logga in. Detta öppnar ytterligare ett fönster.

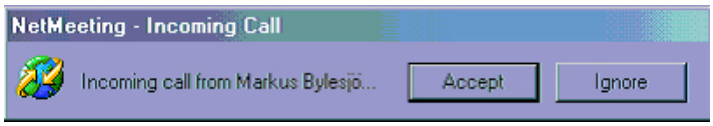
4. I rutan inloggningsnamn skriv: testhemma@hotmail.com
   Lösenord: testar
   Klicka på OK

MSNMessenger öppnas nu

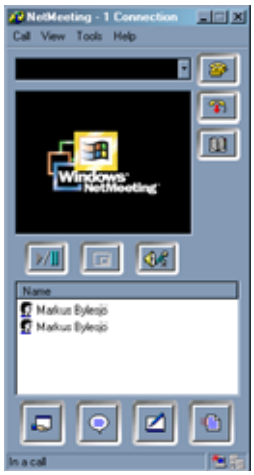
5. Dubbelklicka nu på (OBS!) texten Test Kontor. Detta öppnar ett meddelandefönster.

6. Under fliken Aktiviteter klicka på alternativet NetMeeting.

Detta skickar en förfrågan till den andra användaren. När den har accepterat din inbjudan kommer ett litet fönster upp där du tillfrågas om du accepterar samtalet.



7. Klicka på Accept.
   I fönstret NetMeeting skall det nu finnas två rader med texten Markus Bylesjö. Längst ner i det vänstra hörnet står det också : ”In a call”



8. Klicka nu på ikonen längst ned till vänster i fönstret, dvs handen som håller ett ’program’.
   Detta öppnar ytterligare ett fönster där man får dela med sig av valfria program.

9. I det här fönstret gör följande:
   - Markera Mote Commands, klicka på knappen Share.
   - Klicka på knappen Allow Controll.
   - Klicka i rutan Automatically accept requests for controll

KLART! Nu kan den valda användaren använda det efterfrågade programmet.

Många situationer i vår vardag kan ge upphov till en känsla av otrygghet. Situationen är ofta sådan att möjlighet till kommunikation finns, men de känns onödigt krångliga och blir därför ett lite för stort steg att ta. Känslan av att inte vilja tränga sig på eller störa den andra personen blir också ofta starkare än viljan att dämpa sig egen oro.

Jag har i det här examensarbetet undersökt möjligheten att använda trådlösa sensornätverk för att skapa en enkel och snabb form av trygghetskommunikation.

Arbetet baseras kring motes, en liten dator som drar minimalt med ström och kommunicerar med omvärlden med hjälp av en radiosändare. Datorn övervakar en eller ett flertal sensorer, exempelvis sensorer för temperatur, ljus, ljud, position, acceleration och fuktighet. Tillsammans med hundratals eller tusentals motes bildas ett ad hoc-nätverk där noderna själva inordnar sig i nätverket, kommunicerar och skickar vidare information mellan varandra. Detta ger ett mycket kraftfullt nätverk som kan spridas över en stor yta och med mycket liten energiåtgång överföra lokalspecifik information till en basstation.

Genom att modifiera en befintlig applikation för insamling av sensorvärden skapas ett system där motes används för att sända meddelanden som sedan kan avläsas på en internetansluten dator. Dessutom möjliggörs en tvåvägskommunikation så att information även kan sändas från dator till mote.

Den här rapporten hittar du också på Internet:
http://www.certec.lth.se/trygghetskommunikation

Certec
Avdelningen för
rehabiliteringsteknik,
Inst för designvetenskaper,
Lunds tekniska högskola

Certec, LTH
Box 118
221 00 Lund

Sölvegatan 14 A
Lund

046 222 46 95
046 222 44 31
certec@certec.lth.se
http://www.certec.lth.se

Certec är en avdelning inom institutionen för designvetenskaper vid Lunds tekniska högskola.

Vår forskning och utbildning har en uttalad avsikt: att människor med funktions- nedsättningar skall få bättre förutsättningar genom en mer användvärd teknik, nya designkoncept och nya individnära former för lärande och sökande.

Drygt 20 människor arbetar på Certec. Den årliga omsättningen är cirka 12 miljoner kronor.

EXAMENSARBETE CERTEC, LTH NUMMER 6:2005
JUNI MÅNAD 2005

Markus Bylesjö

Trygghetskommunikation med motes

Security Communication using motes