CMS

Projektrapport

World Wide Medico

1 Resume

1.1 Dansk

Denne opgaves formål er, at udvikle en distribueret udgave af Lokal Monitor Systemet (LMON), det blev udviklet i et tidligere kursus. LMON er et system til at overvåge en enkelt patients tilstand, herunder pulsen, EDR, ECG og en pumpe.
Disse værdier vises efterfølgende på en touch screen lokalt hos patienten, der kan overvåge patientens grænseværdier samt give alarm hvis disse overskrides.

I denne distribuerede udgave, er det muligt at overvåge en eller flere patienter fra f.eks. en sygeplejerskes ellers læges kontor. Det er muligt at monitorere flere patienter på en gang, samt det er muligt at se en enkelt patients tilstand med grafer, som på en LMON. Systemet kan alarmere hvis grænseværdierne overskrides, samt det er muligt at ændre en patients grænseværdier fra den distribuerede udgave.

Opgavens problemstilling var at udvikle en distribueret realtidsapplikation ved at designe og implementere flere designmønstre fra POSA 2. Mønstrene i POSA 2 bogen gjorde udviklingen af et støre distribueret system mere overskueligt.

Vi har gennem udviklingsforløbet opbygget et større framework. Frameworket består af 3 pakker, hvoraf den ene har til opgave at gøre systemet operativsystem-uafhængigt ved at skjule operativsystem specifikke kald. En anden del af frameworket stiller de generelle netværks mønstre til rådighed, så de ikke skal implementeres forfra på alle servere der ønsker at bruge dem. Den sidste del af frameworket er en system manager, som holder styr på hvor alle de andre servere er placeret, og stiller kommunikation til de andre netværks enheder til rådighed, for applikationer der benytter frameworket.

Den oprindelige LMON er udvidet med et netværkskommunikationsmodul, så den nu kan kommunikere med andre delsystemer.

1.2 English

The purpose of this assignment is to create a distributed version of the Local Monitor System (LMON), which was created in an earlier course. LMON is a patient monitoring system aimed for usage in the hospital sector, and receives signals from a patient and a pump.
ECG, EDR and pulse signals are received from the patient that subsequently is shown on LMON's display. LMON must watch the border limits for the patient, and give an alarm if these are exceeded.
From the pump the infusion rate is received, or an alarm message if an error occurs. The rate can be adjusted both on the pump and on LMON. The pump can be started and stopped from LMON.

In the distributed version, it should be possible to supervise one or more hospitalized patients from a central place in the hospital. It should be possible to monitor more than one patient, and it should be possible to choose to monitor a patients vital signals. The system should make an alarm if one of the border limits is exceeded. In addition, it is possible to change patients' border limits, from the distributed version.

The assignment was to develop a distributed real time application, by designing and implementing some of the design patterns from the POSA 2 book. The use of the patterns in POSA 2 made the development of the system easier, and clearer.

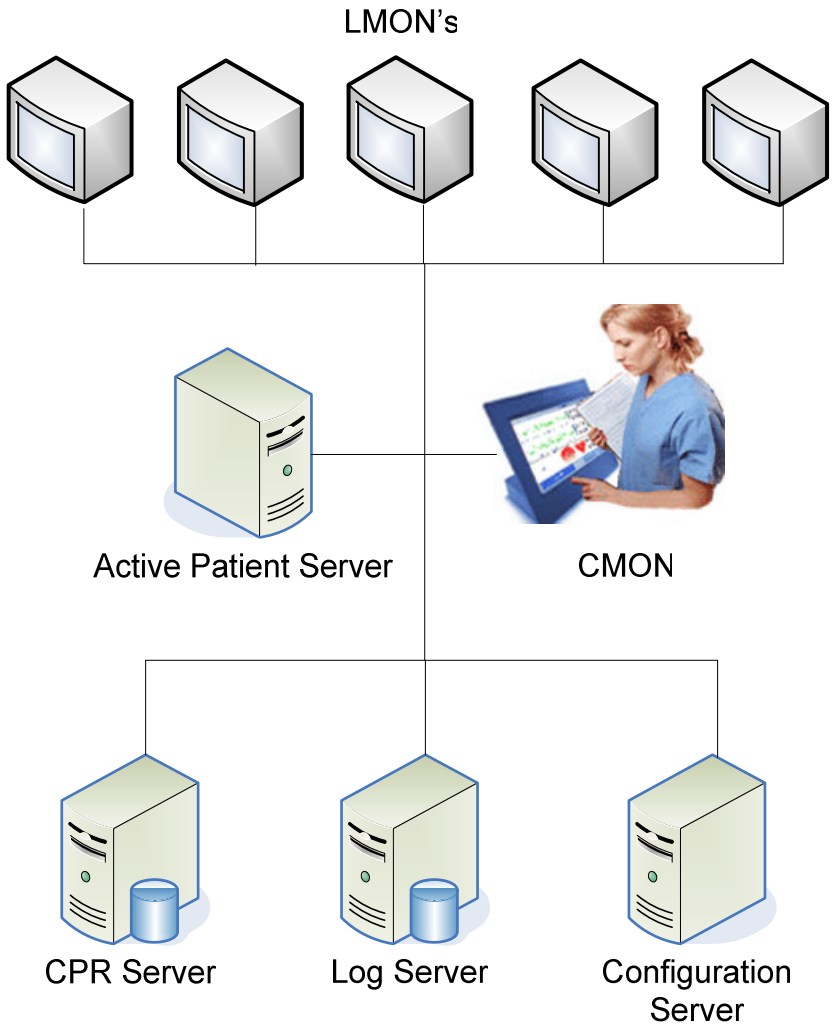
During the development process we created a framework. The framework consisted of 3 packages, where one of them is assigned to make the system operating independent, by hiding operating system specific calls. Another part of the framework offers the use of general network design patterns, so they don't have to be implemented again on all the servers. The last part of the framework is a system manager, which manage the location and communication to the main servers in the system.

The original LMON system is extended with a network communication module, so it can communicate with other subsystems.

2 Indholdsfortegnelse

3 Indledning

Denne projektrapport beskriver fortsættelsen på faget Indlejrede Realtids Systemer – Distribuerede Realtids Systemer. Dette projekts formål var, at videreudvikle en eksisterende løsning fra IRTS, Local Monitor System(LMON), til at indgå som klient i et distribueret system. Der skulle derudover også udvikles en server, Central Monitor (CMON), til at kommunikere med disse distribuerede klienter.
For at gøre designet så skalerbar som muligt, har vi delt arbejdsopgaverne ud til forskellige servere, der så arbejder sammen for at skabe et komplet system.



Figur 1: Ovesigt over hele systemet

CMON er brugerens indgang samt interaktion med systemet. Hver patient indlagt, er tilknyttet en LMON. Hver LMON registrere sig hos Active Patient Server, der holder styr på, hvilken patient der ligger ved hvilken LMON. CMON kommunikere med Active Patient Server, for at få en liste over indlagte patienter.
CPR-serveren benyttes til at finde navnet på en patient. Når en patient indlægges, registreres patienten ved sit CPR-nummer, og navnet findes så ved at benytte CPR-serverens service. Log

serveren benyttes til at logge vigtige events fra hele systemet, så en eventuel fejlfinding for en tekniker vil være nemmere. Configuration Serveren indeholder IP-adresser og portnumre på alle andre servere i systemet, så hvis der tilkobles en ny maskine, skal den kun kende configurations serveren adresse, og ud fra den, få informationer om de andre maskiners adresser.

For at undgår at skrive lange system navne flere gange igennem dokumentet, og for at have et sted man kan slå forkortelser op, har vi lavet en tabel forrest i alle dokumenterne, hvor forkortelserne er for de forskellige systemer er forklaret. Se tabellen herunder.

Forkortelse	Navn	Beksrivelse
CMS	Central Monitoring System	Hele systemet.
LMON	Local Monitor	LMON er ansvarlig for at overvåge en patient.
CMON	Central Monitor	Den centrale server, som kan overvåge flere LMON's
CMONUI	Central Monitor User Interface	CMON's grafiske brugergrænseflade.
COMCMON	COM Central Monitor	COM komponenten som forbinder CMON og CMONUI.
AP	Active Patient	En aktiv patient, er en patient forbundet til en LMON, som er forbundet med en APS.
APS	Active Patient Server	APS er ansvarlig for at registrerer de AP'er, og skal notificerer CMON om ændringer.

Under udvikling af CMON er der lagt vægt på dokumentering, realtid og designmønstre. Dokumenteringen er lavet ved hjælp af use case teknikken og UML.

Fra det tidligere projekt har vi kendskab til de problemer der kan forekomme, når der udvikles et realtidssystem. I udviklingen af CMS er disse erfaringer benyttet til at reducere netværkskommunikationen så meget som muligt, så vigtige pakker ikke bliver fanget i flaskehalse.
Der er ligeledes taget højde for, at systemet skulle kunne udvides med mange klienter samt mange servere, og er derfor opdelt i mindre delsystemer der tilbyder en specifik service.

Gennem udviklingsforløbet har vi løbende taget stilling til hvilke designmønstre der var bedst anvendelige i CMON. Gennem kursets forløb er der blevet implementeret flere forskellige mønstre, hvilke der også har indgået i udviklingen.

Rapporten består af to dele; denne projektrapport der beskriver projektet og den valgte arbejdsproces, samt en projekt dokumentation, der detaljeret beskriver udviklingen af produktet.

Projektrapporten består af resume, indledning, beskrivelse af udviklingsforløbet, samt en konklusion.

Projekt dokumentationen består af kravspecifikation, design og test.

3.1 Læsevejledning

Litteraturhenvisninger er skrevet på følgende format [forfatter – publicerings år].
Et eksempel på dette er : [GOF-94].
For en komplet liste over refererede bøger, se Litteraturlisten, fane 6.

For yderlig information omkring krav til systemet og use cases, se ”CMON Requirement Specification”, fane 2 .

En detaljeret gennemgang af systemets design, realiseringer af use cases og implementation af designmønstre findes i ”CMON Design”, fane 3.

En test af test af use cases samt af det samlede system kan læses i ”CMON Test”, fane 5.

4 Projekt beskrivelse

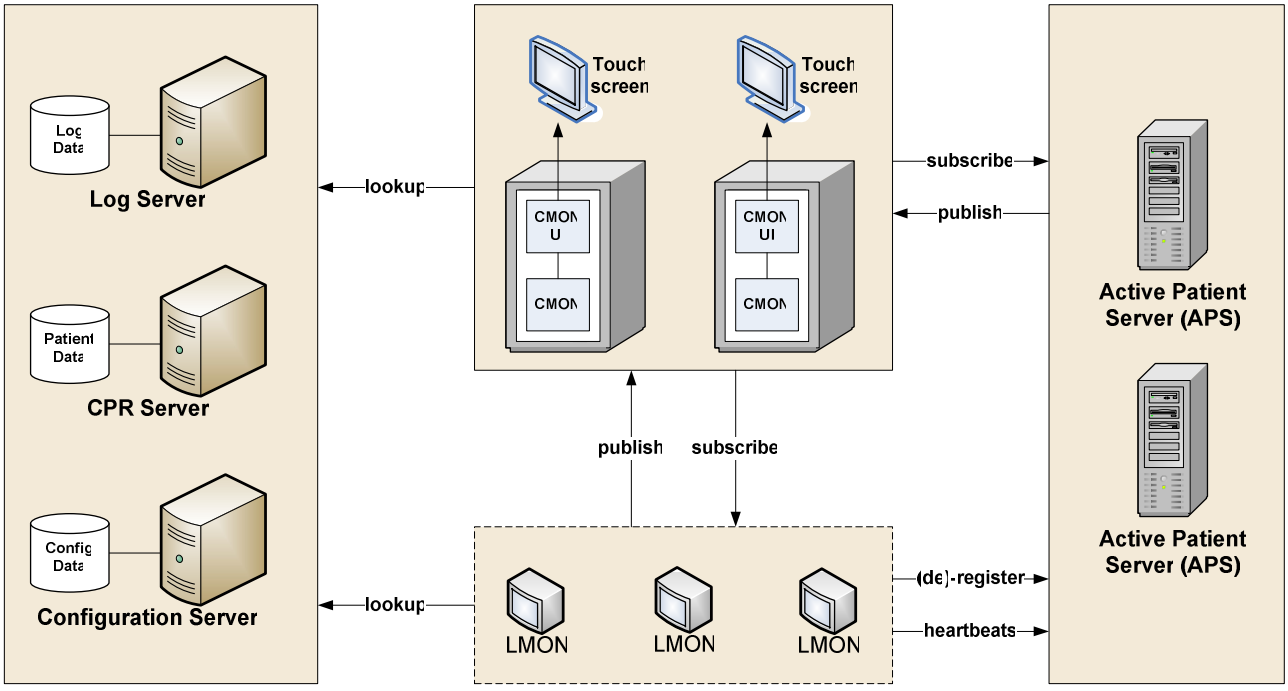
4.1 Projektgennemførelsen

Dette projekt kan ses som værende tredje iteration i forløbet, med de to iterationer i LMON som værende de første to. I denne iteration er systemet blevet distribueret og der er udviklet en række servere der stiller en bestemte services til rådighed.

Udviklingen af systemet blev delt op i flere delsystemer, som kunne laves uafhængigt.

- LogServer
- CPR Server
- Active Patient Server
- LMON
- CMON
- CMONUI
- Configurations server

Figur 2 viser et diagram over systemet med alle servere.



Figur 2 Oversigt over hele systemet.

Vi har i gruppen samarbejdet for at løse opgaven, og delt mindre opgaver ud imellem gruppemedlemmerne. Ligeledes har vi gjort brug af teknikker fra eXtreme Programming som pair programming.

For at styre projektet, udarbejdede vi tidligt i forløbet en tidsplan, så det var muligt at strukturere arbejdet bedre.

Uge	Kravspecifikation	Analyse & Design	Implementering	Test	Rapport
45	**X**				**X**
46	**X**				**X**
47	**X**				
48		**X**	**X**		
49			**X**		**X**
50				**X**	**Aflevering**

Felter markeret gråtone var den oprindelige tidsplan, og felterne med **X** markere den faktiske tidsfordeling.

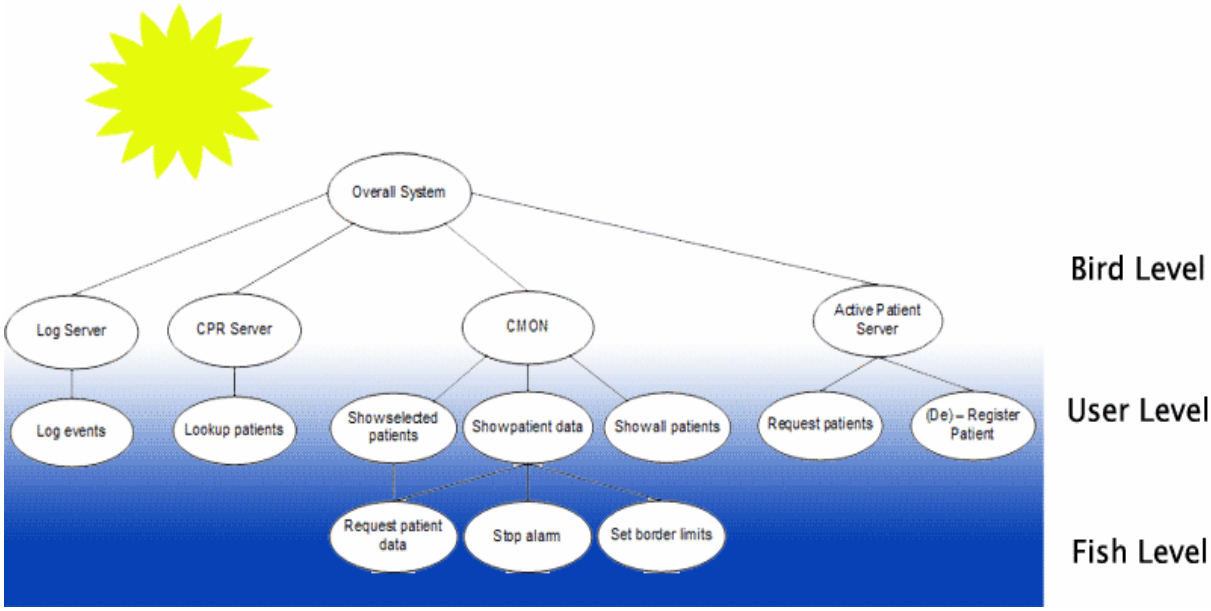
4.2 Metoder

Systemet er lavet ved hjælp af forskellige OOP teknikker. Use Case teknikken er brugt til at beskrive de funktionelle krav i systemet. UML blev brugt til at diagrammere systemet. Designmønstre er brugt til at modellere systemet.

4.3 Specifikations- og analysearbejdet

Til at specificere og analysere systemet har vi primært brugt use cases. Vi har delt vores Use Cases op, så de enten er bruger-, eller system initierede. Brugerinitierede use cases initieres af en aktør, og systeminitierede initieres af systemet.

Use casene er delt op i tre niveauer[Cockburn-01]. 'Bird level', 'user level' og 'fish level'.
'Bird level' beskriver de meget abstrakte use cases. Det vil sige, at det kan være ting som LMON, CMON eller andre delsystemer.
'user level' beskriver, kort sagt, use cases som en primær bruger kan benytte sig af, og bagefter gå tilfreds væk. En 'user level' use case kan have en eller flere 'fish level' use cases under sig.
'fish level' er mere detaljeret, og her kan beskrives detaljerede ting, som er meget system kritiske. En 'fish level' use case har en eller flere 'user level' use cases over sig.
Vores use cases er lavet, så de primært ligger på 'user level'. Det vil sige at de er mere abstrakte end hvis de f.eks. lå på 'fish level'. Figuren herunder illustrerer hvorledes vores use cases ligger.



Figur 3 Oversigt over use case niveauerne

Det har været nemt at specificere kravene ved hjælp af use case teknikken, da man hurtigt får et overblik over hvad hele systemet skal kunne.

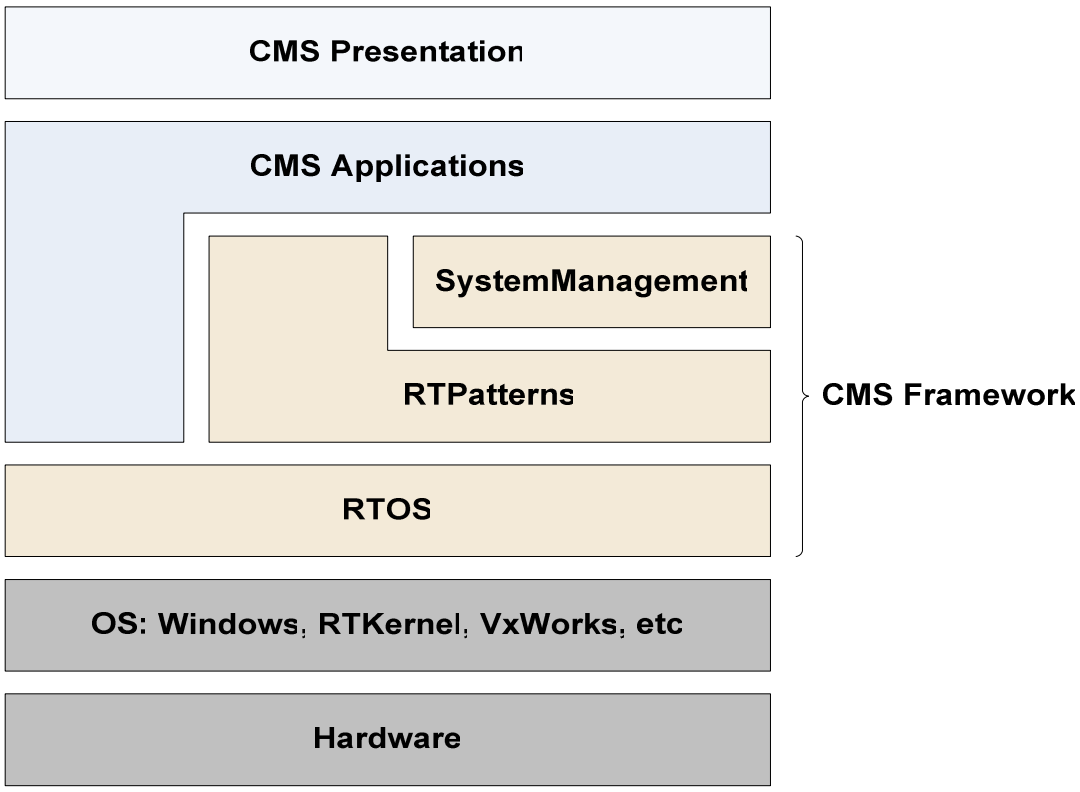
4.4 Designprocessen

”Et par udvalgte designløsninger ville også gøre sig i denne rapport – selv om det er lidt gentagelse af produktdokumentationen”

Vi har anvendt ROPES designprocessen til udvikling af systemet. ROPES er en iterative udviklings proces, der anvender use cases i centrum for udviklingen. Vi har i CMON prcoessen gennemløbet 1 iteration. Iterationen kan ses som 3 iteration af det samlede projekt, som består af LMON og CMS. Vi har anvendt 4+1 view modellen til at afspejle systemet fra flere vinkler.

Vi fik i design processen hurtigt delt hele systemet op i flere delsystemer, med forskellige ansvars områder. Dette gjorde design af forskellige services i systemet mere overskueligt. Alle delsystemer havde dog en del til fælles, så for at undgår at lave de samme ting igen og igen, lavede vi et framework som kunne bruges af samtlige servere. Frameworket bestod af 3 pakker, alle delsystemer skulle inkludere. Den ene pakke var RTOS, som vi havde implementeret dele af i faget IRTS. RTOS funktionalitet var at gøre programmet operativsystemuafhængigt, ved at wrappe operativsystem specifikke kald ind. Vores RTOS pakke kan bruges på operativsystemerne Windows og RTKernel. Vores anden generelle pakke var RTPatterns. RTPattern pakkens opgave var at samle alle større designmønstre, så de nemt kunne genbruges. Alle delsystemer skulle f.eks. bruge designmønstret reactor. I RTPatterns pakken havde vi også defineret de forskellige pakkeformater, for at gøre det nemmere at sende data over netværket. Den sidste pakke vi lavede kaldte vi for SystemManagement.

SystemManagement's opgave, var at holde styr på hvilke services der var til rådighed i hele systemet, og stille dem til rådighed for de overlæggende applikationer.



Figur 4 System arkitektur,

Figur 4 viser system arkitekturen. RTOS pakken er generel og kan til enhver tid genbruges af andre applikationer vi ønsker at lave, som skal køre på enten Windows eller RTKernel. RTOS pakken kan også nemt udvides til at omfatte andre operativsystemer. Størstedelen af RTPatterns pakken er generel, og kan genbruges af andre applikationer vi ønsker at lave. SystemManagement pakken er specifik for CMS, og kan kun bruges til applikationer, som er en del af CMS. I præsentation laget ligger applikationer tilbyder brugergrænseflader til applikationerne der ligger på CMS applikationslaget.

Til at modellere og beskrive systemet er anvendt UML og kendte designmønstre. Begge dele gør systemet mere struktureret, og nemmere at arbejde videre på, hvis det senere skal videreudvikles af os selv eller af andre. Designmønstrene har også givet os et højere abstraktions niveau, hvorved det er nemmere at diskutere hvorledes softwaren skal opbygges.

Vi har brugt UML til at beskrive softwaren, samt systemets funktionalitet. De arkitektursignifikante use cases er beskrevet ved hjælp af sekvensdiagrammer. Softwaren er modeleret ved hjælp af pakke- og klassediagrammer. Systemets tilstand er modelleret ved hjælp af et tilstandsdiagram, som beskriver alle systemets tilstande, og actions der provokerer tilstandsskift.. Systemets processer er modeleret ved hjælp af et proces/task view, som beskriver de forskellige tråde i systemet, og kommunikationen imellem dem.

4.5 Udviklingsværktøjer

4.5.1 Microsoft Visio 2003

Til at lave use case diagrammer og UML diagrammer er brugt Microsoft Visio 2003(Visio). Visio er et program som kan bruges til at tegne UML og use case diagrammer. Det kan ikke generere kode for en, det kan ikke løbe en sekvens igennem og teste det, og det kan ikke opdatere diagrammer løbende ud fra den kode der udvikles.

Vi har valgt at bruge Visio, da det er det vi kender bedst, og det opfylder de krav vi har til et sådan værktøj.

Alternativer er Rhapsody 5.0 og Rational Rose UML tool. Begge er mere avancerede end Visio, og har ikke de svagheder, som vi har nævnt ved Visio.

4.5.2 Microsoft Visual Studio .NET 2003

Til udviklingen af systemet, har vi brugt C++ og C#. Vi har udviklet systemet i Microsoft Visual Studio .NET 2003(VS).
Størstedelen af systemet er skrevet i C++. C# er kun brugt til at lave brugergrænsefladen på til CMON. Brugergrænsefladen til CMON kan således ikke køre på andre operativsystemer end Windows.
Da hele systemet ligger oven på vores RTOS pakke kan det nemt lade sige gøre at teste det hele på en Windows maskine. De servere der ikke har noget at gøre med brugergrænsefladen, kan således også nemt køres på en SBC686.

4.6 Resultater

Det er lykkedes at udvikle de servere vi havde defineret i starten i vores projektforløb. Det vil sige:

- En CPR-server, som er en server der holder styr på CPR-numre, og deres tilknyttede informationer,. CPR-serveren læser CPR-informationerne fra en fil. CPR-serveren kunne udvides, så den fik sine informationer fra en central CPR-server.
- En aktiv patient server(APS). Den aktive patient server holder styr på alle patienter der er tilknyttet en LMON.
- En log server hvor alle andre servere kan logge information.
- En configurations server, som holder styr på, hvor de centrale servere i systemet er. Alle andre systemer der tilkobles, skal således kun kende configurations serverens adresse, og kan derfra få de andre vigtige adresser til andre enheder.
- En CMON klient som kan tilgå de andre servere. CMON kan, efter den har fået den aktive patients servers adresse, tilgå denne og få en liste med samtlige patienter. Herefter kan CMON subscribe på forskellige data fra en eller flere LMON'er.

- Det er lykkedes at lave en brugergrænseflade i C#, som kan kommunikere med CMON klienten, som er skrevet i C++. Kommunikationen er lavet ved at koble CMON klienten ind i en COM DLL, som derefter kunne linkes ind i et C# projekt.
- Netværks modulet til LMON er lavet og kan kommunikere med de andre servere. Vi nåede dog ikke inden afleverings tidspunktet at få det koblet sammen med en LMON, men har i stedet lavet en simpel brugergrænseflade til at teste netværksmodulets funktionalitet.

Vi har fået udviklet et framework der tilbyder platform uafhængighed, stiller designmønstre til rådighed, samt står for kommunikationen i CMS. Hele frameworket kan og skal bruges, hvis man ønsker at udvide CMS med andre services. Store dele af frameworket kan genbruges i andre systemer. Heriblandt hele RTOS pakken, som også nemt kan udvides til at omfatte andre operativ systemer. Størstedelen af RTPatterns pakken kan også genbruges i andre projekter. For mere information om frameworket se i design dokumentet afsnit 6.2.2.

Vi har således opfyldt alle kravene stillet i kravspecifikationen.

4.7 Diskussion af opnåede resultater

Til trods for at vi har opfyldt alle krav, er der nogen få ting der kunne klare en mindre finpudsning.
For at kunne udføre en troværdig test af systemet, skal netværks modulet til LMON skal integreres med LMON. Før dette er sket kan vi ikke garantere at vi kan opdatere en CMON med noget der minder om realtid. Vi har dog lavet mindre stresstest af systemet igennem den simple brugergrænseflade til netværkskommunikations modulet til LMON, som ikke tydede på at hverken netværks kommunikation eller kommunikationen mellem CMON og CMONUI kunne blive potentielle flaskehalse.
Kommunikationen mellem CMON og CMONUI er også lavet og fungerer flydende i de fleste scenarier. Vi har dog registret fejl ske ved specielle tilfælde, så en finpudsning af kommunikationen her imellem mangler.

Alt i alt er vi meget tilfredse med hvad vi har lavet, men en ekstra iteration, ville gøre systemet fuldendt, da det vi så kunne fjerne diverse småfejl, og laver en større test, samt performance forbedringer.

4.8 Opnåede erfaringer

Gennem udviklingsforløbet af CMS her vi opnået at få forståelse for brug af designmønstre, både i standard applikationer, og distribuerede systemer. Vi har set hvordan de kan benyttes, både til at modellere et system, samt at genbruge funktionalitet på tværs af applikationer.

Vi har udviklet et framework, og set de fordele det giver, når man kan bruge det igen i flere forskellige slags applikationer.

Vi har modeleret systemet ved hjælp af use case teknikken, og benyttet flere forskellige designmønstre til at gøre koden mere flexibel og letlæselig.

Da vi udvikler et system, som kan være ansvarlig for liv eller død, for de indlagte patienter, har vi hele tiden haft for øje at lave koden så effektiv som muligt. Dette resulterede i, at der skulle tænkes lidt anderledes med hensyn til f. eks oprettelse og nedlæggelse af objekter i forhold til udvikling af ikke realtids applikationer. Løsningen til at udgå at oprette objekter var at anvende singleton pattern, til at sørge for at kun 1 instans af objektet findes. Ligeledes søgte vi at minimere netværkskommunikationen så systemet ikke skulle håndtere mange unødvendige pakker, men kun får pakker der indeholder vigtig information fra andre subsystemer

Vi har også opnået erfaring med kommunikation imellem to forskellige programmerings sprog (C# og C++). Vi ønskede at lave brugergrænsefladen i C# og skulle derfor finde en løsning til at få beskeder imellem brugergrænsefladen (CMONUI) og CMON. Da de to programmer skulle køre på samme maskine, løste vi problemet ved at pakke CMON ind i en COM DLL fil, og integrere den i CMONUI.

4.9 Projektets fortræffeligheder

Der er dele af CMS som vi har lagt ekstra arbejde i, og derfor er ekstra stolte af.

Vi syntes vi har fået lavet en god og intuitiv brugergrænseflade til systemet. Vi har især lagt vægt på at brugergrænsefladen skal kunne bruges til en touch screen, at den skal være flot, at den skal være overskueligt, give brugeren et overblik over systemet, samt at den skal kunne visse alle de ting som systemet kan, altså ikke skjule funktionalitet for brugeren.

Vi er meget stolte af at vi har lavet vores eget framework, som kan bruges på samtlige systemer i CMS. Store dele af frameworket er også så generelle, at de kan tages ud og genbruges i andre projekter. Frameworket har gjort at den resterende kode er meget letlæselig. Dels fordi den ikke skal lave operativsystem specifikke kald, men også fordi det gør det nemt at bruge mere eller mindre avancerede designmønstre uden at skulle bekymre sig om hvordan de er implementeret. Frameworket tager også hånd om at marshale og de demarshale data sendt over netværket, så laver man en applikation der skal køre oven på frameworket, behøver man heller ikke at tænke på de fejl netværks kommunikation ellers kan give.

Vi syntes opsplitningen af systemet i mange små delsystemer har været en rigtig god ide, da det har givet os et bedre overblik, og det derved er både nemmere for os at arbejde med og finde fejl i. Opsplitningen af systemet gør også, at det er nemmere for eksterne udviklere at tilføje eller ændre funktionalitet ved systemet.

CMS er lavet ved hjælp af velvalgte designmønstre, og er godt dokumenteret ved hjælp af UML, så det vil være nemt for en ekstern programmør at få et overblik over koden, og udvide eller ændre den.

4.10 Forslag til forbedringer af projektet eller produktet

En naturlig forbedring til systemet ville være at integrere det med den Elektroniske Patient Journal(EPJ). Står en sygeplejerske ved CMONUI, skulle hun så kunne få alle relevante oplysninger om patienten ved at klikke på dennes navn. Det samme skulle være gældende hvis hun stod ved den indlagte patient ved en LMON.

CPR-serveren virker på den måde, at den modtager et CPR-nummer over netværket, og svare tilbage med navnet på den person der har det givne CPR-nummer. CPR-serveren slår CPR-nummeret op i en lokal fil, og kender den ikke navnet returnerer den teksten: ”Patient unknown”. En naturlig udvikling ville være at koble CPR-serveren op mod en central CPR-server, med alle navne og CPR-numre i Danmark. Kendte vores CPR-server så ikke navnet kunne den spørge videre ud, og få et svar tilbage og gemme det.

5 Konklusion

Dette projektforløb har budt på en række af nye udfordringer. Vi har for første gang udviklet et større distribueret system ved hjælp af designmønstre fra POSA 2 bogen. Det har været en god hjælp at splitte systemet op i mindre delsystemer med forskellige ansvars områder, da det har gjort det nemmere at dele arbejdet op. De mange delsystemer har også gjort det nemmere at fejlrette, da fejlene på den måde har været nemmere at identificere.

Under udvikling af systemet har vi haft meget fokus på at bruge forskellige designmønstre. Vi har fra start af overvejet forskellige muligheder, og er løbende blevet inspireret under kursus-forløbet, til at bruge flere designmønstre. At vi havde haft en del øvelser inden vi gik i gang med selve medico projektet gjorde, at det føltes naturligt at bruge de allerede udviklede designmønstre . Vi mener ikke at have overdrevet vores brug af designmønstre, da vi mener det er af stor betydning at kunne udvikle software på et højere abstraktionsniveau. Det vil være svært i fremtiden ikke at gøre brug af de mønstre vi har lært.

Brugen af designmønstre har også været med til at skabe et ensartet flow igennem hele systemet, og gør koden mere genbrugelig og nemmere at sætte sig ind i for en ekstern udvikler.

En stor del af vores fokus har ligget på at udvikle et framework, som skulle bruges på alle enheder i systemet. Frameworkets opgave var at sørge for operativsystemspecifikke kald, sørge for kommunikation med centrale enheder i systemet, samt at tilbyde forskellige designmønstre, som så nemt kunne bruges af de forskellige applikationer der lå oven på. Frameworket har gjort det nemmere at tilføje nye enheder i systemet, da de alle kun skulle bygge oven på frameworket. Den ekstra indsats der var i starten, ved at gøre alt på den rigtige måde, og pakke det ind i vores framework, har sparet os en del tid i slutningen af projektet. Det udviklede framework er lavet ved hjælp af kendte designmønstre, og er derfor nemt at gennemskue, og kan nemt genbruges i andre store projekter.

Vi har fået udviklet en brugervenlig brugergrænseflade, der på simpel vis giver overblik over de tilsluttede patienter. Det er muligt for en bruger der står et sted, at overvåge mange patienter der er placeret forskellige steder på et sygehus. Brugeren kan ligeledes fra centralt sted ændre på en brugers grænseværdier, samt slå alarmer fra. Da CMS er et system der kan være årsag til at en patient lever eller dør, har vi under udviklingen af brugergrænsefladen lagt stor vægt på, at det skulle være svært for en bruger at begå fejl. Dette har vi gjort ved at benytte os af diverse teknikker til udvikling af brugergrænseflader. Brugergrænsefladen blev lavet sideløbende i et andet fag. For yderligere informationer om den, se bilag #2.

Vi har testet funktionaliteten af systemet, og alle use cases er realiseret. Netværkskommunikationen til LMON er ikke koblet sammen med den gamle LMON. I stedet er lavet en midlertidig simpel

brugergrænseflade til LMON's netværkskommunikations modul, så det er nemt at simulere at der ligger en patient indlagt ved den.
Under udviklingen af systemet har vi forsøgt at mindske netværkskommunikationen. En CMON kan f. eks kun modtage ECG, EDR og pulse data fra en patient af gangen. Vi har dog forsøgt at lave CMS så skalerbart så muligt, så udvider man f. eks systemet med et stort antal CMON'er, og de alle vil have ECG, EDR og pulse data fra en bestemt patient ved en LMON, kan det få indflydelse på performance. For at forhindre en sådan situation, vil vi enten være nødt til at sætte et krav om hvor mange CMON'er der må være på et netværk, eller et krav om, hvor mange CMON'er der må tilgå en LMON af gangen.

Vi har ikke fra start af sat nogen direkte krav til realtid, da vi savnede redskaber til at afgøre om realtiden blev overholdt. Desuden var det et stort system, der skulle laves på relativ kort tid, og noget skulle skæres væk. Det ville også være endnu svære at teste om realtiden blev overholdt hvis vi skalerede systemet op til noget der var realistisk. Skulle systemet implementeres på et stort sygehus, skulle der være mange LMON'er og CMON'er, og sandsynligvis også mere end en APS. Hvordan belastningen ville se ud hvis det blev skaleret op er svært at vurdere.

Selvom vi ikke havde nogen direkte krav til realtid, har vi alligevel under udviklingen forsøgt gøre koden så effektiv som muligt. Dette har vi gjort ved at forsøge at mindske brugen af oprettelse af objekter og tråde, og i stedet bruge singletons og threadpooling samt mindsket netværkstrafikken.

Vi har test systemet, i et sammenspil mellem Windows computere og en SBC686. Vi har testet, at det er muligt at modtage data fra en LMON, lige som det var muligt at få notifikationer fra APS om ændringer af antallet af patienter. Det var muligt for CMONUI at kontakte CPR Serveren og finde patienternes navne og vise disse for brugeren. Når et system starter, bliver dette også registeret på Log Serveren, dog først efter systemet har kontaktet Configuration Serveren, og modtaget systemets opsætning.

Underskrifter

_____

Frank Henningsen

_____

Claus Kirkegaard Clausen

_____

Jens Peter Troelsen