CODEN:LUTEDX/(TEIE-3034)/1-46/(2014)

Industrial Electrical Engineering and Automation

Dataöverföring mellan noder och serverapplikation med hjälp av lågenergi - Bluetooth.

Mario Botic

Division of Industrial Electrical Engineering and Automation
Faculty of Engineering, Lund University

Dataöverföring mellan noder och serverapplikation med hjälp av lågenergi-Bluetooth.

LUNDS
UNIVERSITET
Lunds Tekniska Högskola

LTH Ingenjörshögskolan vid Campus Helsingborg
Industriell elektroteknik / IEA

Examensarbete:
Mario Botic

LTH Ingenjörshögskolan vid Campus Helsingborg
Lunds universitet
Box 882
251 08 Helsingborg

LTH School of Engineering
Lund University
Box 882
SE-251 08 Helsingborg
Sweden

Tryckt i Sverige
Media-Tryck
Biblioteksdirektionen
Lunds universitet
Lund <2006>

Sammanfattning

Examensarbetets syfte är utvecklingen av ett system, vilket erbjuder dataöverföring mellan trådlösa noder och en serverapplikation. Arbetet utförs på Subvision AB.

Hårdvarukomponenter vilka ingår i systemet väljs, konfigureras och deras applikationer utvecklas. Vidare utvecklas en serverapplikation. Detta sker i enlighet med förutbestämda krav, vilka sätts på systemet.

Resultatet ger en inblick i möjligheterna vid utvecklingen av ett trådlöst dataöverföringssystem.

Nyckelord

Dataöverföring, lågenergi-Bluetooth, radiomodem, serverapplikation, CC2540, MC55i.

Abstract

The purpose of this project is the development of a system, which provides transmission of data between wireless sensor nodes and a server application. The project is performed at the company Subvision AB.

Hardware modules which are included in the system are selected, configured and their software applications are developed. Furthermore, a server application is developed. This is done according to certain predetermined demands, which are put on the system.

The result gives insight of the possibilities, when developing a wireless data transmission system.

Keywords

Data, transmission, BLE, modem, server, application, CC2540, MC55i.

Förord

Rapporten beskriver mitt examensarbete, vilket har utförts på Subvision AB. Arbetet utgör en del av min examination inom kandidatprogrammet elektroteknik med automationsteknik vid Lunds Tekniska Högskola, Campus Helsingborg. Jag vill tacka min handledare Olle Kröling, VD Subvision AB, för möjligheten och alla goda råd. Jag vill också rikta ett speciellt tack till min syster Ivana Krajinovic, vars uppmuntringar och stöd har betytt mer än vad ord kan beskriva.

Helsingborg, November 2013.

Mario Botic

Innehållsförteckning

1 Inledning

1.1 Bakgrund

Subvision AB är ett företag som arbetar med att erbjuda konsultering vid skapandet av diverse undervattensteknologier. Företaget grundades år 1993 och har en lång erfarenhet av utvecklingsprojekt inom den offentliga och militära sektorn.

Projektet, vilket utgör examensarbetet, utförs på Subvision AB. Det ger en inblick i möjligheterna, vilka lågenergi-Bluetooth erbjuder, vid utvecklingen av ett system för dataöverföring.

1.2 Syfte och mål

Projektets syfte är konstruktionen av ett system, vilket möjliggör dataöverföring mellan en, eller flera, sensor-noder och en serverapplikation på internet.

1.3 Problemformulering

Systemet ska uppfylla följande krav:

1. Åtminstone en nod ska vara utrustad med ett radiomodem. Den här noden tillhandahåller uppkopplingen till internet.

2. Serverapplikationen ska baseras på C++ och använda utvecklingsramverket QT. Serverapplikationen ska köras på en PC, vilken använder operativsystemet Windows och är uppkopplad till internet.

3. Det ska vara möjligt att skicka kommandon och data till alla noder från serverapplikationen. Det ska också vara möjligt, för serverapplikationen, att ta emot data från alla noder.

4. Förutom möjligheten av lagring och hantering av den data som är ämnat för den, skall varje nod kunna lagra alla andra noders data.

5. En grundläggande konfiguration ska implementeras (Se appendix 13.1).

2 Metod

Nedan beskrivs varje steg av projektet. Vidare beskrivs och motiveras de metodval, vilka har gjorts för varje steg.

2.1 Val av komponenter

2.1.1 Noder

Texas Instruments CC2540-utvecklarpaket innehåller två moduler, vilka erbjuder dataöverföring via lågenergi-Bluetooth. Modulerna väljs för att agera som systemets noder. Detta görs för att hålla systemet energieffektivt.

2.1.2 Modem

Cinterions MC55i-GSM-Modem används för uppkopplingen mot serverapplikationen. Modemet väljs för att det tillhandahåller ett enkelt gränssnitt för konfiguration och dataöverföring.

2.2 Informationssamling

Metoden, för informationssamlingen, kan mest liknas vid den kvalitativa. Detta för att majoriteten av källorna, till informationssamlingen, utgörs av handböcker, datablad, interaktiva guider och källkod. Informationen, vilken erhålls, är därmed koncis och pragmatisk.

2.3 Testning av funktionalitet

Innan utvecklingen påbörjas, utförs testning av hårdvarumodulernas funktionalitet. Detta tillför en praktisk bekantskap med hårdvarumodulerna. Vidare tillför testningen ytterligare insikt, vilken kompletterar den förståelse, som erhålls vid informationssamlingen.

2.4 Utveckling

Utvecklingen av systemet består av tre delar. Varje del beskrivs nedan.

2.4.1 Mjukvaruutveckling till CC2540-noderna

Applikationer utvecklas, till CC2540-noderna, så att de bidrar till att uppfylla de krav, vilka sätts på systemet.

2.4.2 Gränssnitt mellan central-nod och modem

Gränssnittet mellan MC55i-modemet och CC2540-central-noden möjliggörs med hjälp av seriell kommunikation. Via modemet möjliggörs dataöverföring mellan central-nod och serverapplikation.

2.4.3 Utveckling av serverapplikation

En serverapplikation utvecklas efter de angivna specifikationerna i problemformuleringen.

2.5 Källkritik

2.5.1 CC2540-utvecklarpaket

Källorna som används är följande:

1. Texas Instruments CC2540/41 Bluetooh Low Energy Software Developer's Guide v1.3
2. Bluetooth Low Energy CC2540/41 Mine Development Kit User's Guide
3. OS Abstraction Layer Application Programming Interface
4. HAL Drivers Application Programming Interface
5. Texas Instruments CC2540 Bluetooth Low Energy API Guide

Ovanstående är dokumentationer, som har tagits fram, av paketets utvecklare, för att skapa en förståelse av modulernas funktionalitet. Källorna är därmed ytterst pålitliga.

2.5.2 MC55i-modem

Följande källor används:

1. MC55i AT Command Set
2. MC55i Terminal Hardware Interface Description
3. Gemalto MC55iT Datasheet

Ovanstående är dokumentationer, vilka har tagits fram av modemets utvecklare, för att skapa en förståelse av modulens funktionalitet. Källorna är därmed ytterst pålitliga.

2.5.3 Qt

Vid utvecklingen, av serverapplikationen, används Qt Assistant. Detta är ett interaktivt verktyg, vilket låter användaren att snabbt hitta klasser och funktioner. Det inkluderas vid installationen av Qt. Källan är ytterst pålitlig.

Vidare används följande:

Foundations of Qt Development av Johan Thelin (ISBN-13: 978-1-59059-831-3).

Källan är skriven i utbildningssyfte och är väldigt pålitlig.

2.5.4 Andra källor

Följande källor används vid beskrivningen av teknologier och system, samt vid förklaringen av terminologier:

Mobile Internet av Apostolis K. Salkintzis (ISBN: 978-0-203-49998-6).

Newnes PC Troubleshooting Pocket Book av Howard Anderson och Mike Tooley (ISBN: 978-0-7506-5988-8).

Data Communications and Networking av Behrouz A. Forouzan (ISBN 007-125442-0)

Böckerna är skrivna i utbildningssyfte och anses vara väldigt pålitliga.

3 CC2540-noder

Figur 3.1: Illustrerar Texas Instruments CC2540 Keyfob (vänster) och Dongle (höger) (se [1] sidan 4).

I systemet används Texas Instruments (TI) CC2540-utvecklarpaket. Paketet innehåller två moduler, vilka kallas Keyfob och Dongle. Båda dessa grundas på CC2540-kretskortet, vilket också är utvecklat av TI.

CC2540 erbjuder Bluetooth-lågenergi (BLE) överföring av datapaket. Detta möjliggör dataöverföring mellan Keyfob och Dongle.

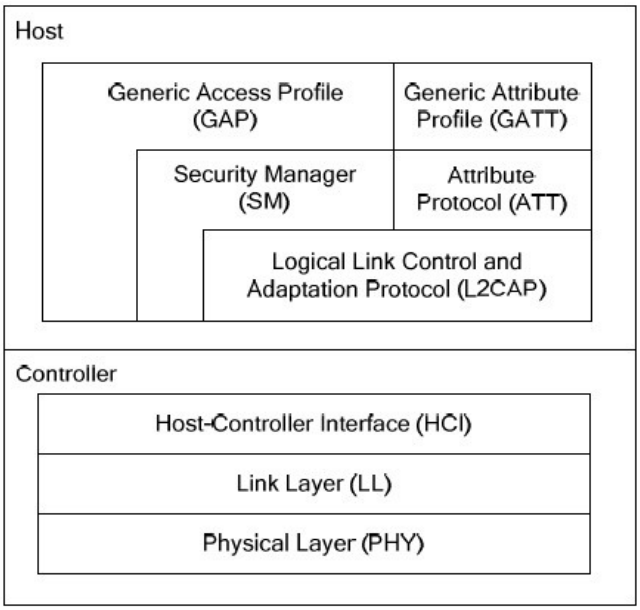
Keyfoben är utvecklad för att anta rollen av en perifer-nod. Dongle agerar som en central-nod (se [1] sidan 4). För enkelhetens skull refereras Keyfoben som perifer-nod och Dongle som central-nod.

Utvecklingen av CC2540-nodernas applikationer grundar sig på fyra sektioner. Dessa är BLE-protokollstack, abstraktionslagret för operativsystem (OSAL), abstraktionslagret för hårdvara (HAL) och profiler. Nedan följer en beskrivning av varje sektion.

3.1 BLE-protokollstack

BLE-protokollet består av ett antal hårdvarulager, mjukvarulager och underprotokoll. Dessa tillsammans utgör BLE-protokollstacken, vilken ingår i CC2540.

Nedan följer en illustration på dess uppbyggnad (se [2] sidan 5):



Figur 3.2: Illustration av protokollstacken och dess lager.

3.1.1 Beskrivning av stackens lager

För att ge en förståelse av BLE-systemets funktion, följer nedan en beskrivning av lagren, vilka ingår i stacken (se [2] sidan 6).

Det fysiska lagret (PHY) är en radio, vilken använder sig av ett frekvensband på 2.4 GHz. Radion använder frekvenshoppning. Detta innebär att två noder, under en anslutning, utbyter paket på ett antal olika frekvenskanaler.

Länk-lagret (LL) kontrollerar en nods tillstånd. En nod befinner sig alltid i ett av följande tillstånd: *standby*, *annonsering*, *avlyssning*, *initiering* och *ansluten*. En annonserande nod skickar ut data vilket tillåter en lyssnande nod att uppmärksamma den. Den lyssnande noden kan skicka en anslutningsbegäran till den annonserande noden och går därmed över till initieringstillståndet. Svarar den annonserande noden med ett godkännande, ändras båda nodernas tillstånd till *ansluten*. I det anslutna tillståndet tilldelas noderna två roller. Den nod, som begärt anslutningen, tilldelas alltid rollen master, medan den godkännande noden tilldelas rollen slav.

HCI-lagret tillhandahåller ett gränssnitt mellan värd- och kontroller sektionerna.

L2CAP-lagret tillhandahåller de övre lagren med inkapsling av data.

Säkerhetshanteraren (SM) är ett lager som tillhandahåller funktioner för säker anslutning och utbyte av data.

GAP-lagret hanterar anslutningsrelaterade tjänster och initieringen av säkerhetsfunktioner. Applikationslagret samverkar direkt med GAP-lagret genom diverse funktionsanrop.

Attributprotokollet (ATT) styr huruvida en nod får dela sina attribut med en annan nod. Attribut är data som läses och skrivs mellan noderna.

Attributprotokollet nås via GATT-lagret. Vid en anslutning mellan två noder, hanteras all datakommunikation mellan dessa genom GATT-lagrets procedurer. Därför samverkar både applikationslagret och profilerna med GATT-lagret.

3.1.2 Fördjupning av GAP-lagret

Den generiska accessprofilen (GAP) hanterar nodens procedurer. Dessa procedurer är upptäckande av andra noder, länk-etablering, länk-avbrott, konfiguration och säkerhetshantering. Den hanterar också nodens tillgångsläge. De relevanta lägena, som en nod kan befinna sig i, är följande (se [2] sidan 12):
Perifer – sänder ut annonseringar och kan svara på anslutningsbegäran från en central. När en anslutning utförts, tilldelas perifer:n slavrollen i länk-lagret.
Central – lyssnar efter annonseringar och sänder anslutningsbegäran, när lämplig perifer upptäcks. Vid anslutning till en perifer, tilldelas centralen master-rollen i länk-lagret. Centralen klarar av att koppla upp sig mot tre olika perifer samtidigt.

Perifer-nodens annonseringar innehåller specifik data. Detta data meddelar, en lyssnande central-nod, att perifer-noden är anslutningsbar och innehåller dess adress. Vid mottagningen av en annonsering, kan en central-nod välja att sända en anslutningsbegäran till den annonserande perifer-noden.

GAP-lagret erbjuder fler funktionaliteter men dessa ligger bortom omfattningen av den här rapporten.

3.1.3 Fördjupning av GATT-lagret

GATT-lagret används, av applikationen, vid datakommunikation mellan två noder (se [2] sidorna 14-15). När två noder är anslutna, till varandra, antar ena noden rollen av GATT-klient och den andra noden antar rollen av GATT-server. GATT-klienten är den nod, vilken läser och skriver data från, respektive till, GATT-servern. GATT-servern är noden, vilken innehåller den data, som GATT-

klienten läser från och skriver till. I projektets system, antar central-noden alltid rollen av GATT-klient och perifer-noden antar rollen av GATT-server.

Datavärden, deras inställningar och beskrivningsdata kallas attribut. I GATT-servern lagras attributen i en tabell, vilken agerar som en databas. Förutom själva värdet, associeras tre egenskaper med varje attribut:
Adress – Visar var, i attributtabellen, ett attribut ligger. Varje attribut har en enskild adress.
Typ – Indikerar vad attributvärdet representerar och benämns som UUID (engelska: Universal Unique Identifier).
Behörighet – Bestämmer om, och på vilket sätt, en GATT-klient får ha tillgång till till attributvärdet hos en GATT-server. Ett attribut kan ha läs- och skrivbehörighet, det vill säga GATT-klienten får både läsa och ändra attributvärdet. Attributet kan endast ha läsbehörighet, vilket innebär att klienten får läsa av attributvärdet men inte ändra det. Slutligen kan attributet ha skrivbehörighet och GATT-klienten får endast ändra värdet men inte läsa från det.

GATT-lagret erbjuder en rad underprocedurer för kommunikationen mellan GATT-klient och server. De relevanta procedurerna är följande :
Läsning av datavärden – GATT-klienten begär läsning av ett attribut, vid en specifik adress, hos en GATT-server. Servern svarar med det begärda värdet. Detta förutsatt att attributet har läsningstillstånd.
Skrivning av datavärden – GATT-klienten begär att ändra ett attribut, vid en specifik adress, i attributtabellen. GATT-servern meddelar klienten om skrivningen lyckades eller inte.

Dessa procedurer initieras med anropet av följande API-funktioner (se [3]):
GATT_ReadCharValue – Anropas i GATT-klientens applikation och initierar läsning av ett attribut.
GATT_WriteCharValue – Anropas i GATT-klientens applikation och initierar läsning av ett attribut.

Eftersom central-noden alltid antar rollen av GATT-klient, anropas API-funktionerna i central-nodens applikation.

API-funktionen **GATT_WriteCharValue** används vid skrivningar av data till perifer-noden. En skrivning kan innehålla data, från serverapplikationen, vilken central-noden sänder vidare till perifer-noden.

Skrivningar kan också vara läsningsbegäran, av data, från perifer-noden. Exempelvis kan en läsningsbegäran skrivas för att värdet av antalet

knapptryckningar, som har gjorts på perifer-noden, ska erhållas. Efter att en lyckad läsningsbegäran skrivs, används **GATT_ReadCharValue**, för att läsa den data som har begärts.

Om en skrivning misslyckas, upprepas den med samma data, som i föregående försök.

3.2 Profiler

Vid mjukvaruutvecklingen, till CC2540-noderna, används profiler för att noderna ska kunna utföra funktionaliteter såsom länketablering och dataöverföring. Det är dessa profiler som utgör GAP- och GATT-lagren, i en nod. Nedan nämns de profiler som är relevanta. Dessa ger också djupare insikt om hur anslutning, och dataöverföring, sker.

3.2.1 GAP-profil för perifer-rollen

Profilen tillhandahåller perifer-noden med möjligheterna att sända ut annonseringar och acceptera anslutningsbegäran från central-noden (se [2] sidan 18). Det meddelar också perifer-nodens applikation om tillståndsändringar.

Profilen innehåller en mängd API-funktioner och parametrar.

Följande API-funktion är relevant:
GAPRole_SetParameter – Funktionen används för att sätta GAP-parametrar.

Följande parameter är relevant:
GAPROLE_ADVERT_ENABLED – Parametern används för att sätta igång, eller stänga av, annonseringar från perifer-noden.

Utöver detta tillhandahåller profilen en funktionspekare, vilken används för att anropa en funktion i applikationen, vid förändringen i perifer-nodens GAP-tillstånd (se [2] sid 18). En förändring är exempelvis övergången från tillståndet *annonserande* till tillståndet *ansluten*.

I perifer-nodens applikation anropas funktionen **GAPRole_SetParameter**, med parametern **GAPROLE_ADVERT_ENABLED** och booleska **TRUE**, för att initiera annonsering. Första anropet av funktionen sker efter att alla initieringsrutiner, av mjukvaran har slutförts. På så sätt börjar perifer-noden annonsera efter uppstart. Om inga anslutningar har gjorts, efter att annonseringen slutförs, anropas funktionen igen med samma parametrar. Detta gör att perifer-noden annonserar tills dess att central-noden gör en

anslutningsbegäran. Detta försäkrar om att central-noden kan göra en anslutning, till perifer-noden, vid behov.

Profilen innehåller fler funktioner och parametrar, som ligger bortom omfattningen av den här rapporten.

3.2.2 GAP-profil för central-rollen

Profilen ger central-noden möjligheten att initiera sökning av annonserande perifer-noder. Vidare möjliggör profilen länketablering och länkavbrott till en perifer-nod (se [2] sidan 19).

Profilen innehåller en mängd API-funktioner, Vilka kan anropas från central-nodens applikation. Följande API-funktioner är relevanta:
GAPCentralRole_StartDiscovery – Funktionen anropas för att initiera sökning av annonserande perifer-noder.
GAPCentralRole_EstablishLink – Om en annonserande perifer-nod hittas, anropas funktionen för att initiera länketableringen till perifer-noden.
GAPCentralRole_TerminateLink – Funktionen anropas för att avbryta en etablerad länk till perifer-noden.

När central-noden har mottagit serverapplikationens data, initieras en process för anslutning till perifer-noden. Inom processen anropas **GAPCentralRole_StartDiscovery**, för att central-noden ska upptäcka, den ständigt annonserande, perifer-noden. När perifer-noden upptäcks, anropas **GAPCentralRole_EstablishLink** för att initiera länketableringen till perifer-noden.

Därefter initieras processerna för dataöverföring till, och från, perifer-noden. När dessa har avslutats, anropas **GAPCentralRole_TerminateLink** för att bryta länken till perifer-noden.

Profilen har fler funktioner, som ligger bortom omfattningen av den här rapporten.

3.2.3 GATT-profil

GATT-profiler används för hantering och lagring av data inom en GATT-server (se [2] sidan 21). I GATT-profilen lagras attributtabellen. BLE-protokollstacken tillhandahåller ett exempel av en GATT-profilimplementation. Detta används som grund vid utvecklingen av GATT-profilen, vilken används i systemets

perifer-nod. Profilexemplet heter **simpleGATTProfile**.

Profilexemplet innehåller följande relevanta API-funktioner:
SimpleProfile_GetParameter – Applikationen anropar funktionen för att hämta värdet från ett givet attribut i attributtabellen.
SimpleProfile_SetParameter – Applikationen anropar funktionen för att sätta ett värde till ett givet attribut i attributtabellen.
simpleProfile_WriteAttrCB – Funktionen registreras med GATT-servern och anropas varje gång GATT-klienten försöker att skriva ett värde till ett attribut. Innan skrivningen görs kontrollerar funktionen om värdet är giltigt. Om detta är fallet, skrivs värdet till attributet.
simpleProfile_ReadAttrCB – Funktionen registreras med GATT-servern och anropas varje gång GATT-klienten försöker att läsa värdet från ett attribut. Funktionen kontrollerar att attributadressen, som GATT-klienten angett, finns i attributtabellen. Om adressen är riktig låts, i funktionen, en pekare att peka på attributvärdet. Denna pekare används sedan för att lägga in attributvärdet i svaret, vilket GATT-servern skickar till GATT-klienten.
SimpleProfile_RegisterAppCBs – Funktionen tillåter registreringen av en applikationsfunktion, som kommer att anropas varje gång GATT-klienten gör en lyckad skrivning till GATT-servern.

Inom profilen, har fem attribut registrerats i attributtabellen. Detta är i demonstrationssyfte.

Profilen används i samband med perifer-nodens applikation. Detta för att perifer-noden alltid antar rollen som GATT-server i systemet.

Inom **simpleGATTprofile** modifieras attributtabellen så att den innehåller endast ett attribut. Attributet ges skriv- och läsbehörighet. Det är detta attribut, vilket central-noden skriver till och läser ifrån. Vidare modifieras funktionerna **simpleProfile_WriteAttrCB** och **simpleProfile_ReadAttrCB**, så skrivning och läsning, till det nya attributet, möjliggörs.

I applikationen definieras en funktion som registreras med **SimpleProfile_RegisterAppCBs**. Funktionens syfte är att anropa **SimpleProfile_GetParameter** när central-noden gör en lyckad skrivning till attributet. Det mottagna datavärdet tolkas och lagras i applikationen.

Om skrivningen är en läsningsbegäran av data, vilken lagras i applikationen, anropas **SimpleProfile_SetParameter**. Vid anropet sätts det begärda datavärdet som en parameter. Funktionen placerar det begärda datavärdet i attributet. Central-noden läser av attributet för att erhålla datavärdet.

Profilen har fler funktioner, som ligger bortom omfattningen av den här rapporten.

3.3 OSAL

OSAL är en kontrollslinga, vilken tillåter alla mjukvarulager, det vill säga applikationslager, BLE-stacken och profilerna att sätta upp exekveringen av händelser (se [2] sidan 9).

3.3.1 Initiering

Alla lager har varsin initieringsrutin, vilken anropas när OSAL initieras. Varje lager tilldelas identifierare (ID), vid anropet av dess initieringsrutin. Dessa ID används för att OSAL ska kunna skilja på de olika lagren (se [2] sidan 10).

3.3.2 Händelsehantering

Händelserna, för varje mjukvarulager, implementeras som en 16-bitars variabel. Varje bit svarar mot en händelse (se [2] sidan 10).

När en händelse sätts, identifierar OSAL vilket lager händelsen gäller. OSAL anropar därefter händelsehanteringsrutinen, vilken är definierad för att hantera det lagrets händelser. Hanteringsrutinen initierar de relevanta processerna, baserat på värdet av händelsevariabeln. Varje mjukvarulager har möjligheten att initiera en OSAL-händelse för alla andra lager, samt sig själv.

3.3.3 Prioritering

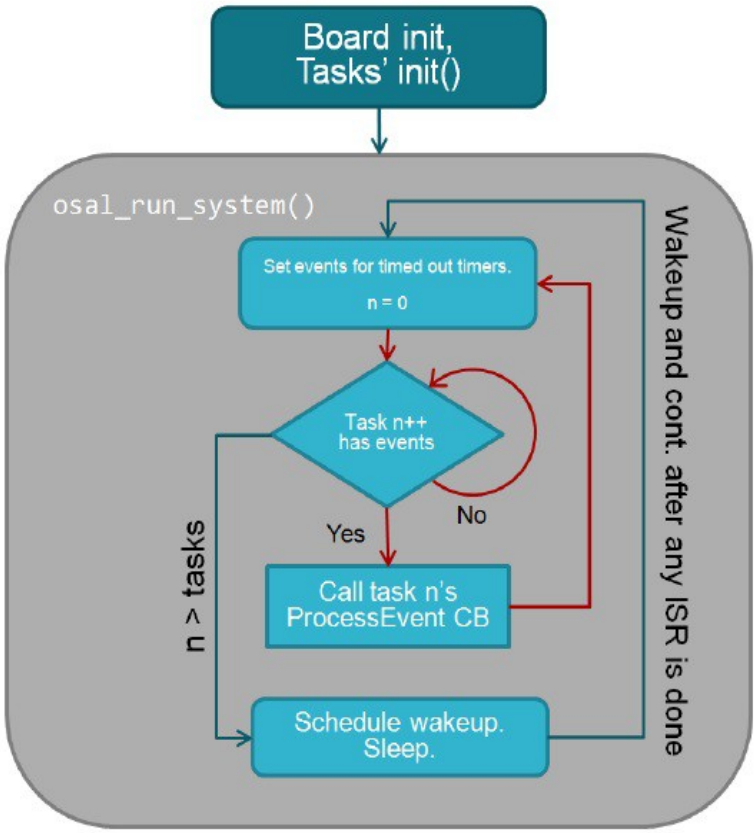
Förutom att identifiera mjukvarulagren, används ID-värdet för att bestämma prioriteten för varje lagers händelser. Ju högre ID-värde ett lager har, desto lägre prioriteras dess händelser. Protokollstackens lager måste ha högst prioritet och därmed lägst ID-värde. Detta för att försäkra proper funktionalitet. Applikationslagret har i regel alltid det högsta ID-värdet och därmed lägst prioritet.

3.3.4 Meddelanden

Inom OSAL finns ett system för kommunikation mellan mjukvarulagren. (SOFTWARE DEV 3.1.4) Detta görs med sändningen och mottagningen av meddelanden. Meddelandeöverföring registreras som en händelse. När ett lager

får ett meddelande, anropar OSAL det lagrets händelsehanteringsrutin. Inom händelsehanteringsrutinen anropas den funktion, vilken behandlar meddelandet. Alla lager tillhandahåller en unik funktion för att behandla meddelanden.

Flödesdiagrammet nedan illustrerar OSAL:s process (se [2] sidan 9):



Figur 3.3: Illustrerar kontrollslingan, vilken utgör OSAL. Läsaren bör notera att ett mjukvarulager är en uppgift (Engelska: Task) sett från OSAL.

3.3.5 API-funktion

OSAL tillhandahåller följande API-funktion:
osal_set_event – Funktionen används för att sätta en händelse till ett givet mjukvarulager (se [4] sidan 5). Funktionens parametrar är ID-värdet av mjukvarulagret, vilket ska hantera händelsen, och händelsens 16-bitarsvärde.

Varje process, i nodernas applikationer, initieras som en OSAL-händelse med **osal_set_event**. Exempel på processer, i central-noden, är: anslutning till perifer-nod, dataöverföring till perifer-nod och datahämtning från perifer-nod.

Genom att processerna schemaläggs med OSAL, undviks kollisioner mellan aktiva processer i stacken.

Initiationsfunktionerna, för varje process, lagras i händelsehanteringsrutinen och tilldelas ett 16-bitarsvärde var.

3.3.6 Exempel av processhantering

Följande är ett exempel vilket visar vad som sker när en process, vilken utgörs av en skrivning till perifer-noden, initieras (se [2] sidorna 16-17):

1. I central-nodens applikation anropas **osal_set_event** med applikationens eget ID-värde och initiationsfunktionens 16-bitarsvärde som inparameter.
2. Initiationsfunktionen anropar **GATT_WriteCharValue**.
3. GATT-lagret hanterar begäran och svarar med **SUCCESS**.
4. Stacken gör skrivningen till perifer-noden. Perifer-noden sänder ett svar, vilket innehåller data, som meddelar om skrivningen lyckas.
5. Stacken sänder detta svar till GATT-lagret, vilket tolkar meddelandet och tilldelar en lämplig pekare till det.
6. Applikationen mottar ett OSAL-meddelande, vilket indikerat att GATT-lagret har data som är ämnat för applikationen.
7. Applikationen behandlar OSAL-meddelandet i dess händelsehanteringsrutin. I händelsehanteringsrutinen anropas funktionen, vilken använder den ovannämnda GATT-pekaren för att erhålla svaret från perifer-noden.

3.4 HAL

Abstraktionslagret för hårdvara (HAL) används för att utveckla eller lägga till hårdvara. Detta görs utan att förändringar av protokollstacken, eller applikationens källkod, behövs. Det tillhandahåller ett abstraktionsgränssnitt mellan hårdvara och applikation (se [2] sidan 11). Detta gränssnitt utgörs av API-funktioner för hanteringen av LED-lampor, knappar, UART-kommunikation och så vidare.

3.4.1 API-funktioner

Följande API-funktioner är relevanta:
HalUARTRead – Läser data från UART-porten (se [5] sidan 21).
HalUARTWrite – Skriver data till UART-porten (se [5] sidan 22).
HalLedSet – Används för att styra en given LED-lampa (se [5] sidan 8).
Central-nodens UART-port är sammankopplad med MC55i-modemets seriella

port. I central-nodens applikation, anropas funktionen **HalUARTRead** för att läsa data från UART-porten och därmed modemet. Funktionen **HalUARTWrite**, anropas för att skriva data till UART-porten och därmed modemet. Funktionerna möjliggör alltså att kommunikationen, mellan central-noden och modemet, kan styras från applikationen.

Vid varje länketablering, mellan central-nod och perifer-nod, anropas **HalLEDSet** för att tända en lampa på central-noden. Detta för att indikera att länken är etablerad. När ett avbrott av länken sker, anropas funktionen igen för att släcka lampan.

HAL innehåller fler API-funktioner men dessa är bortom omfattningen av den här rapporten.

4 MC55i-modemet

Figur 4.1: Illustrerar MC55i-modemet (se [6] sidan 1).

GSM-modemet är utvecklat av Cinterion och tillhandahåller GPRS, vilket möjliggör överföringen av datapaket. TCP/IP-kommunikation används för sändning, och mottagning, av datapaketen (se [6] sidan 2).

Modemet har en RS-232 port, vilken tillåter seriell kommunikation med andra moduler (se [7] sidan 18).

4.1 Konfiguration och användning av TCP/IP-kommunikation

Interaktionen, med modemet, görs med hjälp av AT-kommandon (se AT-kommandon under sektion 7). Nedan följer en överblick av hur dessa kommandon används, för konfigurera och använda, modemets TCP/IP-

kommunikation (se [8] sidorna 223-224).

4.1.1 Konfiguration

Först skapas en uppkopplingsprofil, med användning av kommandot **AT^SICS**. Uppkopplingsprofilen bestämmer typen av uppkoppling till internettjänsten. Uppkopplingsprofilen kan vara av typen CSD eller GPRS (Se kort om GPRS)

En tjänstprofil skapas med användning av kommandot **AT^SISS**, vilket specificerar typen av internettjänst, som används av modemet. Typen av internettjänst kan vara: TCP-ändpunkt, HTTP, FTP, eller någon av e-post-tjänsterna POP3 eller SMTP.

4.1.2 Öppning och stängning av internetsessioner

Efter att profilerna skapas, används kommandot **AT^SISO** för att öppna en internetsession. Detta möjliggör initieringen av sändningen, och mottagningen, av datapaket. För att stänga sessionen, används **AT^SISC**.

4.1.3 Dataöverföring

För att begära uppladdning av data, används kommandot **AT^SISW**, med datalängden som en inparameter. När modemet svarar med **^SISW**, kan uppladdningen påbörjas

För att begära nerladdning av data, används kommandot **AT^SISR**. Om en server begär att skicka data till modemet, meddelar modemet detta genom att skicka **^SISR**.

4.2 Användning

Modemets RS-232 port kopplas till central-nodens UART-port. Modemet möjliggör dataöverföring mellan serverapplikationen och central-noden. Nedan beskrivs hur AT-kommandona används, i central-nodens applikation, för att utföra de ovanstående momenten.

4.2.1 Konfiguration av TCP/IP-kommunikation

I central-nodens applikation lagras alla AT-kommandon som strängar. Dessa strängar skrivs till central-nodens UART-port och sänds därmed till modemet.

Strängarna, med AT-kommandona **AT^SICS** och **AT^SISS**, används för att skapa uppkopplingsprofilen och tjänstprofilen.

Uppkopplingsprofilen sätts till att vara av typen GPRS. Internettjänsten sätts som en TCP-ändpunkt.

4.2.2 Öppning och stängning av internetsessioner

För att öppna en internetsession, skrivs strängen, som innehåller kommandot **AT^SISO**, till UART-porten och sänds vidare till modemet.

För att stänga en session, skrivs strängen, som innehåller kommandot **AT^SISC**, till UART-porten och därmed vidare till modemet.

4.2.3 Mottagning av data från serverapplikationen

När serverapplikationen vill skicka data, meddelas central-noden genom att modemet skickar **^SISR** till central-nodens UART-port.

När detta sker skrivs strängen med AT-kommandot **AT^SISR,** till porten. Kommandot får modemet att initiera nerladdningen av det data, som serverapplikationen vill skicka.

Modemet skickar den data, vilken laddas ner från serverapplikationen, till UART-porten. Detta data läses och lagras i applikationen.

4.2.4 Sändning av data till serverapplikationen

För att initiera uppladdningen av data, skrivs strängen med AT-kommandot **AT^SISW** till UART-porten. Modemet skickar en bekräftelse till porten, när uppladdningen kan initieras. Därefter initieras skrivningen, av det data som är ämnat för serverapplikationen, till UART-porten. Detta data kommer att, via modemet, skickas till serverapplikationen.

5 MAX3232-nivåkonverterare

Modulen MAX3232 tillhandahåller ett elektriskt gränssnitt mellan en asynkron kommunikationsregulator och en seriell port (se [9] sidan 2).

Den innehåller förstärkare och kretsar, vilka används för att förstärka signaler. På motsvarande sätt kan modulen användas för att dämpa signaler.

5.1 Användning

I systemet utgör central-nodens UART-port den ovannämnda asynkrona kommunikationsregulatorn och modemets RS-232 port är den seriella porten.

Modemets RS-232 port sänder ut signaler, vilka har en storlek på cirka 5.5-V (se [7] sidan 37). Dessa måste dämpas till 3.6-V innan de når ingången, vilken leder till central-nodens UART-port. MAX3232-modulen används mellan modemet och central-noden för att förse denna dämpning. På motsvarande sätt används MAX3232-modulen för att förstärka signalerna från central-noden innan de når modemets RS-232 port. Central-nodens utgående signaler förstärks från 3.6-V till 5.5-V.

6 Qt

Qt är ett utvecklingsramverk, som använder programmeringsspråket C++ som bas. Dess styrka är att det bidrar till att skapandet, sammankopplingen och utbytet av applikationskomponenter blir enkelt. Qt tillhandahåller kompilering och körning av applikationer på operativsystemen Windows, Linux, olika sorters Unix och Mac OS X (se [10] sidan 3).

6.1 Klasser och funktioner

QT förser klasser och funktioner, vilka möjliggör TCP-anslutningar och dataöverföringar.

6.1.1 Klassen QTcpServer

Klassen tillhandahåller en TCP-baserad server, som gör det möjligt att acceptera en TCP-anslutning, från ett modem.

Följande funktioner, inom klassen, är relevanta (se [11] sektion QtcpServer):
listen – Meddelar servern att lyssna efter inkommande TCP-anslutningar.
nextPendingConnection -- Funktionen anropas när en anslutning till servern begärs. Funktionen returnerar, den avvaktande, anslutningen som ett **QTcpSocket** objekt (se nedan).

6.1.2 Klassen QTcpSocket

Klassen tillhandahåller en ändpunkt vid en TCP-förbindelse. Den möjliggör en

applikation att göra TCP-anslutningar och dataöverföringar.

Följande funktioner, inom klassen, är relevanta (se [11] sektion QTcpSocket):
write – Funktionen initierar sändningen av data över en TCP-anslutning. Dess inparameter är en buffert innehållande den data som ska sändas.
readAll – Funktionen anropas vid mottagningen av data över en TCP-anslutning. Den returnerar en buffert med den data som har mottagits.

6.2 Användning

6.2.1 Anslutning till modemet

Anropet till funktionen **listen** görs efter uppstart av serverapplikationen. När modemet begär en anslutning, anropas **nextPendingConnection**, för att en TCP-ändpunkt skapas och förbindelsen upprätthålls.

6.2.2 Dataöverföring

För att initiera dataöverföringen, till modemet, anropas funktionen **write**.

När modemet gör en uppladdning, till servern, anropas funktionen **readAll**. Detta hämtar den data, som modemet har sänt.

7 Teknisk bakgrund

Bluetooth-lågenergisystem (BLE) – BLE-systemet utvecklades vid behovet av att tillgodose behovet av dataöverföring med låg effektförbrukning. Denna låga strömförbrukning åstadkoms genom att datapaketen, vilka överförs via BLE-systemet, görs väldigt små (se [2] sidan 5).

Universal Asynchronous Receiver/Transmitter (UART) – Är en hårdvarumodul, vilken tillhandahåller ett asynkront seriellt gränssnitt. Det erbjuder sändning och mottagning av data. När data sänds, från UART, delas varenda byte upp i bitar och sänds bitvis. Vid mottagning, sätts de mottagna bitarna ihop till bytes.

RS-232 – Är en standardiserad typ av seriell port, vilken används vid dataöverföringar. RS står för rekommenderad standard.

D-Sub – Är en kontakt, vilken används vid koppling till RS-232-porten.

AT-kommandon – Samling kommandon, vilka tillsammans utgör ett språk och används vid hanteringen av ett modems funktionaliteter.

GENERAL PACKET RADIO SERVICE (GPRS) – Är en tjänst, vilken möjliggör internetåtkomst för mobila GSM-kommunikationer. GPRS kan liknas vid ett IP-nätverk, vilket tillhandahåller uppkoppling till IP-terminaler. Det skapar en kommunikationskanal, vilken möjliggör överföringen av IP-paket mellan exempelvis ett GSM-modem och en ip-terminal (se [12] Sektion 3.1-3.2).

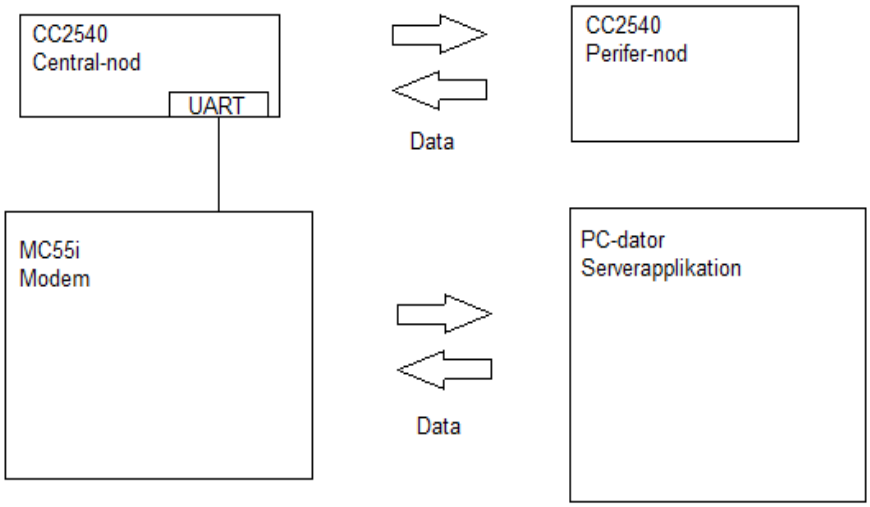
Transmission Control Protocol/Internet Protocol (TCP/IP) – Är ett protokoll, vilket består av olika lager och erbjuder dataöverföring över nätverkstjänster (se [13] sidan 42).

8 Resultat

Nedan beskrivs projektets resultat.

8.1 Konstruktion

Systemet är konstruerat för att uppfylla de ställda kraven i problemformuleringen och möjliggör därmed dataöverföring mellan serverapplikation och noder.



Figur 8.1: Illustrerar systemets hårdvarukomponenter och dataöverföringen mellan dessa.

Konstruktionen kan delas in i tre delar: serverapplikation, central-nod med modem och perifer-nod.

8.1.1 Central-nod och modem

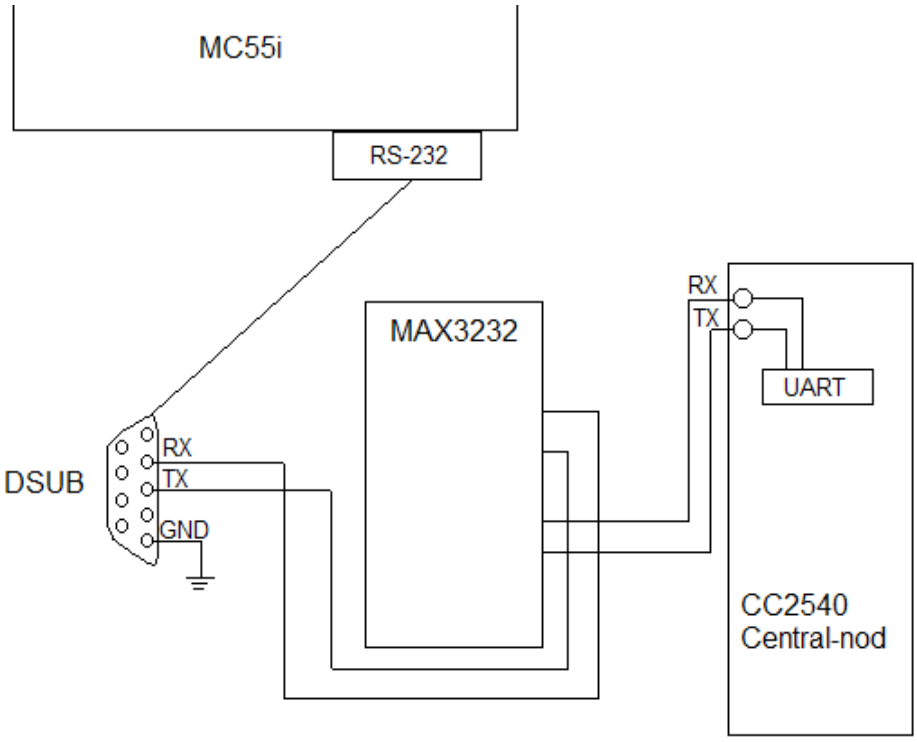
Central-noden är utvecklad att agera som en master och har möjligheten att initiera en anslutning till perifer-noden. Denna egenskap gör att central-noden väljs för att kopplas ihop med modemet.

Konfigurationen gör att dataöverföring till och från serverapplikationen, sker via central-noden och modemet. Detta innebär att data, vilken är ämnad till perifer-noden, överförs först till central-noden via modemet. Central-noden sänder därefter detta data vidare till perifer-noden via lågenergi-Bluetooth. På motsvarande sätt sänds data, som är ämnat till serverapplikationen, från perifer-noden till central-noden. Detta data sänds därefter från central-noden till serverapplikationen via modemet.

Hårdvarugränssnitt

För att signalerna, vilka utgörs av data, ska kunna överföras mellan central-noden och modemet, används MAX3232-modulen. MAX3232 fungerar som ett gränssnitt, vilket omvandlar central-nodens och modemets signaler till acceptabla nivåer.

Figuren nedan illustrerar sammankopplingen mellan central-nod, MAX3232-modulen och modemet.



Figur 8.2: Illustrerar sammankopplingen mellan central-nod, MAX3232-modul och modem.

Central-nodens pin, för utgående datasignaler, markeras TX. Dessa utgående signaler förstärks i MAX3232-modulen och skickas till D-SUB-kontaktens RX-pin. D-SUB-kontakten är kopplad till modemets RS-232-kontakt. Datasignalerna, från modemet, går genom D-SUB-kontaktens TX-pin och dämpas i MAX3232-modulen. Den dämpade datasignalen sänds till central-nodens RX-pin.

8.1.2 Perifer-nod

Perifer-nodens funktion är att göra dataöverföringar med central-noden. All data, vilken mottas av perifer-noden, lagras i dess applikation. Central-noden kan hämta data från perifer-noden. Exempel på detta data är antalet knapptryckningar, som görs på perifer-noden.

8.1.3 Serverapplikation

Utvecklingen av serverapplikationen görs i programmeringsspråket C++, med hjälp av utvecklingsramverket Qt. Applikationen körs på en PC-dator som är uppkopplad till internet.

Applikationens funktionalitet innefattar sändning och mottagning av data till, respektive från, noderna. Dess syfte är att erbjuda en demonstration av möjligheten till dataöverföringen mellan serverapplikationen och noderna.

När en anslutning mellan serverapplikationen och modemet etableras, initieras överföringen av data från serverapplikationen till modemet. Detta data är ämnat till både central-nod och perifer-nod. Vid mottagning, sänds data vidare från modemet till central-noden. Efter att överföringen slutförts bryts anslutningen mellan modemet och serverapplikationen.

När data från noderna ska överföras till serverapplikationen, etableras en anslutning mellan modem och serverapplikation. Central-noden sänder därefter data till serverapplikationen via modemet. Efter att överföringen slutförs, bryts anslutningen mellan modemet och serverapplikationen.

8.2 Matriser och datahantering

I systemet sker datalagringen och dataöverföringen med hjälp av matriser. Dessa matriser är egentligen vektorer, vars element är dataunioner och utgör matrisens kolumner. Inom varje union deklareras datastrukturobjekt, som i sin tur innehåller flera olika datastrukturobjekt. Uppbyggnaden möjliggör lagring, av olika typer av variabelvärden, inom en matris. Vidare möjliggör det att matriserna kan delas upp i bytes vid dataöverföringar.

8.2.1 Uppbyggnad

Nedan beskrivs uppbyggnaden av en matris.

Inom en kolumn deklareras ett antal datastrukturer. Exempel på datastrukturer kan vara följande:

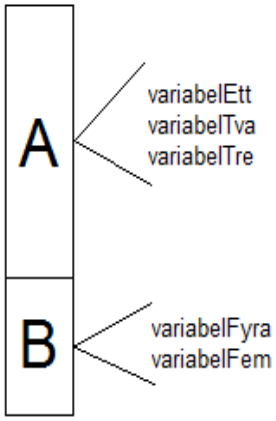
```
typedef struct{

  uint8 variabelEtt;
  char variabelTva;
  uint16 variabelTre;

}dataStrukturEtt;
```

```
typedef struct{

  uint16 variabelFyra;
  char variabelFem;

}dataStrukturTva;
```

Själva matriskolumnen är också en datastruktur, i vilken objekt av **dataStrukturEtt** och **dataStrukturTva** deklareras:

```
typedef struct{

  dataStrukturEtt A;
  dataStrukturTva B;

}kolumn;
```

Ett objekt, av typen **kolumn**, innehåller alltså objektet A, av typen **dataStrukturEtt**, och objektet B, av typen **dataStrukturTva**.



Figur 8.3: Illustrativ tolkning av ett objekt, som är av typen kolumn.

Deklarationen av **kolumn**-objektet sker inne i en union, tillsammans med deklarationen av en **uint8**-vektor. Vektorn är av **kolumn**-objektets storlek.

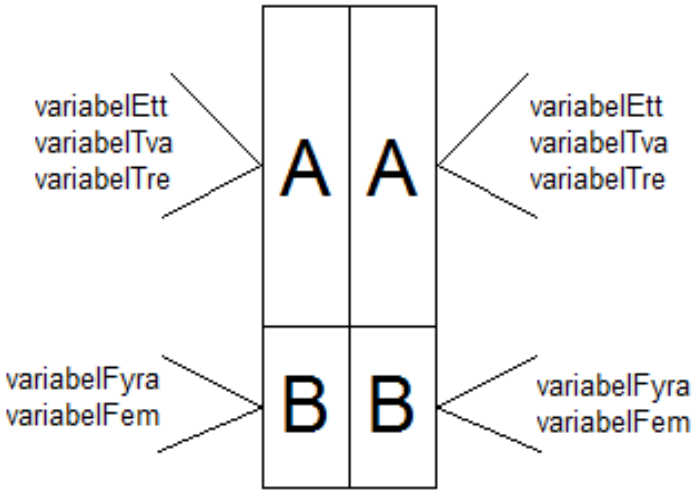
Exempel:

```
typedef union{

  kolumn matrisKolumn;
  uint8 kolumnVektor[sizeof(kolumn)];

}matris;
```

Unionen möjliggör att databytes i minnet, som allokerats för matrisKolumn, kan nås via kolumnVektor. Objekten matrisKolumn och kolumnVektor är alltså olika representationer av samma data.

Figur 8.4: Illustrativ tolkning av kolumnVektor, vars element består av de databytes, som utgör kolumn-objektet. Varje element innehåller en byte.

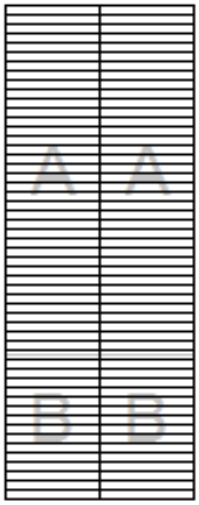
För att erhålla en matris, deklareras en vektor av **matris**-element. Nedan illustreras matrisen, som erhålls när deklarationen matris exempelMatris[2]; görs:



Figur 8.5: Illustrativ tolkning av matrisen exempelMatris, som består av två kolumner.

För att, exempelvis, erhålla värdet av variabelFyra, i den första kolumnen, skrivs: exempelMatris[0].matrisKolumn.B.variabelFyra;. Följande skrivs för att variabelTre, i den andra kolumnen, ska tilldelas värdet 15: exempelMatris[1].matrisKolumn.A.variabelTre=15;.

Matrisens vektor-representation illustreras på följande sätt:



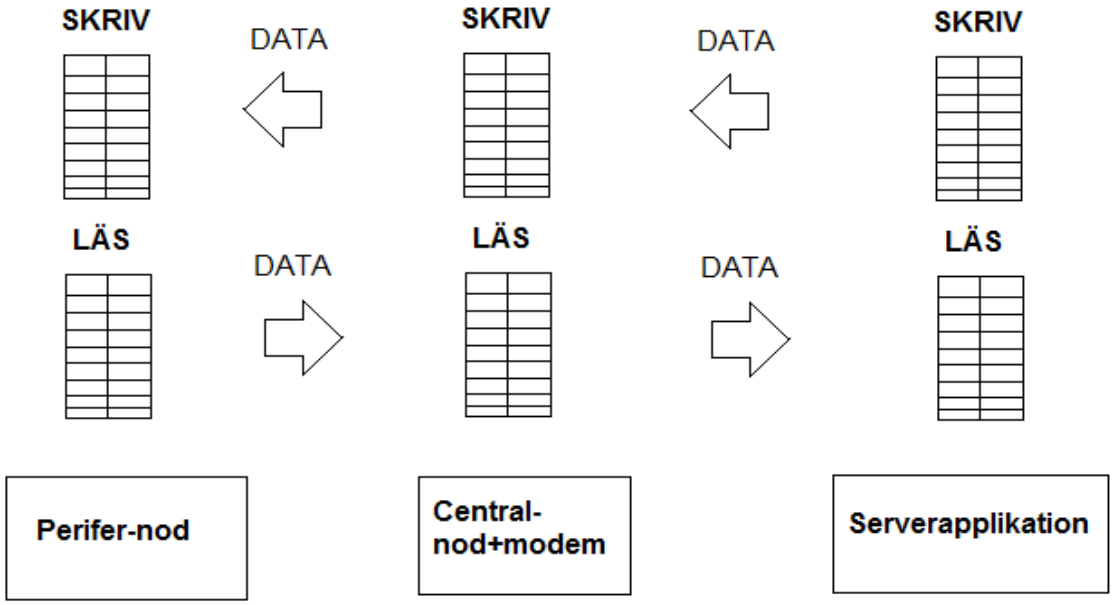
Figur 8.6: Illustrativ tolkning av vektor-representationen av matrisen exempelMatris.

Elementen i vektor-representationen är matrisens rader. Följande skrivs för att n:te byten, i första kolumnen, ska erhållas:

exempelMatris[0].kolumnVektor[n];.

8.2.2 Användning vid dataöverföringar

Serverapplikationen, central-nodens applikation och perifer-nodens applikation innehåller två matriser var. I varje matrispar används den ena matrisen för dataöverföringen från serverapplikationen. Den gemensamma benämningen för dessa är skrivningsmatris. Den andra matrisen, i varje matrispar, används vid dataöverföringen från noderna till serverapplikationen. Den gemensamma benämningen är läsningsmatris. Alla skrivningsmatriser har likadana kolumner och alla läsningsmatriser har likadana kolumner.



Figur 8.7: Figuren illustrerar matrisernas roll i dataöverföringen.

Nedan följer exempel, som beskriver hur skrivningen och läsningen, mellan matriserna, sker. I exemplen antas skrivnings- och läsningsmatriserna vara av typen **matris**, som beskrivs ovan. Vidare antas att:

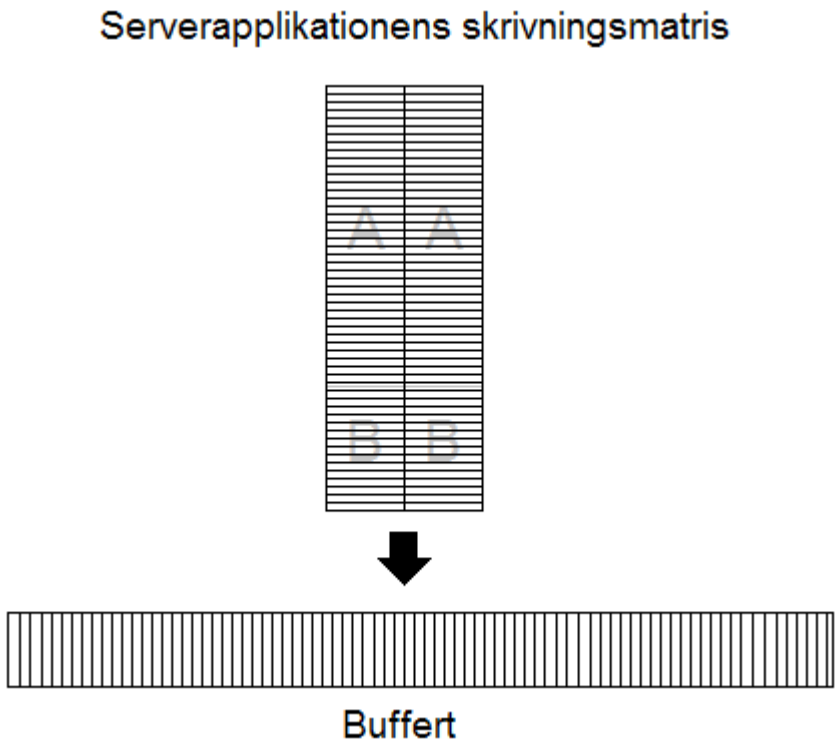
1. Serverapplikationen skickar följande datavärden till central-noden: 1, 'A', 2, 3 och 'C'.
2. Serverapplikationen skickar följande datavärden till perifer-noden: 2, 'B', 3, 4 och 'P'.
3. Perifer-noden registrerar antalet knapptryckningar, som görs på den. Värdet lagras i variabeln variabelEtt i perifer-nodens läsningsmatris.
4. Datavärdena från nodernas läsningsmatriser skickas till serverapplikationen.

Överföring från serverapplikation till central-nod

Hela systemet är konfigurerat så att variablerna, i den första kolumnen av varje skrivningsmatris, tilldelas de datavärden, som är ämnade för central-noden. Variablerna, i den andra kolumnen, tilldelas de datavärden, som är ämnade för perifer-noden.

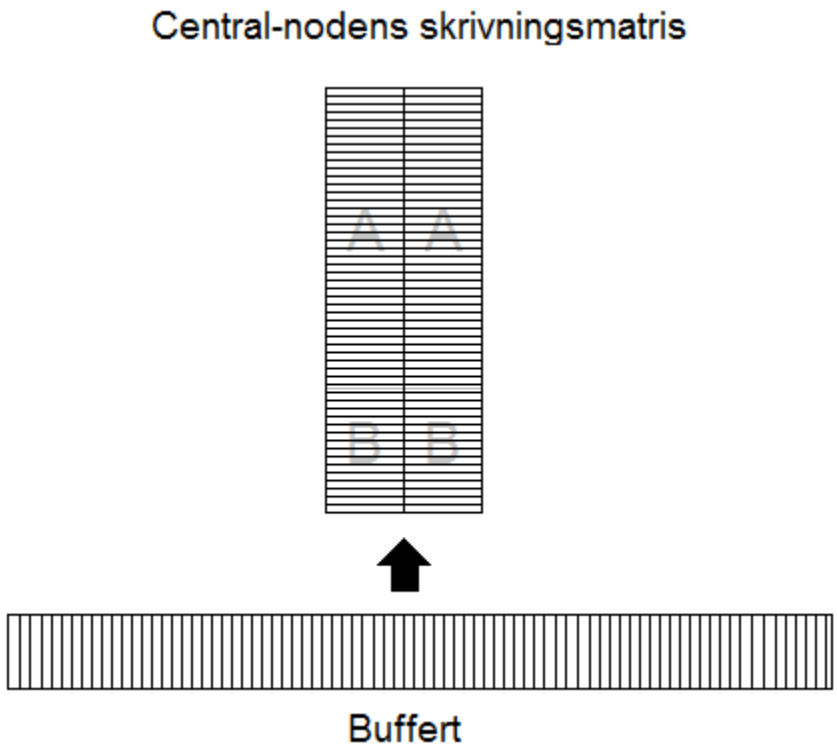
Variablerna, i serverapplikationens skrivningsmatris, tilldelas de värden som ska sändas till noderna. Den första kolumnens variabler tilldelas värdena 1, A, 2, 3 och C. Variablerna, i den andra kolumnen, tilldelas värdena 2, B, 3, 4 och P.

För att kunna skicka datavärdena, till central-noden, skapas en vektor som fungerar som en buffert. Varje element, i buffertvektorn, tilldelas en byte från skrivningsmatrisen.



Figur 8.8: Buffertvektorns element tilldelas alla bytes från skrivningsmatrisen.

Efter sändning mottas alla buffertens bytes av modemet och sänds till central-noden. I central-nodens applikation definieras en buffertvektor vars element tilldelas värdena, av de bytes, som kommer in från modemet. Central-nodens skrivningsmatris tilldelas alla bytes som buffertvektorn innehåller.



Figur 8.9: Elementen i central-nodens skrivningsmatris tilldelas alla bytes från buffertvektorn.

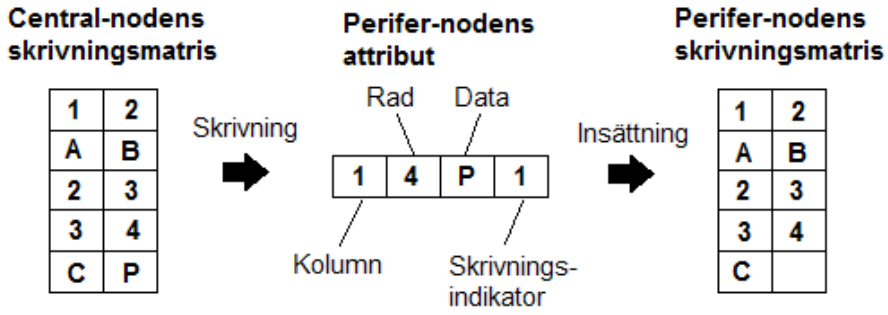
Efter överföringen innehåller variablerna, i central-nodens skrivningsmatris, samma värden, som variablerna i serverapplikationens skrivningsmatris. Den vänstra kolumnens variabler, i central-nodens skrivningsmatris, innehåller alltså värdena 1, 'A', 2, 3 och 'C'. Variablerna i den högra kolumnen, av samma matris, innehåller värdena 2, 'B', 3, 4 och 'P'.

Överföring från central-nod till perifer-nod

Efter att central-noden har mottagit alla datavärdena, från serverapplikationen, initieras processen för att överföra dessa till perifer-noden.

Processen består av överföringar, som central-noden gör till perifer-noden. Varje överföring är en skrivning till perifer-nodens attribut och innehåller fyra bytes. Av dessa utgörs en byte av den data, som är ämnad till skrivningsmatrisen. En byte används för att indikera att överföringen innehåller data ämnat för perifer-nodens skrivningsmatris. Två bytes används för att specificera den kolumn och den rad, i skrivningsmatrisen, som ska tilldelas den överförda databyten. Överföringarna görs tills dess att alla bytes, i central-nodens skrivningsmatris, har överförts till perifer-noden och satts in i dess skrivningsmatris.

Figuren nedan illustrerar överföringen av den sista byten i central-nodens skrivningsmatris.



Figur 8.10: Illustrerar överföringen, från central-nod till perifer-nod, av den sista byten i skrivningsmatrisen. För enkelhetens skull används tecken- och decimalrepresentationer av alla bytes.

Överföring från perifer-nod till central-nod

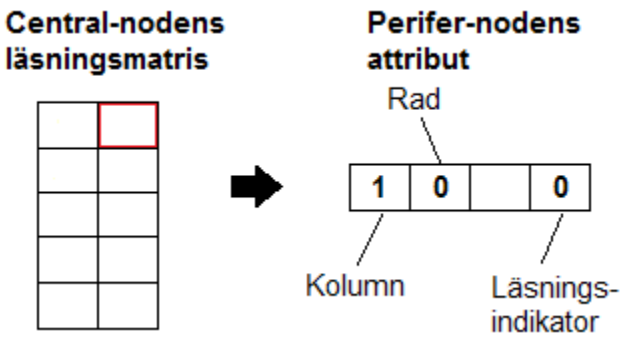
Efter att överföringen till perifer-noden har fullbordats, initieras processen för hämtningen av den data, som lagrats i perifer-nodens läsningsmatris.

Processen består av läsningar, som central-noden gör av varenda byte, i perifer-nodens läsningsmatris. Efter varje läsning lagras den erhållna byten i central-nodens läsningsmatris. Läsningarna pågår tills dess att alla bytes, i perifer-nodens läsningsmatris, har hämtats till central-nodens läsningsmatris.

Innan en läsning görs, sänder central-noden en läsningsbegäran till perifer-noden. En läsningsbegäran är en skrivning till perifer-nodens attribut och innehåller fyra byte. Två byte specificerar kolumnen och raden, i läsningsmatrisen, som ska läsas.

En byte används för att indikera att överföringen är en läsningsbegäran. Den resterande byten används inte.

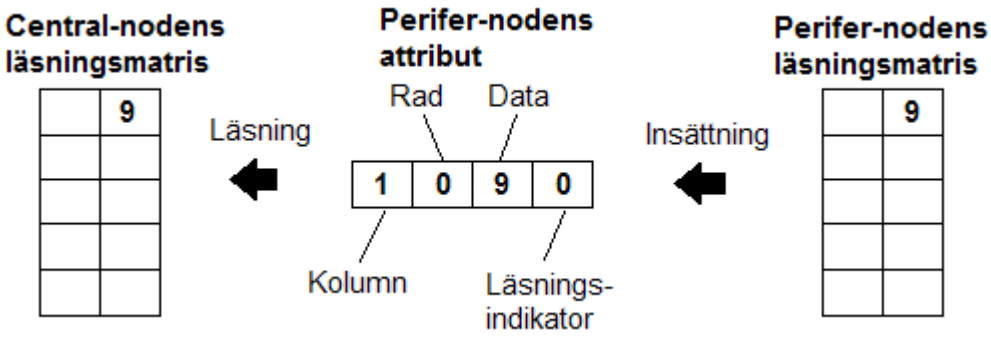
Figuren nedan visar en läsningsbegäran av den byte, som utgör variabelEtt i den högra kolumnen av läsningsmatriserna. I perifer-nodens läsningsmatris innehåller variabeln värdet på antalet knapptryckningar, som görs på noden.



Figur 8.11: Illustrerar en läsningsbegäran, till perifer-noden, av den byte, som utgör variabelEtt i den högra kolumnen av läsningsmatriserna.

När perifer-noden mottar en läsningsbegäran, hämtas den begärda databyten från läsningsmatrisen och sätts i attributet. Därefter gör central-noden en läsning av attributet. När läsningen slutförs, lagras databyten i central-nodens läsningsmatris.

Figuren nedan illustrerar läsningen av den byte, som utgör variabelEtt i läsningsmatriserna. Det antas att nio knapptryckningar görs på perifer-noden.

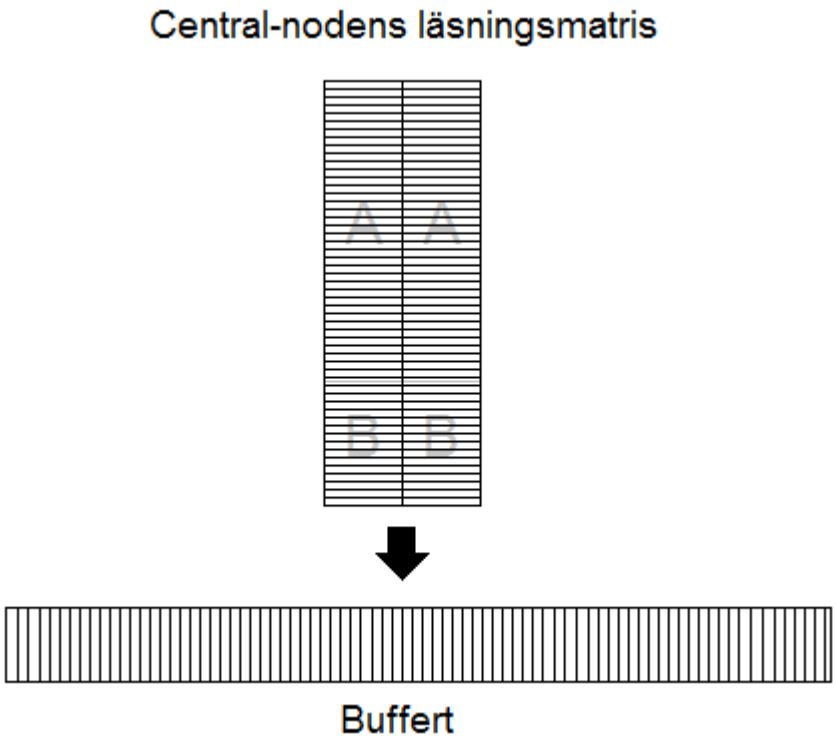


Figur 8.12: Illustrerar läsningen av byten som utgör variabelEtt. Variabelns värde antas vara nio.

Överföring från central-nod till serverapplikation

När hämtningen från perifer-nodens läsningsmatris har slutförts, initieras processen för överföringen från central-noden till serverapplikationen.

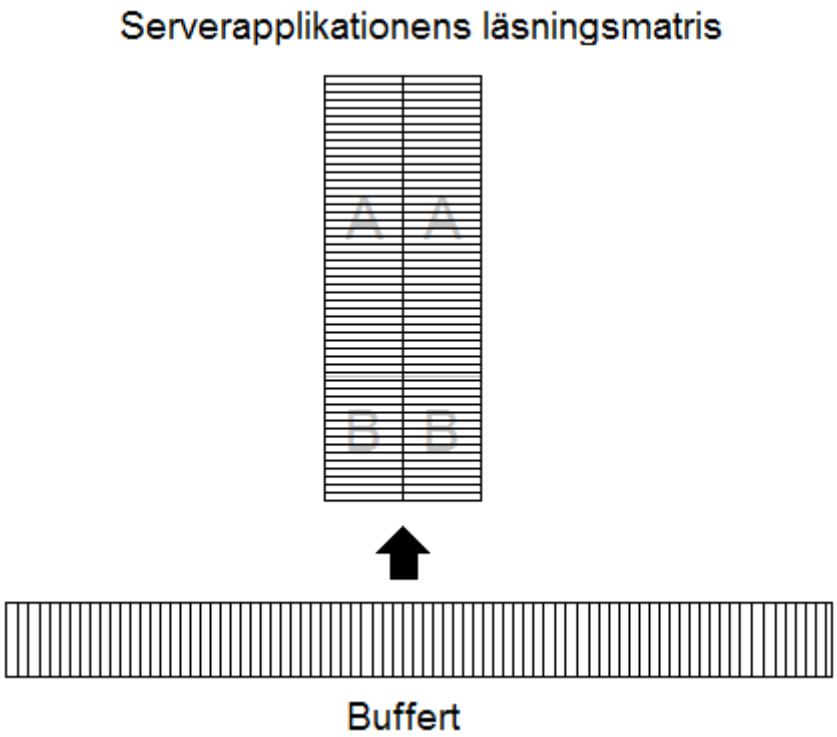
Återigen används, i central-nodens applikation, den fördefinierade buffertvektorn. Dess element tilldelas läsningsmatrisens bytes.



Figur 8.13: Buffertvektorns element tilldelas alla bytes från central-nodens läsningsmatris.

Alla bytes, i buffertvektorn, sänds till serverapplikationen via modemet.

Elementen, i serverapplikationens buffertvektor, tilldelas alla inkommande bytes, som sänts från central-noden via modemet. Därefter tilldelas elementen, i serverapplikationens läsningsmatris, alla bytes från buffertvektorn.



Figur 8.14: Elementen, i serverapplikationens läsningsmatris, tilldelas alla bytes från buffertvektorn.

Därmed har alla bytes från perifer-nodens läsningsmatris överförts till serverapplikationen. Värdet av variabelEtt, i den högra kolumnen av serverapplikationens läsningsmatris, anger antalet knapptryckningar, som har gjorts på perifer-noden.

9 Slutsats

Systemet är utvecklat för dataöverföringen mellan serverapplikation och noderna. Under hela utvecklingen har de förutbestämda kraven på systemet beaktats (se sektion 1.3 Problemformulering).

Central-noden är ansluten till ett modem, vilket upprättar en förbindelse med serverapplikationen vid dataöverföring. Detta uppfyller det första kravet i problemformuleringen.

Serverapplikationen är utvecklat i programmeringsspråket C++, med hjälp av utvecklingsramverket Qt och körs på en PC-datorn. Detta uppfyller det andra kravet i problemformuleringen.

Perifer-noden lagrar data, vilket indikerar antalet knapptryckningar som har gjorts på den. Detta data läses av central-noden och sänds till serverapplikationen. På detta sätt demonstreras dataöverföringen från noderna

till serverapplikationen. Detta bidrar till att uppfylla det tredje kravet i problemformuleringen.

Data, vilket är ämnat till båda noderna, sänds från serverapplikationen till central-noden. I central-nodens applikation lagras detta data och sänds vidare till perifer-noden, där det också lagras. Detta demonstrerar att dataöverföring, från serverapplikationen till båda noderna noderna, är möjlig och bidrar till att vidare uppfylla det tredje kravet i problemformuleringen.

Central-noden och perifer-noden tar emot och lagrar all data, vilket har sänts från serverapplikationen. Central-noden lagrar alltså även data, vilket är ämnat till perifer-noden och perifer-noden lagrar data vilket är ämnat åt central-noden. Detta uppfyller det fjärde kravet i problemformuleringen, att varje nod ska lagra alla andra noders data i systemet.

Lagringen och hanteringen av data, inom serverapplikationen och nodernas applikationer, sker med hjälp av matriser. Matriserna utvecklas för att erbjuda möjligheten för lagring av olika sorters variabler.

Utvecklingen av systemet utgår från en grundläggande konfiguration, vilken består av serverapplikation, central-nod med modem och perifer-nod. Detta uppfyller det femte kravet i problemformuleringen. Det är dock möjligt att utveckla systemet och möjliggöra flera olika konfigurationer.

10 Framtida utvecklingsmöjligheter

Systemet utgör en grund, vilken erbjuder dataöverföringen mellan noder och serverapplikation.

Noderna kan konfigureras, så att mottagen data bearbetas och hanteras på önskat sätt. Till noderna kan, i princip, valfria sensorer kopplas för olika mätningar. Mätdata lagras i nodernas matriser och sänds sedan till serverapplikationen.

Vidare kan data, vilket sänds från serverapplikationen, användas för att konfigurera noderna på olika sätt. Exempelvis kan detta data användas för inställningen av sensorer, anslutningar, timrar och så vidare.

Serverapplikationen kan utvecklas, för att erbjuda ett användargränssnitt vilket gör att insättningen och avläsningen av data underlättas. Vidare kan en databas kopplas till serverapplikationen, för lagringen av data, vilket mottas från noderna.

Central-noden har möjlighet för uppkoppling till flera olika perifer-noder. Central-nodens applikation kan konfigureras så att dataöverföring till flera perifer-noder möjliggörs. Central-noden har också möjligheten att agera som en perifer. Olika central-noder och perifer-noder kan alltså användas för att utgöra en mängd olika systemkonfigurationer (se appendix 13.2).

11 Terminologi

uint8 – Står för 8-bit unsigned integer. Är en datatyp som består av 8 bitar och kan anta värden från 0 till 255.

uint16 –Står för 16-bit unsigned integer. Är en datatyp som består av 16-bitar och kan anta värden från 0 till 65535.

Union – Inom många programmeringsspråk tillåter unioner att en delar av minnet nås som olika typer av data. Exempelvis kan en uint16 variabel läsas som en vektor av två uint8 element.

API-funktioner – Är funktioner, vilka används för att möjliggöra kommunikation mellan olika programvaror.

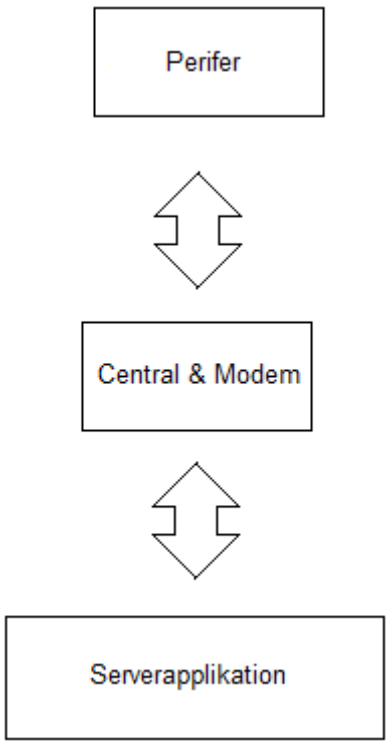
12 Referenser

[1] Bluetooth Low Energy CC2540/41 Mine Development Kit User's Guide
[2] CC2540/41 Bluetooth Low Energy Software Developer's Guide
[3] CC2540 Bluetooth Low Energy API Guide
[4] OS Abstraction Layer Application Programming Interface
[5] HAL Drivers Application Programming Interface
[6] Gemalto MC55i Datasheet
[7] MC55i Terminal Hardware Interface Description
[8] MC55i AT Command Set
[9] MAX3232 Datasheet
[10] Foundations of Qt Development
[11] Qt Assistant
[12] Mobile Internet
[13] Data Communications and Networking

13 Appendix

13.1 Grundläggande konfiguration

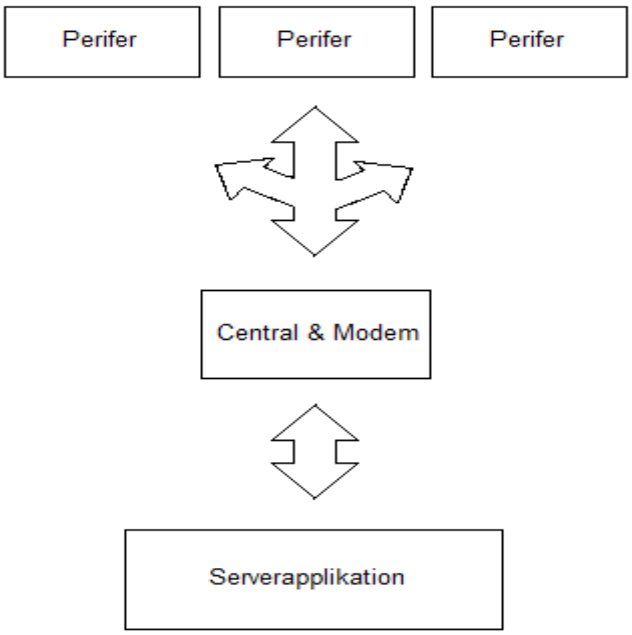
Nedan illustreras den grundläggande konfigurationen, av systemet, vilken utvecklas i projektet. Pilarna indikerar dataöverföringen och dess riktning.



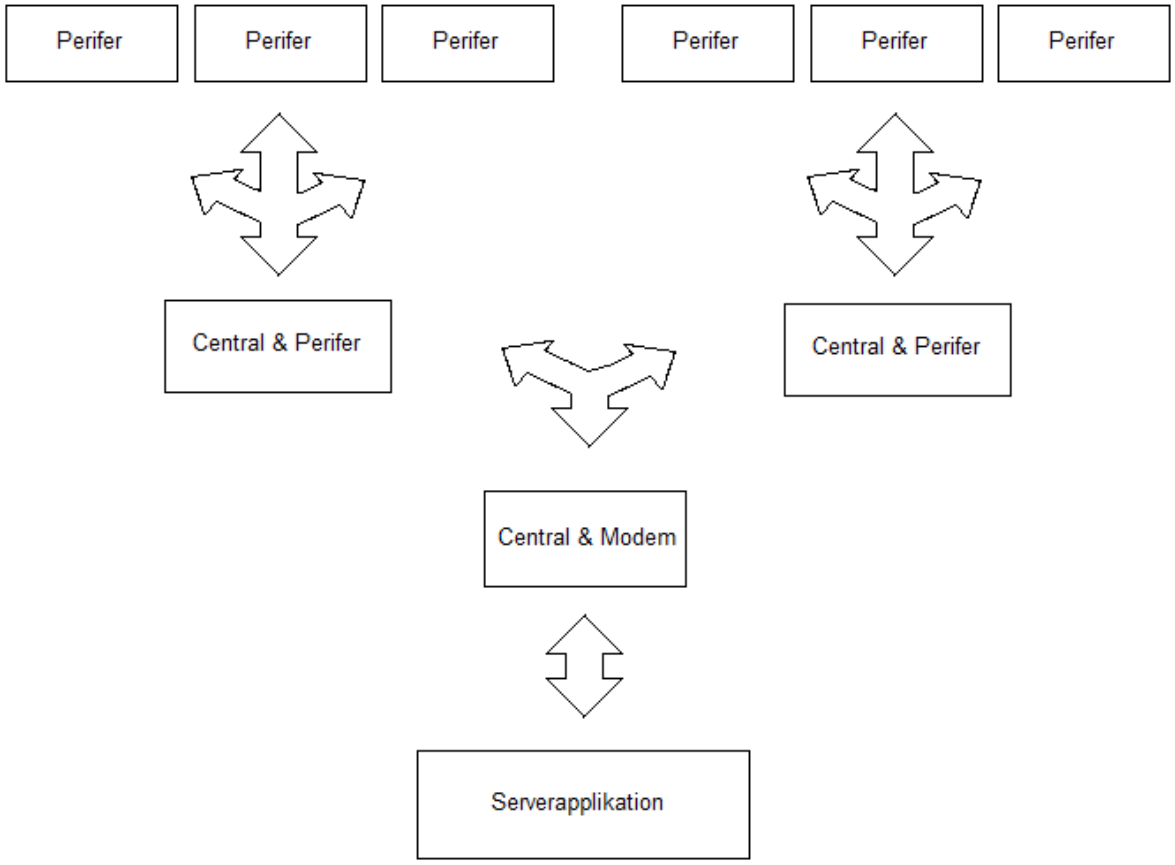
Figur 13.1: Illustrerar grundkonfigurationen, vilken skapas i projektet.

13.2 Exempel på potentiella konfigurationer vid framtida utveckling.

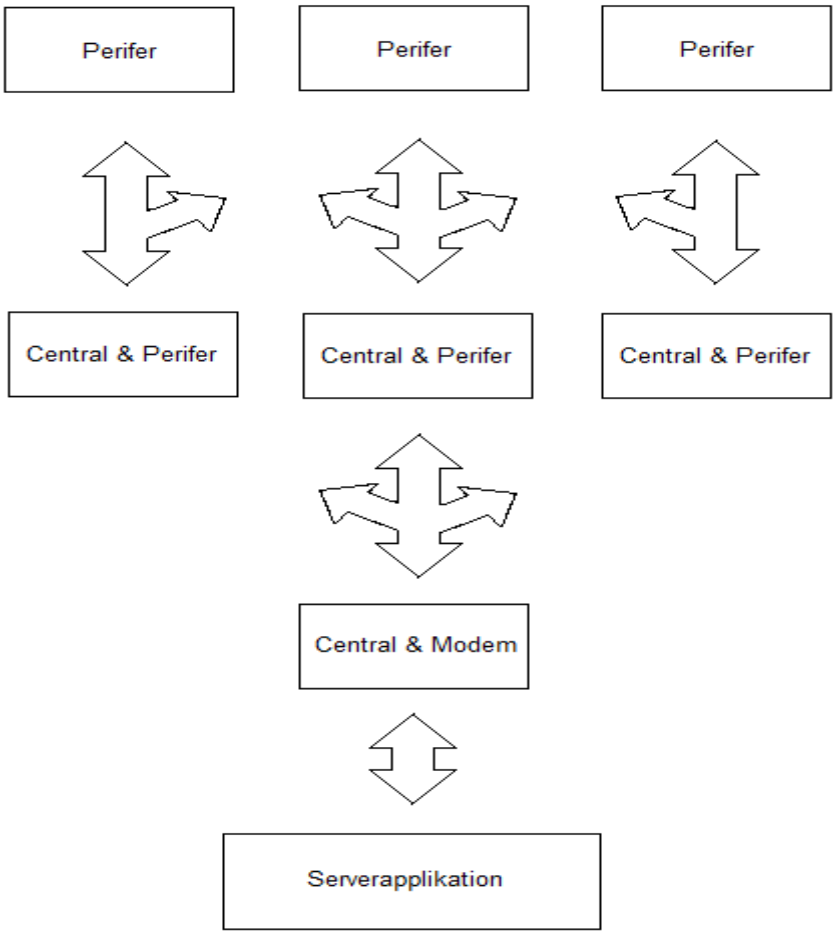
Nedan illustreras potentiella konfigurationer av systemet. Pilarna indikerar dataöverföringen och dess riktning.



Figur 13.2: Illustrerar en potentiell konfiguration, av systemet, innehållande tre perifer-noder och en central-nod, vilken är sammankopplad med ett modem.



Figur 13.3: Illustrerar en potentiell konfiguration av systemet innehållande sex perifer-noder, två noder vilka agerar både som central- och perifer-nod, samt en central-nod, vilken är sammankopplad med ett modem.



Figur 13.4: Illustrerar en potentiell konfiguration av systemet innehållande tre perifer-noder, tre noder vilka agerar både som central- och perifer-nod, samt en central-nod, vilken är sammankopplad med ett modem.