Datalogisk Institut
Det Naturvidenskabelige Fakultet
Aarhus Universitet

Hovedopgave

Diplom i Informationsteknologi

Linien i Softwarekonstruktion

Spatial Information Sharing

af

15. Juni 2010

Ole Ventzel Davidsen, Studerende

Niels Olof Bouvin, Vejleder

Morten Wegelbye Holm, Studerende

Datalogisk Institut
Aarhus Universitet
Åbogade 34
8200 Århus N

Tlf.: 89425600
Fax: 89425601
E-mail: cs@cs.au.dk
http://www.cs.au.dk/da

Abstrakt

I denne diploma thesis har vi lavet et framework, der kan bruges til informationsdeling ud fra en brugers kontekst(primært lokation). Vi har kigget på forskellige platformes SDK'er og tidligere arbejde inden for frameworks til Context Aware computing, samt kigget på de sensorer, som kan bruges til lokation. Vi har ud fra vores undersøgelser udviklet et framework til Android platformen med en Windows backend, som kan bruges til hurtigt at udvikle context aware applikationer til Android. Vi har via nogle prototyper vist, hvordan man hurtigt kan lave nogle simple applikationer via frameworket og hurtig kan tilføje nye sensor typer. Vi har lavet nogle forsøg med at få vores server til at geocode de lokationer, som ikke er fysisk bestemt ud fra allerede gemte fysiske lokationer, samt lavet et hieraki på informationerne således at små netværk, der ligger inden for store netværk også vil kunne finde informationer koblet til det store netværk selvom sensoren til det store netværk ikke er til stede på søgningstidspunktet. Vi har eksperimenteret med følgende sensor teknologier: GPS, Wifi, Bluetooth, Kompas, GSM CellId og IMEI. Vi har i nuværende iteration ikke taget hensyn til sikkerhed og brugerrettigheder. Hvor sikkerhed dækker over fortrolighed, autentitet og tilgængelighed.

Introduktion

Motivation

En mobiltelefon er i dag ikke længere bare en telefon. Man har i mange år kunnet tage billeder, sende sms beskeder, besøgt WAP hjemmesider og i et begrænset omfang rigtige hjemmesider.

Idag kan man næsten ikke købe en mobiltelefon uden, at den har stor skærm, kamera, 3G, GPS, Wifi, Bluetooth, kompas osv. Mobiltelefoner i dag går mere og mere i retning af at være mobile computere.

Idag stormer de "mobile computere" frem. Microsoft har være på banen i mange år, men deres enheder har ikke været særlig brugervenlige. De er inden for meget kort tid blevet overhalet af Apples Iphone samt Googles Androids, dog ser deres Windows Phone 7 udspil spændende ud. Enkelte producenter har dog selv forsøgt sig med at ligge deres egen UI oven på Microsofts modeller med mere eller mindre succes. Eksempelvis HTC Sense.

I og med at mobiltelefoner i dag er spækket med så mange teknologier, der kan benyttes som sensorer, åbner det for en masse muligheder for at lave kontekst årvågne applikationer, som kan lette vores hverdag.

Mobil computer enhederne er også ved at blive assimileret ind i vores hverdag. Vi har i dag allerede IPads fra Apple og Android konkurrenterne er også lige på trappen, så er der Google TV, som ligger Android ind i dit TV enten via en boks eller direkte indbygget i tv'et. På den nyligt overståede CES messe blev der fremvist demoer af Android enheder indbygget i Mikrobølgeovne, vaskemaskiner, printere og VoIP telefoner. I Kina er de allerede begyndt at indbygge Android i biler.

Det åbner for utallige muligheder for at hente information ud fra brugers/objekterne kontekst.

En brugers kontekst kan f.eks. findes ud fra GPS, WiFi, bluetooth eller Celle ID, opkaldsliste, sms, telefon status, kalender osv.

Information i dag kan også tit sammenkobles med en lokation, men sammenkoblingen er ikke altid åbenlys.

Man kunne godt forestille sig at man lavede en søgning ud på den information der var koblet på ens nuværende spatielle kontekst.

Eksempelvis kunne en virksomhed gemme et kort over bygningen elektronisk i deres reception. Hvor virksomhedens gæster i dette område så ville kunne kalde dennes oplysninger frem.
En person kunne oprette en 'cv-sphere' omkring sig - hvormed folk i hans nærhed kunne trække informationer på dem man er sammen med lige nu, og herigennem hurtigt opbygge relationer mellem folk.
Live Point Of Interest var et andet eksempel, en by kunne dele nogle urls ud fra forskellige steder rundt omkring i byen, så besøgende nemt kunne se, hvad der var værd at se. Evt. sammen med en side, der viste en event kalender for den specifikke dag.
Vi kender også alle den situation, hvor vi skal udveksle information med en person vi fysisk står og snakker med, og enten skal til, at skrive email adresse, messenger id eller lignende ned i stedet for bare at dele informationen og koble den til den fælles kontekst.
Man kunne også vende den om så det ikke er den mobileenhed der er klienten, men infostanderen, vaskemaskinen, fjernsynet, ovnen eller andre enheder, der kunne tænkes at få en android indbygget i fremtiden. Disse enheder ville kunne se, om der dukkede nye enheder op, der måske havde en form for information tilkoblet, der havde relevans for den stationære enhed. F.eks. kunne man rende rundt med sit personlige tvprogram tilknyttet sin telefon eller på anden måde tilknyttet ens person. Og på den måde kunne et tv f.eks. automatisk skifte kanal eller gøre opmærksom på, at der var noget, man måske gerne ville se.

Hypotese

Det må være muligt via de kontekst informationer, som omgiver os at binde informationer i de områder, vi bevæger os.
Det må være muligt at lave et framework, der gør det nemt at lave applikationer til at søge og gemme informationer, der knytter sig til dine omgivelser. Et system hvor man fra en klient vil kunne binde informationer til spatiale kontekster. Og via en klient hente de gemte informationer frem.

Eksisterende løsninger tilbyder programudviklere at finde deres position, eller lave en tilnærmelsesvis lokaltions bestemmelse ud fra de omkringliggende GSM celler, eller WiFi - i dette system vil vi forsøge at abstrahere sammenhængen mellem data og lokaltioner.

Relateret arbejde

Der findes utroligt mange undersøgelser/frameworks som på den ene eller anden måde kan relateres til vores framework. For at kunne lave en fair sammenligning vil vi kort liste, hvad der er vigtig i vores framework, og ud fra den vinkel kigge på de undersøgelser/frameworks der findes. Designmålene vil blive uddybet under design afsnittet.

Design mål

- Kommunikations protokol så let som mulig. (Hold sendetid nede)
- KISS - Klient interface så simpelt som mulig.
- Fokus på informationsdeling. Anden kontekst er pt. ikke interessant og kan løses i applikationerne.
- Selvlærende lokationer.

HyCon[1]

HyCon er et projekt der er opstået på Århus Universitet. Deres use case er baseret på, at give folkeskole elever mulighed for aktivt at lære om dyr og andre ting i den virkelige verden i stedet for at læse det i en bog. Det er også designet til, at eleverne skal kunne registere ting, der hvor de ser dem og så sidenhen finde oplysningerne frem, så de kan bruges til præsentationer/fremlæggelser.

Vi vil beskrive hovedpunkterne i HyCon og sammenligne dem med vores fokus punkter.

Det første HyCon gør er, at få defineret nogle modeller for kontekst, hvilket selvfølgelig også er relevant for vores arbejde. Se under design hvordan vi har defineret vores kontekst.

I HyCon er kontekst delt op i 3 domæner. Fysisk, digital eller konceptuel. Hvor fysisk er vores omgivelser, sted objekter, mennesker, absolut tidspunkt. Den digitale dækker over computer modeller, infrastruktur, protokoller, andre enheder osv. Konceptuelt dækker over brugerens aktivitet, intentioner, fokus og forståelse af omgivelserne.

HyCons arkitektur er opdelt i 4 lag, Storage, Server, Terminal og Sensor. Det innovative dengang HyCon blev lavet var sensor laget. Sensor laget er lavet som en

logisk separation mellem applikationskode og kode til at hente sensor information fra forskellige sensorer.

HyCons applikations del består af 3 dele: Browsing with your feet, Annotating the world og Overview at a Glance.

Browsing with your feet går ud på, at man istedet for at indtaste et søgeord for søge information, så får man informationen ud fra den kontekst man er i. Hvis man så vil finde ny information, skal man skifte kontekst dvs. fysisk flytte sig. Dette er præcist det samme vi gør. Der er ikke beskrevet hvordan der kobles information på et GPS koordinat så det kan fremfindes igen uden at man rammer det helt præcist. Der er dog snak om geocodning og reverse geocodning. Som går ud på, at man har en database med GPS koordinater koblet op på symbolske lokationer. F.eks. lat, long, gadenavn, postnr, by. Hvis du så vil geocode en adresse slår man op i den tabel og finder et GPS koordinat ud fra adressen. Reverse geocodning går den anden vej. Du har et GPS koordinat og vil finde adressen. Her bruger man nok oftest en radius, da man højst sandsynligt ikke vil stå på præcist det samme GPS koordinat som adressen blev oprettet på.

Annotating the world. Dette er oprettelse af de data der kan søges frem i HyCon. Oprettelse og kobling af information til en kontekst. HyCon bruger også begrebet digital grafitti. Og beskriver det som at udlægge digitale spor.

Overview af a glance er beregnet på at give et overbliks billede over de informationer, der er oprettet. Det kunne f.eks. være de spor man selv har efterladt sig i forbindelse med en opgave, men det kan også være andres spor. Dette er beskrevet som "indirekte repræsentionelt navigation" i HyCon, eller som Pascoe kalder "imaginary world" eller "pretend context" og det går ud på, at man specificere en virtuel kontekst og dermed skaber en "Hvad hvis jeg var der?" situation. Sagt med TDD termer så er det muligt at mocke konteksten.

Når vi sammenligner HyCon med vores framework, så har vi samme sensor abstraktion som HyCon. Model mæssigt er vores framework forskelligt fra HyCon, da vores server og storage lag er usynlige for klienten. Hvis vi kigger på vores kontekst arkitektur, så har vi opdelt vores kontekst i to domæner. Et GPS domæne, fordi GPS er speciel på grund af kravet omkring radius. Og et net domæne, som dækker al anden kontekst.

Derudover så er det lidt uklart hvordan HyCon linker informationerne sammen. Vores framework har mulighed for koble en samling af forskellige typer hypermedia sammen med en samling af kontekst. Hvor man i HyCon umiddelbart skal oprette flere informationer koblet på samme kontekst for at opnå det samme.

ContextPhone

Er skrevet af researchere og professor hos Helsinki Universitet i 2005.

Deres design mål er følgende: Tilbyde kontekst som en ressource, indarbejde eksisternede applikationer, tilbyde hurtig interaktion og samtidig være diskret, robust, understrege aktualitet og tilbyde hurtig udvikling.

ContextPhone er et framework til fremstilling af ContextAware programmer til smartphones. Målet for dem er at gøre det nemt for brugeren (programmøren) bruge forskellige Contexts til deres programmer. Herunder også lokaltionen for en bruger, og brugere i nærheden.

ContextPhone opdeler sine sensorer i 4 kategorier Lokation, Bruger interaktion, kommunikation og fysiske omgivelser.

Lokation dækker over GPS og GSM. Bruger interaktioner kan være aktiv appliaktion, er telefonen aktiv eller er den på stand by, hvilken profil har telefonen(Lydløs, i bilen osv), hvor meget strøm er der på batteriet osv. Kommunikation er opkald, opkaldsforsøg, sms beskeder osv. Fysiske omgivelser er bluetooth enheder i nærheden og optiske markøere.

Sammenlignet med SIS frameworket, så er de noget mere detaljerede i deres sensorer. I SIS frameworket har vi endnu ikke haft nogen grund til at splitte vores sensorer mere op end de er nu (GPS og Net), men det kan selvfølgelig komme i en senere iteration.

I forbindelse med frameworket er der også lavet nogle applikationer. ContextLogger som kan gemme Contexten historisk og som efterfølgende kan bruges til at se hvordan en bruger har brugt sin telefon. ContextContacts er en integrations med telefonens kontakt bog, så man kan se ens kontakters status og ContextMedia som bruges til at dele media med andre. ContextPhone kan bruge eksisterende infrastruktur til at notificere andre brugere omkring ændringer f.eks. via Jabber.

En anden ting ved Context Phone er, at de har en service kørende på sensorerne hele tiden. Hvordan det har kunne lade sig gøre i 2005 ved vi ikke, det må skyldtes, at sensorer dengang ikke trak så meget strøm som de gør i dag. Hvilket selvfølgelig heller ikke er tilfældet hvis kontekst sensoreren f.eks. kun skal kigge på, hvad der er i telefonens opkaldsliste.

ContextPhone bærer præg af, at det er skrevet i 2005. Der er mange ting der har ændret sig inden for smartphones de sidste 5 år. Tag f.eks. følgende citat fra teksten.

A phone's most important use isn't to run applications but to communicate with the outside world.

Idag er smartphones på vej til at overtage markedet, og grunden til at smartphones sælger så godt, som de gør, er jo netop applikationerne. Alle de store spillere har lanceret butikker eller app. stores hvor brugerne kan hente/købe programmer. Hardwaren bliver også bedre og bedre med flere muligheder for multitasking.

Context Toolkit[2]

Dey beskriver kontekst som al den information, som kan benyttes til at karakterisere en entitets situation, hvor en entitet kan være en person, et objekt eller et sted som er relevant for interaktionen mellem bruger og applikation, inklusiv bruger og applikationen.

Han beskriver efterfølgende context-aware computing, som et system der bruger kontekst til, at levere relevant information eller ydelser til brugeren hvor relevansen afhænger af brugerens aktivitet.

Context Toolkittet vil gerne hjælpe udviklere med nemmere at komme fra design til implementation ved at tilbyde en arkitektur der er en kombination af funktionalitet og abstraktioner.

Context toolkits sensor lag kalder de for widgets. En widget er også en abstraktion der gør at man ikke skal tænke på hvordan de forskellige sensors virker. Som applikations udvikler er konteksten bare til rådighed. En anden ting der er specielt ved deres widgets er at de ikke hænger på applikationen. Dvs. at deres widgets faktisk er services som flere applikationer kan benytte på samme tid. Widgets gemmer også den kontekst de opdager. Dvs. at en applikation kan hente en historik fra en widget.

En af de ting der helt sikkert har ændret sig siden context toolkitet er kommet frem er sensorerne. Dengang var sensorerne ikke så integreret i enhederne som de er idag. Derfor håndterede toolkittet også at distribuere sensor data transparent til applikationerne, der skulle bruge den. Et eksempel var IR modtagere der kunne føle på IR sendere så systemet kunne se hvor mange mennesker der var i et rum.

De har også en abstraktion for entiteter. Denne kalder de for en Context Aggregator. Denne aggregator står for, at indsamle al context, som er relevant for en entitet.

Hvis vi skal sammenligne med vores eget framework og med de muligheder der findes i dag. Så er der nogle ting der træder frem. Widget begrebet med at have sensorerne kørende, som services der altid er til rådighed vil nok ikke være muligt idag. Det ville tappe telefonerne for strøm med det samme. Der er dog fremtidsmuligheder i det. Under fremtidigt arbejde kigger vi lidt på hvad Intel forsker i, på det felt. Hvis ens applikation kører på en stationær pc er strøm selvfølgelig heller ikke et problem.

Det med at gemme kontekst historik, er heller ikke noget vi har kigget på. Vi har lavet nogle overvejelser omkring at nogle af de informationerne der kobles til en kontekst, kun skal have en begrænset levetid, fordi de simpelthen ikke har noget relevans i x tid, efter de er blevet oprettet.

SDKs til mobil udvikling

I dette afsnit kigger vi på hvad de forskellige SDKs der findes til udvikling af applikationer til mobile enheder tilbyder. Vi kigger på Android, som vi selv har brugt, men derudover vil vi også kigge på hvad der findes til IPhone, Microsofts .Net compact framework, Microsofts Windows Phone 7 SDK og Java ME.

Android

Androids SDK tilbyder klasser til mange af dets sensorer, og de er i mange tilfælde meget nemme at gå til. De er dog ikke lavet over en abstraktion, så man kan ikke direkte bruge dem som kontekst. Wifi og bluetooth klasserne er f.eks. beregnet til at styre/configurer enhederne fra en anden applikation end den indbyggede indstillings applikation. Ud over wifi og bluetooth findes der en lokations klasse som snakker med GPS modtageren eller henter positionen via nærmeste mobilmast eller via et kendt wifi netværks, hvis man kan se et sådant. Der findes en sensor klasse som kan snakke med flere af de andre indbyggede sensorer, hvis hardware understøtter dem. Sensorer der understøttes pt er:

- Accelerometer
- Gyroskop
- Lys
- Magnetfelt
- Orientation
- Tryk
- Nærhed(Proximity)
- Temperatur
- GPS
- Kompas

Så findes der en MediaRecorder klasse, der kan optage video og lyd og en kamera klasse, der kan tage billeder.

Android SDK er ret åbent i alle henseender. SDK'et dækker stort set alt hvad brugeren kan gøre på telefonen, det kan en program udvikler også tilgå. Så hvis vi vil have fat i brugerens SMS beskeder, kalender aftaler, mails osv. så er det også

muligt. Som kontekst platform er Android derfor meget velegnet.

Set i forhold til vores framework så gør Android SDKet det utroligt nemt at tilgå en enheds sensorer, så de kan benyttes til vores framework.

Iphone

Ingen af os har nogen dybere erfaring med Iphone SDK, så nedenstående er baseret på den dokumentation, der findes omkring IPhones SDK på nettet.

IPhones udvikling er generelt et noget mere lukket land end f.eks. Android. Der findes f.eks. ikke et officielt understøttet API til at liste de trådløse access punkter man kan se. Der skal man istedet ud og finde et privat API som f.eks. iPhone-Wireless[3].

Problemet med private API'er er, at ens applikation ikke længere bliver godtkendt i Apples Appstore. Så man vil godt kunne lave en applikation, der har funktionalitet, men man vil ikke kunne få den ud til masserne. Bluetooth er heller ikke tilgængeligt for udviklere, der findes et gamekit, som via en discovery service(bonjour) kan returnere andre enheder som kører samme program som en selv, men den vil også returnere enheder på wifi netværk. Det skulle også være muligt, at bruge bonjour servicen til at se alle de enheder, der er på enten samme netværk eller som kan ses via bluetooth. Men selve bluetooth servicen er ikke tilgængelig og det vil nok også kræve, at det er enheder der understøtter bonjour servicen selv.

I Iphone SDK'et findes der en indkapsling af accelerometer API, som de kalder motion events. Som gør det mere simpelt at opfange forskellige typer af bevægelser. Dette kunne f.eks. være nyttigt ti,l at finde ud af hvad brugere gør lige pt. Er denne gående. Sætter vedkommende sig ned, osv. Der er f.eks. nogle der har brugt denne funktion til, at opfange hvor dybt man sover, ved at man skal placere telefonen på hjørnet af ens madras, når man går i seng. Så kan telefonen opfange hvor meget man bevæger sig i løbet af natten, og hvis man så har sat en alarm til, så kan den vække en mens man alligevel er tæt på at være vågen (hvis man bevæger sig inden for en halvtime af det tidspunkt hvor man skal op).

GPS modtageren er heller ikke direkte tilgængelig. Istedet har man et Location API som indkapsler lokation. Denne indkapsling benytter også CellId og kendte wifi access punkter til at bestemme ens position.

Understøttede sensorer:

- Accelerometer
- Nærhed (Proximity)

- Lokation
- Kompas

Det er muligt, at der findes flere sensorer der kan opnås adgang til. Når man søger på information omkring IPhone SDK'et, så falder man ofte over nogle, der nævner udokumentere API'er. Hvilket nok skal sidestilles med private API'er.

Hvis vi sammenligner IPhone SDK'et med vores framework, så ville det blive en lidt tynd implementation vi ville kunne lave på platformen. Selve SDK'et er ret lukket land, det er f.eks. heller ikke muligt at tilgå mails, sms beskeder, kalender aftaler osv.

Microsoft .Net Compact Framework

Igen er nedenstående baseret på information vi har fundet i den officielle dokumentation eller i diverse community forums, da ingen af os har erfaring med andre sensorer end GPS sensoren på denne platform.

Microsoft Frameworket stiller ikke umiddelbart et API til rådighed for tilgang til diverse sensorere. En af grundene til det er nok, at der findes så mange forskellige producenter af hardware til platformen og at den stammer tilbage fra før det blev almindeligt, at have alle de muligheder i hardwaren, så der ikke fra starten har været mulighed for producenterne at lave/bruge et fælles API. Derfor skal man, hvis man vil snakke med sensorerne på en HTC telefon, have fat i et 3. parts API fra HTC. Skal man snakke sammen med en Intermec PDA skal man have et SDK fra Intermec osv. En af de ting som Microsoft dog har fået lagt i frameworket er et GPS API. (GPS Intermediate Driver eller GPSID).

Det man oftest gør, når man laver en driver til en Sensor er at kode driveren i C++, så pakker man den ind i en DLL. Derefter så laver man en .net klasse der indkapsler kaldene til denne dll, så den nemt kan kaldes fra .net frameworket.

For nemt at tilgå wifi information skal man også have fat i et 3. parts API. OpenNETCF Smart Device Framework.

http://msdn.microsoft.com/en-us/library/aa446491.aspx#build_wifi_discover_app_netcf2_topic2

Det er samme problematik med bluetooth. Her kan man benytte følgende framework

http://32feet.codeplex.com/

Vi har valgt ikke at liste sensorer for .Net Compact frameworket, da selve

frameworket i sig selv ikke understøtter andet end GPS, men lader det være op til hardware producenterne at sørge for, at deres hardware kan bruges på platformen.

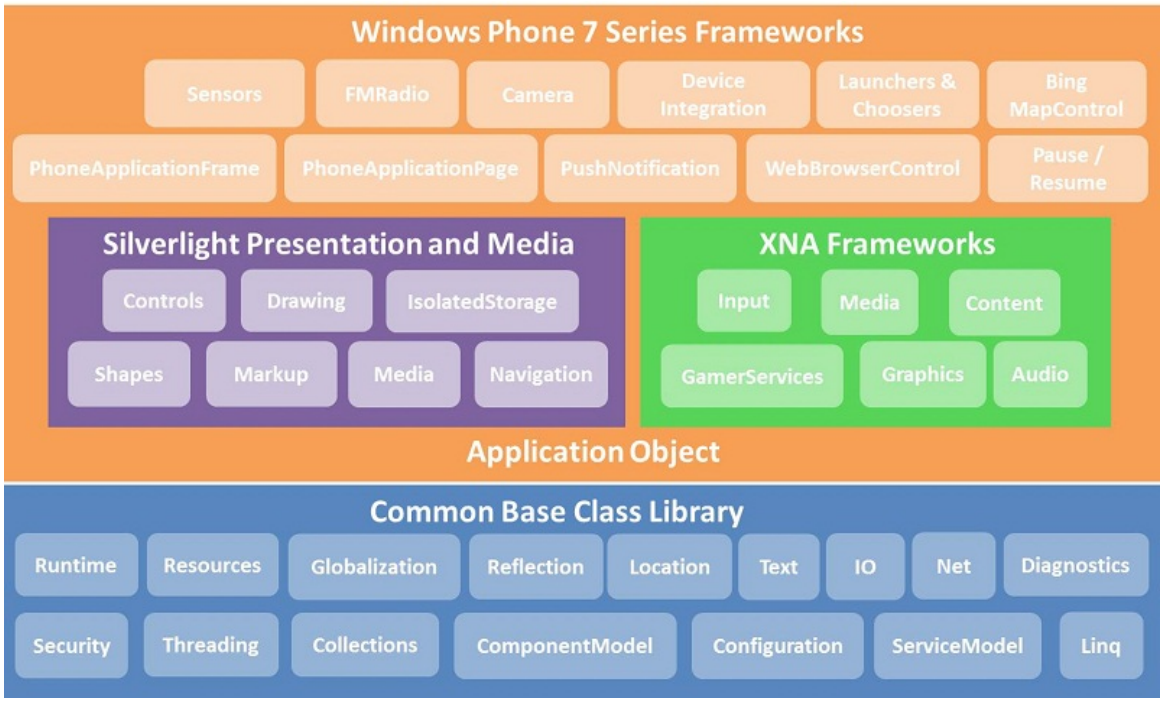
Sammenlignet med vores framework, så vil vi sagtens kunne lave et brugbart framework på platformen, som i sig selv intet tilbyder. Udfordringen ville være at få lavet sensor implementationen således, at frameworket nemt ville understøtte de mange forskellige hardware producenter.

I næste afsnit kigger vi på, om Microsoft så har lært noget, nu de har haft chancen for at starte forfra.

Microsoft Windows Phone 7[4]

Da windows Phone 7 endnu ikke er lanceret så er SDK dokumentation stadig i en beta fase, så der kan være ting, der kan ændre sig når det kommer på markedet.

Hvis vi kigger på deres system oversigt kan vi se, at sensorer er blevet en del af deres framework. En af de nye ting med Phone 7 er også nogle strengere krav til hardware producenterne. Så det alt andet lige bliver nemmere for udviklere, at de undgår at skulle tænke alt for meget på forskellige hardware konfigurationer.



Hvilke sensorer der vil være supporteret fra start er svært at sige endnu. I deres

dokumentation er Accelerometer, Lokation og kamera beskrevet med eksempler.

Der findes ingen dokumentation omkring Wifi eller bluetooth på nuværende tidspunkt. Der er heller ingen information omkring hvor lukket systemet bliver, og om det bliver muligt at tilgå kalenderen, sms og mails osv.

I og med at sensor begrebet er tænkt med i det grundlæggende framework, så ser operativ systemet spændende ud ud fra et context-aware computing synspunkt. Det er dog for tidligt, at sige noget om hvordan det virker i praksis endnu, da det kun er muligt at lege med en beta i en emulator på nuværende tidspunkt. Vores framework ville sagtens kunne bruges på denne platform og hvis man lavede det rigtigt ville man kunne dele meget med den klassiske .net compact framework platform.

Java ME

Java Micro Edition (j2me), Suns udgave af java til ressource svage enheder. Oprindeligt under JSR-68 - bredt deployed på 2.1 milliarder enheder. (www.java.com/en/about/) Spredt over sattop bokse, embedded computere og ikke mindst mobiltelefoner. Videre udvikling på platformen i projektet PhoneMe (https://phoneme.dev.java.net/).

Jsr-68 historieske rødder tilbage til år 2000.

I JavaME har man et grundlæggende API, med en række valgfrie udvidelser. (JSR'er, MIDP og CLDC). Man er således afhængig af om den pakke man har brug for er tilgængelig på din platform.

Bluetooth er godt understøttet gennem JSR 82, og integreret i Sun wireless toolkit for j2me.

GPS er understøttet i JSR 179.

GSM Celler er ikke understøttet i standarden, nogen leverandører har lavet deres egen metode.
En hurtig søgning på Nokias udvikler forum afsløre at det ikke er uden problemer at bruge den slags.
Det skal dog nævnes at JSR 179 er et lokation API - og en leverandør kan have lavet et system der gør at klienten kan bestemme sin lokaltion ud fra synlige celler.

WiFi er ikke rigtigt understøttet gennem noget standard API, der er enkelte telefon leverandør som har lavet noget proprietært understøttelse.

Efter vores bedste overbevisning ligger det i j2me´s natur at abstrahere alt det hardwarenære information. Vi er bange for, at vi kommer til at bruge alt for meget tid på enten leverandørers specifikke ting, eller ikke understøttede del api'er.

Til J2ME fordel er klart den voldsomme mængde enheder der understøtter en eller anden grad af J2ME, men eksempelvis Nokia 3410 fra 2001 er en af dem.

Opsumering på SDKer og sammenlignings tabel

For alle SDK'erne gælder det, at ingen på nuværende tidspunkt kan tilbyde et framework/API som kan løse de udfordringer der ligger i et Context-aware framework, både generelt men også med henblik på informations deling. Nogle af SDK'erne er mere klar på det end andre og andre er så lukkede at det næsten vil være umuligt at lave uden at producenter selv tilføjer funktionalitet i SDK'erne. Heldigvis er der ingen af producenterne der er dumme og det virker også som om de i en vist udstrækning lytter til forbrugerne, hvad angår ny funktionalitet i de nye versioner der bliver frigivet efterhånden.

	Lightweight kommunikation	KISS (Implementering)	Smalt fokus	Selvlærende lokationer
SIS	x	x	x	x
HyCon	n/a	n/a		
Context Phone	x	x		
Context Toolkit				

n/a indikere punkter, hvor litteraturen ikke er præcis på området, hvorfor vi ikke kan være sikre på et svar.

Valg af platform

Af de forskellige SDK'er til udvikling på smartphones vi har kigget på, virker Android som den bedste platform. Åben standard og veldokumenteret api. Vælger man den, vil man i fremtiden ikke bare ramme mobiltelefoner, men vil også kunne ramme husholdningsmaskiner som tv, vaskemaskiner, ovne osv.

Analyse og design

Design mål

I det følgende afsnit vil vi argumentere for vores design mål.

Kommunikations protokol så let som mulig:
Selvom hastigheden på mobil netværk stadig bliver hurtigere og hurtigere. Så er det stadig ikke lige hurtigt alle steder. Det er heller ikke gratis at overfører data over mobilenetværk.

Vores andet design mål var KISS - Klient interface så simpelt som mulig:
Det skal være nemt, simpelt og hurtigt at bruge vores framework af en klient. Det skal ikke være sådan, at man skal igennem mange siders dokumentation, eller ind og kigge i framework koden.

Vores trejde design mål er en indsnævring af fokus. Vi fokusere kun på informations deling. Byggestenen i alle kontekst framework er information omkring hvilken kontekst man er i. Vores fokus er derfor at kunne håndtere information. Andre kontekst framework har også metoder til, at fortælle hvilke enheder der er i nærheden eller anden mere applikations specifik funktionalitet. Ved at kunne levere konteksten som information vil man som applikations udvikler kunne løse problemet med f.eks. enheder i nærheden i selve applikationen. Det kunne også være, at man ville ligge en sådan fuktionalitet i SIS frameworket i en senere iteration.

Vores fjerde design mål er: selvlærende lokationer. Hvilket går ud på, at sensor data, som ikke er GPS baseret, vil kunne stedbestemmes hvis det findes i nærheden af et andet sensor data, der er blevet GPS bestemt. Se afsnittet omkring geocodning af net for yderligere detaljer.

Framework

Der er flere fordele ved at lave/benytte et framework, f.eks. kodegenbrug. Vi programmører er per definition dovne(de fleste af os ihvertfald). Vi vil altid prøve at lave tingene så simple og gennemskuelige som muligt og så hurtigt som muligt. En programmør vil meget hellere bruge sin tid på program features/krav end på de standard lav niveaus detataljer der skal til for at få programmet til at køre. Og her kommer frameworks ind i billedet.

Et andet begreb der skal defineres når man snakker frameworks er library. Et library er en samling af kode. Kan være en række funktioner, men kan også være en samling af klasser, der frit kan plukkes fra. Et eksempel kunne være et bibliotek der indeholdte en samling af funktioner til streng håndtering.

Inden vi kommer ind på forskellen på et library og et framework, skal vi lige have defineret et andet begreb, Inversion Of Control.

Inversion of Control er ifølge Martin Fowler[5] et af de definerede karakteristika for et framework. Hans eksempel er UI i en konsol vs. et windows program. I konsollen har man en writeline("Input navn"); Efterfulgt af en readline der læser brugerens input. Her har programmøreren 100% kontrol over programmet. Modsat Windows UI, hvor brugeren har mere frie hænder. Programmøren ved ikke hvornår brugeren udfylder teksten boksen med sit navn. Han kan abonnere på et event på teksteboksen og så kalder UI frameworket programmørens kode når brugeren har indtastet sit navn. Det vil sige at det er frameworket der styrer flow eller sagt på en anden måde har kontrollen. Der er også nogle der bruger begrebet "Hollywood's Law: Don't call us, we'll call you".

Hvis man søger på Inversion of control, vil man støde på folk der sætter lig med mellem Inversion of Control og Dependency injection, men det er en fejl. Dependency injection kommer fra Inversion of Control Containers, og et eller andet sted er der opstået nogle misforståelser der.

Tilbage til forskellen mellem et library og et framework. Når man benytter et library så kalder man en funktion, den udfører et stykke arbejde og returnere måske et resultat. Ved et framework er det anderledes jvf. Inversion of Control, det er ofte frameworket der kalder klientens kode. Det vil sige, at frameworket har en arkitektur, der styrer klient programmets opførsel.

Indenfor frameworks snakker man normalt om 2 forskellige typer af frameworks. Blackbox og Whitebox.

Klassifikationen af et framework inden for de 2 typer er ofte givet ud fra hvordan en programmør bruger/tilpasser frameworket.

Whitebox frameworks er karakteriseret ud fra, at man bruger nedarvning af abstrakte klasser eller implementere interfaces og returnere disse instanser til framworket for at tilpasse frameworket.

Blackbox frameworks er karakteriseret ud fra, at man bruger komposition af forud implementerede instanser.

I praksis vil det dog oftest være et mix. I vores framework benytter vi hovedsageligt blackbox, men hvis en bruger f.eks. vil tilføje en ny sensor skal brugeren implementere et sensor interface og derefter tilføje den nye sensor via komposition.

Man kan sige at, ved whitebox skal man kende meget til den eksisterende kode i detaljer, hvor man ved blackbox skal vide, hvad der sker ved forskellige sammensætninger af delene i frameworket.

Sensorer

I dette afsnit vil vi beskrive de forskellige sensor typer, der kan benyttes samt deres bagvedliggende teknologi.

Trådløse netværk [6] [7]
Der findes flere forskellige typer af trådløse netværk. Man kan dele dem ind i 5 kategorier.

Wireless Personal Area Network eller WPAN dækker over netværk med relativ kort rækkevidde. Et eksempel på et WPAN kunne være bluetooth eller ZigBee (Bruges oftes til automatisering af huse, trådløse stikkontakter, termostater osv.)

Wireless Large Area Network eller WLAN i mange tilfælde kaldet WIFI. Er nok den type trådløst netværk, som er mest brugt. Det er det vi bruger til vores trådløse computere, til vores mobiltelefoner når vi er derhjemme eller på arbejde og til alle spille konsollerne. Der findes den mest almindelige, som de fleste mennesker benytter med et eller flere Access punkter, som agerer base stationer for de trådløse enheder. Og så findes der dem, der benytter Line of sight, altså 2 enheder, der fysisk kan se hinanden som regel over længere afstand.

Wireless Metropolitan Area Network eller WMAN er ikke så brugt. Et eksempel på dette er f.eks. WIMAX, som aldrig rigtig er slået igennem i Danmark.

Wireless Wide Area Network bruges oftest til store udendørs tit offentlige netværk. Består af en samling af access punkter samt trådløse relays der kan øge rækkevidden. Disse enheder kan f.eks. trække strøm fra solpaneler så det er muligt at lave trådløse netværk på steder hvor der ikke er direkte adgang til strøm.

Globale System for Mobile Communications (GSM) er den sidste type trådløse netværk. Det er også meget relevant set fra et context-aware computing synspunkt, da de fleste masters placering idag er kendt, så de kan også bruges til lokations bestemelse.

Et trådløst netværk kan bestå af flere access punkter. Selvom SSID'et vil være ens for hele det trådløse netværk, så vil hvert access punkt have sin egen unikke MAC adresse. Derfor vil man også kunne lave stedbestemmelse inden for det samme trådløse netværk.

Fælles for alle trådløse netværk er, at de kan bruges til stedbestemmelse med en lille undtagelse for bluetooth enheder der er mobile. Det kommer vi nærmere ind på under fremtidigt arbejde. Normalt sættes der et access punkt op, og så bliver det siddende der indtil det enten går i stykker, eller bliver udskiftet, eller hvis det er hos en privat person, så kan det være man flytter. Fordi access punkterne er så statiske som de er, kan man koble informationer sammen med deres id'er og derved har man koblet information på lokationen.

Bluetooth
Som beskrevet tidligere så er bluetooth også et trådløst netværk et såkald WPAN, men det adskiller sig alligevel så meget fra almindelige trådløse netværk, at vi har valgt gå lidt mere i detaljer med den. Bluetooth enheder findes typisk i mobile enheder: PDA, mobiltelefoner, bærbarecomputere, headsets, dongler til stationære, mus, tastaturer, spil kontrollere osv.

Der findes flere klasser af bluetooth enheder. Klassen siger noget om rækkevidde og strømforbrug. Klasse 1 har den største rækkevidde, men brugere også mest strøm. Klasse 1: 100 meter, klasse 2: 10 meter, klasse 3: 1 meter. Udover klasser er der også forskellige versioner. Jo højere version desto højere datatransmissions hastighed og mere funktionalitet.

Hastighed og funktionalitet betyder dog ikke noget for vores framework. Det vi er interreseret i er navne og MAC adresser på bluetooth enhederne.

Under prototyping har vi beskrevet nogle af de erfaringer vi gjorde med bluetooth i praksis.

GPS[8]
Den mest præcise offentligt tilgængelige fysiske lokaltions sensor type. Består af et antal satelitter 24 oprindeligt, men der er planer om at udvide det til 33, som hele tiden sender et signal ned til jorden. En modtager kan så ud fra de satelitter den kan se og den tid som det tager signalerne om at komme ned til modtageren regne ud hvor på jorden den befinder sig. Det kaldes lateration. Modsat hvad mange tror, så skal en modtager kunne se 4 satelitter for at få en præcis position. GPS blev oprindeligt oprettet af det amerikanske militær under navnet NAVSTAR, men er så senere blevet åbnet for private. Militæret har dog mulighed for, at få mere præcise

målinger med deres modtagere end private er. En GPS position er en fysisk position, hvor en adresse er en symbolsk position. Alle navigations systemer har i dag en database, der kan konvertere fra symbolsk til fysisk.

Optiske markører og billede genkendelse
Alle smartphones har idag et eller flere kameraer indbygget. Så snart man har et kamera indbygget kan man via software aflæse stregkoder. Især 2d stregkoder er oplagte til identifikation, idet de kan indeholde både tal og tekst. I mange tilfælde er det en url. Det er f.eks. brugt under festugen i Århus, hvor man på forskellige lokationer kunne skanner en 2d stregkode, som så åbnede en url med information omkring stedet man stod på.

Lokation kan også være givet ud fra billede genkendelse. Man kunne forestille sig, at man tog et billede af Eifeltårnet og ud fra billedet fik man informationer, der var koblet til de objekter der var i billedet. Google er allerede godt på vej med en løsning, der kan det. Google Googles, som er en android applikation, der søges i googles database ud fra billeder. Det kan være billeder af bøger, bygninger, tekster osv. Man kunne f.eks. forestille sig en sensor, der slog en adresse op i en database ud fra billede genkendelse og ud fra adressen fandt GPS koordinatet, som så kunne bruges til opslag i vores system.

Sikkerhed og rettigheder

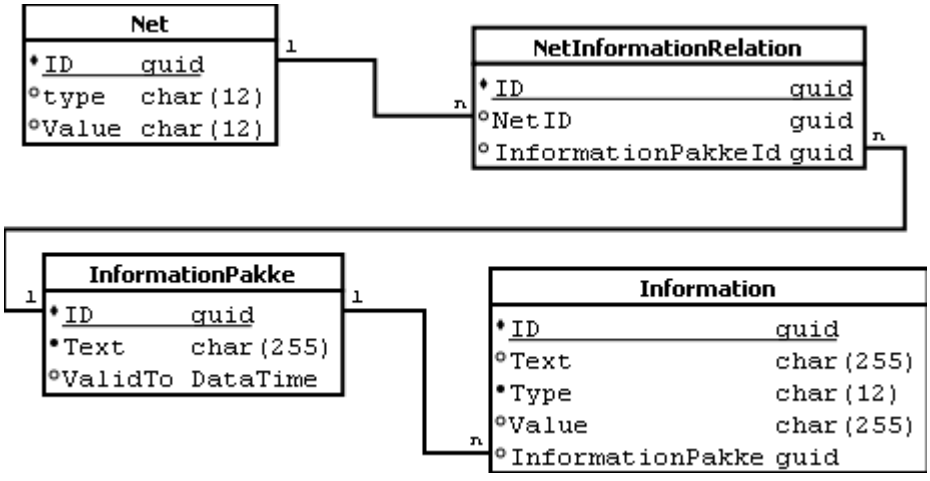
Vi har ikke implementeret hverken sikkerhed eller rettighedsstyring i den nuværende iteration af vores framework, men det er helt sikkert noget, der skal overvejes, hvis frameworket skal kunne bruges på en fælles database. Se under fremtidigt arbejde for uddybelse af problematikken.

Datastruktur / Database model

I vores datamodel har vi to typer. *Net* som er et resultat af en sensor. Og en *Informationspakke* som er det resultat, vi gerne vil kunne sende tilbage til klienten.

Net tabellen er relateret til de informationer, der kan findes via de sensore vi skal have i vores klients kode. Vi har diskuteret om vi skulle oprette en relation per sensor type, således der kunne være en relation WiFiNets{MAC, SSID} access-points der er fundet, en anden relation GPSPos{ lat, lng, distance } til GPS positioner.
Da vi ønsker en arkitektur, hvor vi kan udvide sensor begrædet, vil det ikke være optimalt med denne løsning, da man så ved udvidelser ville være nød til at oprette en ny relation.

En Informationspakke er knyttet mod et eller flere Net. Og har en eller flere Informationer knyttet til sig.
Det giver i udgangspunkt følgende datastruktur i en relationel database.



InformationsPakke og Information er relativt simpel. Det skal dog understreges at *Information.Type* er klientens domæne. Dvs. klient systemet skal selv bestemme sig for hvordan de enkelte informationer skal differencers.
Net er en abstraktion over de oplysninger, der er fundet.

Der er et specialt tilfælde af sensors som er dårligt dækket i vores datamodel. Modellen ligger op til at et resultat fra en sensor skal kunne mappes, ud fra en eksakt værdi, til en post i tabellen *Net* - for GPS vil det ikke være tilfældet.
Hvis vi opretter en *InformationsPakke* relateret til en længegrad/bredegrad, formatteret ind i feltet value (eks. "56.479013/10.042577") skulle en klient der søger informationen være på et ret præcist punkt, og man ville være afhængig af, at de enkelte GPS modtagere leverede resultatet med samme mængde decimaller. Læs mere om problemstillingen under Prototype erfaringer og fremtidigt arbejde.

Vi ser tre løsninger på problemet.

1 Vi kunne ignorere problemet, og lade den enkelte klient omgå det efter behov. Det ville medføre, at vi opgive lidt omkring frameworkets virkeområde. (At abstrahere knytningen mellem lokaltion og data).
2 Have en konvention omkring antallet af decimaller. Et antal der ville være så tilpas lille, at et enkelt punkt ville udgøre et passende areal. Det ville være en behændig løsning, som ville være med til at sikre en meget simpel datamodel. Men det ville betyde at en informationspakke der skal bindes mod et område der er større end det som vi tilfældigvis har valgt, skal have mere end en tuple i *Net* relationen.
3 Eller også må vi tænke modellen lidt om, og gøre GPS til et specialt tilfælde. Hvilket giver den største fleksibilitet.

Løsning 1 er udelukket, da det ville implicere klienten i frameworkets virkeområde, hvilket strider mod princippet om, at det er frameworket, der skal styre flowet.
Nummer 2 vil blive noget underligt noget, hvis man ville dække et stort område, skulle man til at oprette mange Nets. Samt enheden ville være forskellig afhængigt af hvor man befinder sig i forhold til ækvator.
Vi vælger nummer 3 - da den giver størst fleksibilitet. Hvis klienten får behov for at gemme informationer helt ned på enkelte meter, kan det blive understøttet. Polygoner er heller ikke noget problem. Ulempen er så, at vi skal special håndtere GPS.

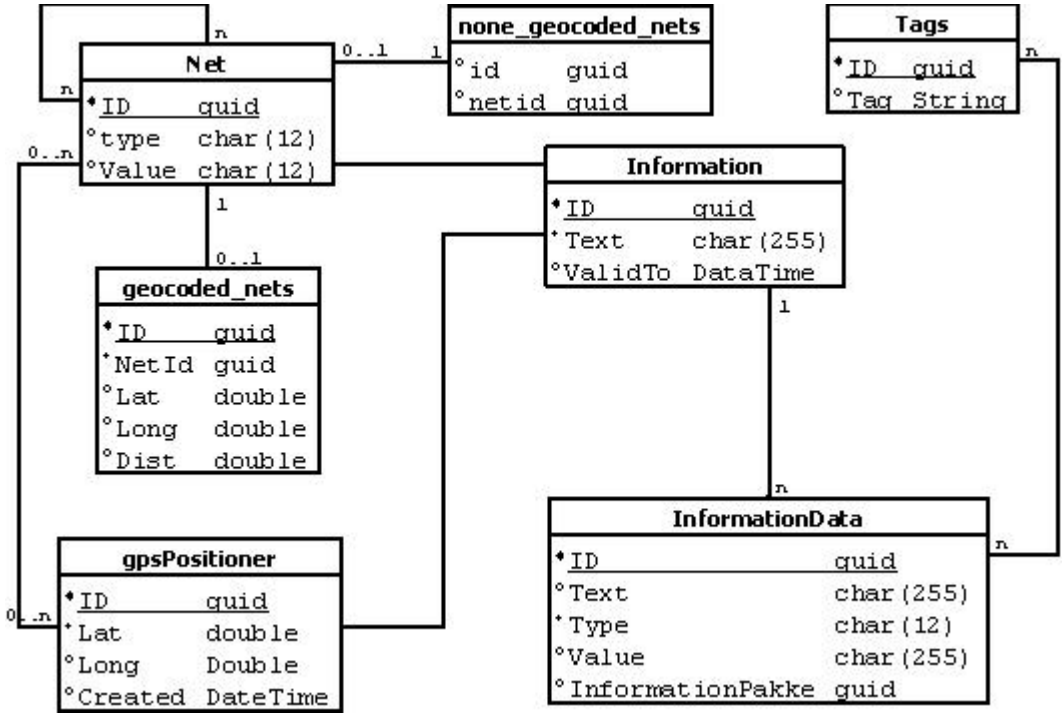
Vi ønsker at være i stand til, at være transparent i forhold til tilgængelige sensors. Eksempelvis hvis en information er knyttet med et CellId, så vil den automatisk også være tilgængelig på de hotspots der er i området. Derfor skal vi have en relation fra net til net.

Vi ønsker altså at lave et hierarki af sensors, således at hvis en enhed skanner et område, hvor der er en information knyttet til en nettype, men ikke har denne sensor type der skal til for, at finde den type net, men har en sensor aktiv med en bedre præcision end den nettype informationerne er relateret til, vil enheden kunne se disse. Mere om algoritmen til dette under implementeringen af serveren.

Men alt dette betyder, at vi skal have en relation der beskriver hvor et net er placeret.

Man kan godt forestille sig, at et område kunne indenholde mange informationer, hvilket kan betyde store mængder data. Derfor introducerer vi at InformationsPakker kan tilknyttes et eller flere tags. Når klienten laver en søgning, vil denne kunne medsende et tag, som så vil blive brugt til at filtrere resultatet.

Vi er endt med en database model som følgende (0...n til 0...n relationstabeller er undladet for læsevenlighedens skyld)



Generelt omkring udviklingen

Metode

SiS frameworket er blevet til gennem en iterativ process. Vi har opstillet nogle use cases, som vi gerne ville kunne løse via frameworket og har på den måde arbejdet baglæns ind i frameworket for, at finde ud af hvilke funktioner, den skulle udstille for at opfylde de opstillede use cases. Se afsnittet omkring brug af frameworket for eksempler herpå.

Vi har baseret vores klient del på Android platformen og vores server på Windows platformen, mere om det i de følgende afsnit.

Til klient udvikling har vi brugt Eclipse og til server siden har vi brugt Visual Studio. Vores database er baseret på en Microsoft SQL 2008 (Express under udvikling og Standard til vores test server). Vi har benyttet Subversion til både klient og server

koden. Vores repository kan findes her: http://code.google.com/p/abcmobil/source/list

Til UML har vi brugt VioletUML[9], ER diagrammer er lavet ved hjælp af Dia[10] og til de rige billedere har vi brugt Microsoft Visio.

Design og implementering af serveren

Serveren er udviklet til Windows platformen. Vi har benyttet C#, WCF, EF og LINQ to EF. I det følgende afsnit vil vi beskrive server teknologien, begrunde valg af frameworks og protokoller samt beskrive hvordan vi har implementeret den.

WCF

Windows Communication Foundation eller WCF som det normalt kaldes. Er Microsofts svar på SOA. Det er en videre udvikling af webservices på en meget fleksibel måde. En WCF service kan f.eks. hostes på mange forskellige måder. Man kan hoste den i en IIS[11] , eller den kan hostes af det program der kører den. Det kan være alt fra en Windows Service til et almindeligt rigt windows program til en konsol applikation. Kommunikationen med klienter sker via såkaldte endpoints, som kan være af forskellige typer. Det kan være standard XML baseret SOAP webservices, men det kan også være binært eller i vores tilfælde JSON via REST. Endpoints kan konfigureres via config filer, så de kan ændres uden at servicen skal kodes om. Det er dog også muligt at specificere nogle attributter på ens service funktioner således, at de altid benytter en given type. Det har vi f.eks. brugt til at sige, at vi altid vil returnere et JSON objekt. Grunden til at vi har gjort det er, at med REST kald så er WCF så smart, at hvis et request kommer formateret som XML så vil WCF formatere response som XML, kommer det som JSON vil response være JSON. Problemet opstod i forbindelse med et REST kald uden request body, hvor WCF servicen returnerede XML til os, og det var vores Android klient ikke klar på.

Entity Framework

Entity Frameworket er en af Microsoft ORM[12] teknologier. En ORM er kort fortalt en abstraktion af ens database over i nogle modeller, som de forskellige programmeringssprog kan forstå. Kort fortalt betyder det at har man en tabel, der hedder kunder, så vil ORMen genere en klasse der hedder kunder, som vil indeholde alle de felter, som er i tabellen. Dvs. at ORMen generere en objekt model ud fra databasen skema. Det betyder, at man ikke længere skal skrive SQL sætninger, hvilket igen betyder at compileren vil fange, hvis man laver syntax fejl i sin database kode. ORMen håndtere også relationer mellem tabeller. Så hvis du f.eks. har en kunde og en ordre tabel, så vil du på kunde objektet have en reference til en liste af alle ordrer relateret til kunden. I en en-til mange relation vil kunde klassen have et array af ordre klassen.

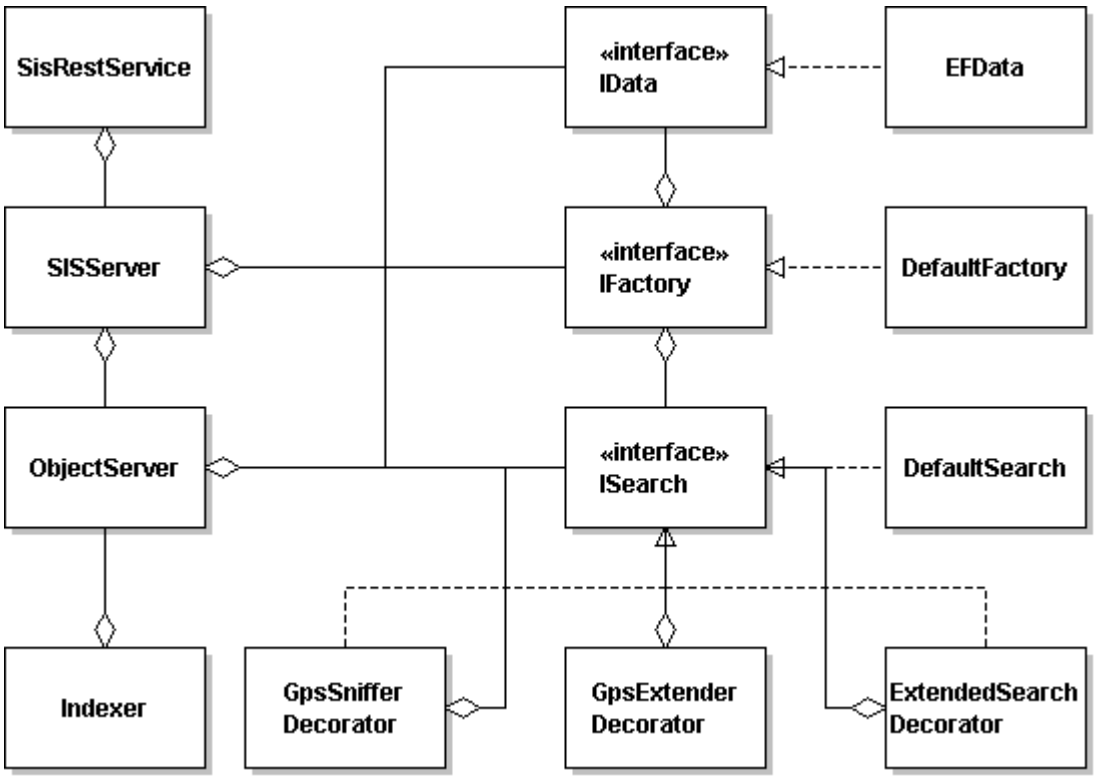
Entity Frameworket består af en grafisk designer, hvori man har to muligheder for at genere sin objekt model. Enten kan man designe objekt modellen først og så genere en database ud fra modellen eller også kan man genere en model ud fra en eksisterende database. Vi gjorde det sidste, da ingen af os har erfaringer med at genere modellen først. Når modellen er genereret, vil man kunne se alle ens tabeller/ objekter samt relationerne imellem dem. Nu har man mulighed for at omdøbe tabeller, felter og relationer. Man kan også slette felter hvis man ikke skal bruge alle felter fra en tabel.

Normalt ville man være færdig her, og når man gemte modellen ville ens objekt klasser blive genereret, men da vi gerne vil kunne styre de felter fra vores objekter der skal sendes via vores REST klient, så benytter vi en ny ting i version 4 af EF. Muligheden for at genere POCO[13] klasser istedet for de normale EF framework klasser. POCO klasser er meget simple objekt klasser, som man selv ville lave det, hvis man f.eks. skulle lave en kunde klasse. POCO objekterne bliver genereret via T4[14] templates, som vi redigerer og hvori vi tilføjer nogle WCF attributter, så alle vores relationer ikke vil blive sendt, når vi overfører en klasse via REST servicen.

Nu har vi et backend setup, der gør, at vi lynhurtigt ville kunne tilføje et nyt felt til en tabel i database og få den med over i vores klienter. Det eneste vi skal gøre er at opdatere EF modellen, og gemme, så vil den automatisk generere vores POCO klasser og dermed vil det nye felt være tilføjet.

Klasse diagram

Vi har med vilje undladt POCO klasserne i nedestående diagram.



Server er opbygget af en WCF REST service (Klassen *SisRestService*) som er en facade til den reelle SIS server(Klassen *SISServer*). Det er lavet sådan i tilfælde af, at man skulle ønske at udskifte WCF delen af server projektet. Så har vi et factory interface(*IFactory*) som opretter vores data interface (*IData*) samt vores søge interface (*ISearch*). På selve server objektet bruger vi en objectServer (*ObjectServer*) til at holde styr på de 2 instanser af henholdsvis *IData* og *ISearch* fra factoryen.
På dataadgang siden er der lavet en Entity framework implementation.
Søgningssiden er der lavet en Default søgnings implementering. Det er muligt at ændre serverens funktion ved hjælp af klasserne *GpsSnifferDecorator*, *GpsExtenderDecorator* og *ExtendedSearchDecorator* - alle tre lavet som Decorator[15] , således at de dynamisk kan tilføjes, mere om deres funktion dem lidt senere.
Indexer er klassen der er i stand til at GeoCode nets, den vil også blevet beskrevet senere.

Vores WCF facade har følgende funktions kald til rådighed for klienterne:

```
private SISServer _server;

public SisRestService()
{
    _server = new SISServer(new DefaultFactory());
```

```
}
```

Vores constructor, hvor man kan se vores server blive oprette med en default factory.

```
[WebInvoke(UriTemplate = "Information/create", Method = "POST",
ResponseFormat=WebMessageFormat.Json)]
  public Model.RestInformation CreateInformation(Model.RestInformation information)
  {
    return _server.CreateInformation(information);
  }
```

Opret RestInformation funktion. Før selve funktionen kan man se vores WCF attributter. Den første(UriTemplate = ...) angiver hvilken REST Url som funktionen skal bruges til. Den næste(Method = ...) angiver i hvilken HTTP metode vi forventer at request kommer i, i det her tilfælde POST. Den sidste(ResponseFormat = ...) tvinger responseformattet til at være JSON.

RestInformations klassen er en adaptor[16] klasse til vores POCO objekter, for at kunne styre relaterede informationer der skal sendes til klienterne.

```
  public class RestInformation
  {
    public Information Information { get; set; }
    public List<RestInformationsData> InformationsData { get; set; }
    public List<Lokation> Lokationer { get; set; }
  }
```

Hvis vi ikke gjorde det på denne måde, ville rest snitfladen lave en deepcopy på alt relateret data og dette vil blive sendt til klienten. Da POCO objekterne har rekursive relaterede data i alle klasserne. F.eks. skal man kunne finde Net objekter fra informations objekter, men også omvendt.

RestInformationsData er også en adaptor, da den relaterer til Tags. Princippet der den samme så vi vil ikke vise den her.

CreateInformation funktionen returnere den Informations pakke der er blevet oprettet, med id'er udfyldt.

```
   [WebInvoke(UriTemplate = "Information/search/{filter}", Method = "POST", ResponseFormat
= WebMessageFormat.Json)]
  public List<Model.RestInformation> SearchInformation(List<Model.Lokation> locations,      string
filter)
  {
```

```
    return _server.GetInformationPackagesFromLocationSearch(locations);
  }
```

SearchInformation søger i informationen ud fra en række lokationer. Dvs. klienten sender den kontekst med som den kan se og så vil serveren returnerer de informationer der matcher. Hvis der bliver søgt med en lokation, som er udkendt af systemet, vil den blive oprettet og koblet sammen med de andre lokationer der søges med. Så selvom der ikke er oprettet nogen informationer i systemet, så vil lokations databasen blive udbygget så længe der søges.
Filter parameteren er endnu ikke implementeret, men meningen med den er, at man skal kunne angive, at man kun vil søge efter informationer af en bestemt type, eller information der har en bestemt tag tilknyttet dets data. DefaultSearch er beskrevet længere nede.

```
  [WebGet(UriTemplate = "Information/all", ResponseFormat = WebMessageFormat.Json)]
  public List<Model.RestInformation> GetAllInformations()
  {
    return _server.GetInformations();
  }
```

Vores DefaultSearchs søge funktion ser ud som følger:

```
public List<RestInformation> DoSearch(List< Lokation > locations)
{
            //Inden vi søger gemmer vi alle informationerne. Så får vi måske også udbygget vores
 relationer på den nuværende søgning.
  _data.SaveAndRelateLocations(locations);
  //Søg efter informationer der matcher vores lokationer.
  var information =  _data.GetInformationForLocations(locations);
  var restinfoList = new List<RestInformation>();

  //map data fra information til restinformation
  foreach (var info in information)
  {
    var restInfo = new RestInformation();
    restInfo.Information = info;
    //map informations data til rest klassen.
    restInfo.InformationsData = new List<RestInformationsData>();
    foreach ( var idata in info.InformationsData )
      restInfo.InformationsData.Add( new RestInformationsData() {
                        InformationsData = idata,
                        Tags = idata.Tags.ToList() } );          //Map gps og net lokationer over i
en search lokations struktur.
    var searchLokationer = new List<Model.Lokation>();
```

```
        foreach (var gps in info.gpsPositioner)
        {
            searchLokationer.Add( new Model.Lokation() {
                        Type = "GPS",Value = gps.Lat + "/" + gps.Long
                        } );
        }

        foreach (var net in info.net)
        {
            searchLokationer.Add(new Model.Lokation() {
                                Type = net.Type,
                                Value = net.Value } );
        }

        restInfo.Lokationer = searchLokationer;
        restinfoList.Add( restInfo );
    }
    return restinfoList;
}
}
```

En af de ting, der er værd at bide mærke i er, at vores kontekst er delt op i 2 tabeller net og gps, men at klienten kun kender en der hedder lokation. Derfor fletter vi resultatet sammen når vi returnere et resultat.

Her er vores GetInformationsForLocations fra vores EFdata klasse. Et eksempel på at hente eksisterende nets findes længere nede.

```
    public List<Information> GetInformationForLocations(List<Lokation> locations)
    {
        //Løb søge lokationerne igennem og lav en liste over gps + net lokationer.
        var gpspositioner = new List<gpsPositioner>();
        var netpositioner = new List<net>();
              var information = new List<Information>();

        foreach (var location in locations)
        {
            if ( location.isGps() )
            {
                var g = GetExistingGpsIncludingRadius(SearchToGps(location));
                if ( g != null )
                    gpspositioner.Add( g );
                continue;
            }
```

```
            netpositioner.Add(
                GetExistingNet(
                    lokationToNet( location ) ) );

        }

        foreach (var positioner in gpspositioner)
        {
            foreach (var i in positioner.Information)
            {
                if (!information.Contains(i))
                {
                    information.Add(i);
                }
            }
        }

        foreach ( var positioner in netpositioner )
        {
            foreach (var i in positioner.Information)
            {
                if (!information.Contains(i))
                {
                    information.Add(i);
                }
            }
        }
        return information;
    }
```

Vi benytter relationer på entity frameworket til, at søge vores informationer frem. Det vil sige, at vi søger aldrig direkte efter information. Vi søger kun efter kendte lokationer og som det kan ses i vores existingNet funktion herunder, så inkludere vi relationer i Linq søgningen. Det er meget simpelt og det virker.

```
    private net GetExistingNet(net netData)
    {
        return _data
                    .net
                    .Include("net1")
                    .Include("net2")
                    .Include("gpsPositioner")
                    .Include( "Information.InformationsData.Tags" )
                    .Where( n => n.Type == netData.Type & n.Value == netData.Value )
                    .FirstOrDefault();
```

```
}
```

Til sidst er vores factory interface og implementering vist. Det ses ret tydeligt, at det vil være ret nemt at udskifte data eller søge funktionaliteten på vores server, ved at implementere et nyt interface og udskifte factoryen i WCF rest facaden.

Interface

```
public interface IFactory
{
    IData GetData();
    ISearch GetSearcher(IData data);
}
```

Implementering

```
public class DefaultFactory : IFactory
{
    public Model.IData GetData()
    {
        return new EfData();
    }

    public ISearch GetSearcher(IData data)
    {
        return new DefaultSearch(data);
    }
}
```

Selvlærende nets

GeoCoding af nets

Vi ønsker, at lave et system hvor en enhed kan stedbestemmes ud fra de trådløse sendere enheden kan se omkring sig. Fremgangsmåden er at vi gemmer alle de søgninger, der er lavet i systemet, sammenkæder de indrapporterede GPS positioner med de indrapporterede GSM Celler og WiFi access punkter med hinanden.
Herefter vil en klient ved hjælp af identifikationen på nogle GSM celler og/eller WiFi punkter kunne raffinere en netværks lokaltion til spatial lokaltion.

Når en klient udfører en søgning, vil den sende alle de informationer, som de aktive sensors kan se. Hvis der blandt disse resultater er et resultat fra en GPS modtager,

har vi basis for at sige noget om hvor de andre opfangede resultater befinder sig.

Undersystemet til GeoCoding af nets ligger altså sig ind foran den aktive *ISearch*, vil så på alle indkommende søgninger se om der er et resultat fra en GPS modtager. Hvis der er, vil der blive indsat en ny tuple i relationen none_geocoded_nets. I relationen none_geocoded_nets er det en identifikation af hvilke nets der i øvrigt var med i søgningen. Herefter udføres søgningen så som tideligere beskrevet.
Laget foran *ISearch* er implementeret som en decorator.

På serversiden har vi så en tråd kørende, som løber indenholdet af none_geocoded_nets igennem, finder de nets frem der er tale om, beregner en spatial lokaltion, og derefter gemmer den oplysning i geocoded_nets.
Denne tråd er realiseret i klassen *Indexer.*

Vi vil således med tiden opbygge en database over, hvor vi stedbestemmer de enkelte nets.

Baggrunden for denne fremgangsmåde er, at denne feature så bliver en tilføjelse til systemet mere end en ændring. Og performance indflydelsen på en søgning vil være minimal, når vi laver beregningen på bagkant.

Systemet altså vil løbende lære hvor den enkelte sender befinder sig. Illustreret ved billeder herunder

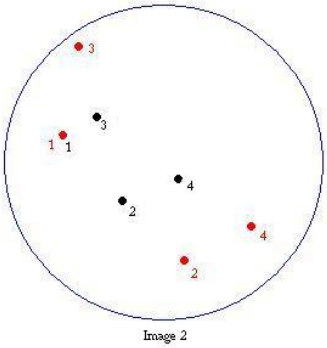
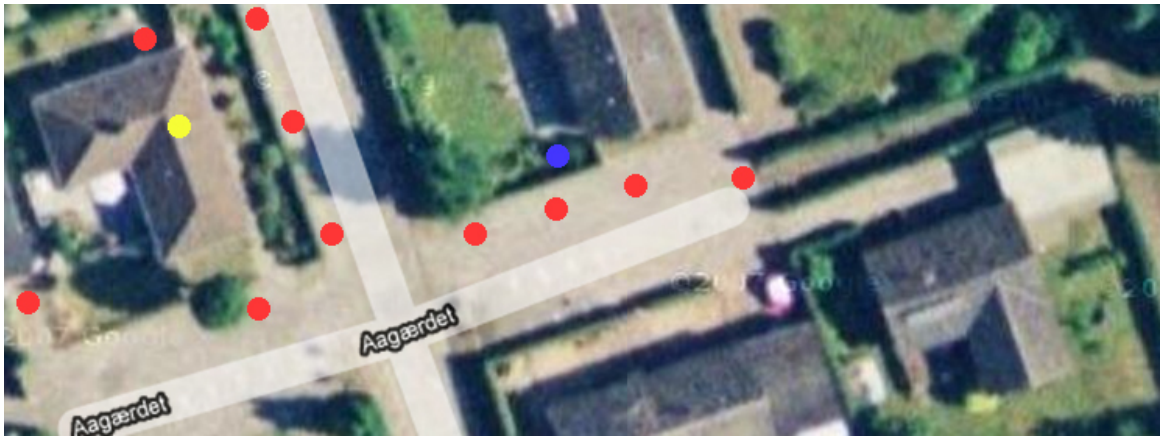


Image 2

Cirklen illustrere området der er dækket af et net, senderens faktiske placering er midt i cirklen. De røde prikker illustrere mållinger der er indrapporteret, og de sorte prikker hvor systemet mener at en radiosender befinder sig.

Den algoritme der bruges er begrænset af de informationer der sendes fra klienten, og vi har i udgangspunktet valgt ikke at sende signalstyrke med. Vores algoritme virker ganske simpelt ved at beregne en middelværdi for de registered GPS punkter. Hvis et net er set i (1, 1) og (3, 3) vil vi regne os frem til at senderen er i (2, 2).

I forbindelse med afprøvningen af denne algoritme forsøgte vi at bestemme placeringen af et access-point med en kendt position.



Den gule prik indikere hvor access punktet er installeret, de røde punkter er steder, hvor vi har målinger fra det, og den blå prik indikere hvor algoritmen har regnet sig frem til, at det access point må befinde sig.
I dette tilfælde er der tale om et WiFi net, og vi rammer cirka 20 meter ved siden af, men den eneste grund til at det ikke er mere er WiFi nettets begrænset rækkevidde. Algoritmen flytter altså sendere i retning af hvor der er tættest trafik. Hvis eksemplet havde været en GSM sende maste havde dette fænomen været mere udbredt.
Her er plads til forbedringer, men det ligger uden for fokus af denne opgave.

GPS uden GPS sensor

Ved hjælp af ovenstående system har vi altså nu mulighed for, at tilføje et GPS punkt til en søgning der er fortaget uden GPS aktiv GPS sensor.
Ligeledes en decorator på *ISearch* vil kunne teste på om der er en GPS position blandt de indkommende sensor resultater, hvis der ikke er, kan vi se om nogen af de nets der er opfanget eksistere GPS koordinater til, hvis der gør, beregnes et gennemsnit mellem disse, og sendes med videre til den decorated *ISearch.*
Der er nogle gode grunde til at gøre dette, dels vil vi kunne give en GPS position til folk der befinder sig indendøres, dels kan brugeren køre uden aktiv GPS modtager.

Dette er blevet implementeret ved hjælp af *GpsExtenderDecorator.* Det er klart at den algoritme som bruges ikke per nuværende er optimal. Og aktivering af denne egenskab skal gøres med forsigtighed.

Netværkshieraki

Nogle opfangede sensor resultater vil altid have en større præcision end andre. Eksempelvis vil en GSM sende-maste dække et større område end et WiFi access punkt.
Vi gemmer altid hvilke nets der er opfanget i samme skanning.

Hvis vi tør at antage at en GSM sender altid dækker et WiFi access punkt helt, vil vi på baggrund af dette knytte en GSM sender til en søgning udført af en enhed, der ikke har aktiv GSM radio.
Vi har implementeret en klasse *ExtendedSearchDecorator,* som tilføjer denne egenskab til en *ISearch,* igen er der her tale om en feature, der kan kobles ind og ud, som man har behov for det.

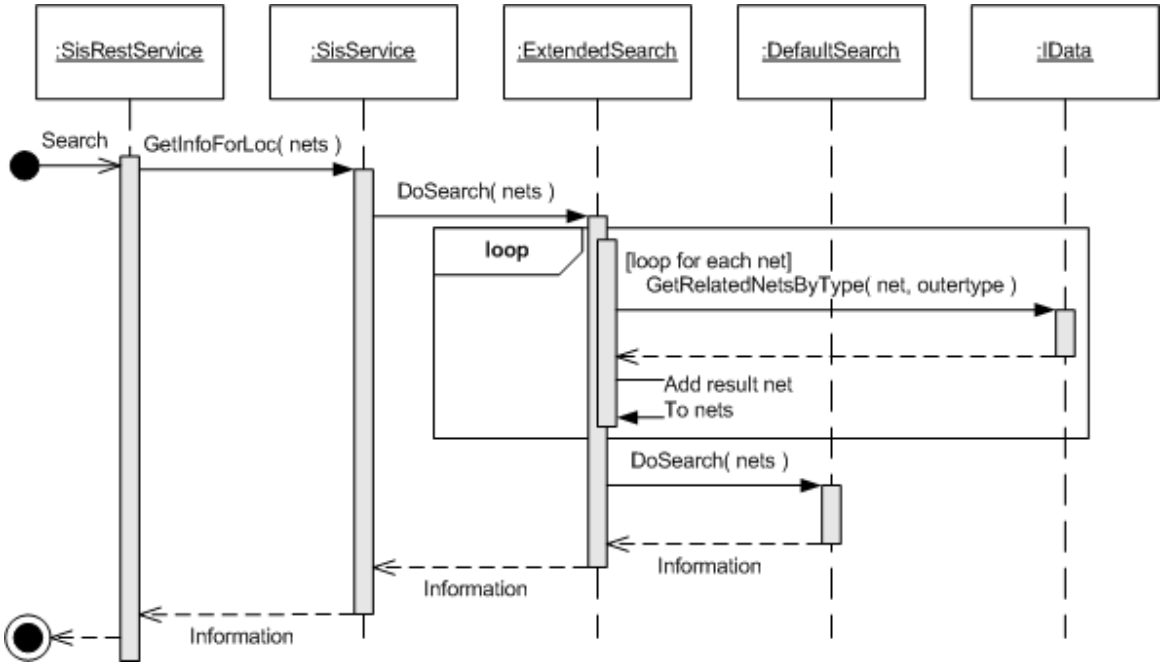
```
public ISearch GetSearcher( IData data )
{
   ISearch result = new DefaultSearch( data );
   result = new ExtendedSearchDecorator( data, result, "small", "large" );
   return result;
   ...
```

Small er typen af nets som har en mindre udbredelse end typen large. Oplagt kunne være at bruge parametrene "WifiMac" og "GSMCellID".

Note: Normalt kan en GSM modtager kun fortælle omkring master som er opstillet af din mobiludbyder. Ved hjælp af den overstående Decorator kan vi linke os igennem det problem. Gennem koden:

```
result = new ExtendedSearchDecorator( data, result, "WiFiMac", "GSMCellID" );
result = new ExtendedSearchDecorator( data, result, "GSMCellID", "WiFiMac" );
```

Forudsætningen for at det virker er naturligvis, at der er data for dette i databasen. For at visualisere følger her et sekvensdiagram, hvor en *ExtendedSearch* er sat ind foran den aktive *ISearch*.



Flowet vil være det samme for *gpsExtenderDecorator.* For *DefaultSearch* er det helt transparent om nogle af udvidelserne er aktive.

Fælles for for begge systemer til udvidelser af klient sensor er at de introducere en vis grad af usikkerhed.
Eksempelvis kan man godt forstille sig et WiFi net som ligger ud over grænsen af af dækningen fra en GSM celle. Her vil der kunne opstå situationer hvor *ExtendedSearchDecorator* tilføjer nets til søgningen som klienten faktisk ikke er i. Ligesom *gpsExtenderDecorator* ikke giver den præcision som man normalt forventer af en GPS modtager.
Thomas Riisgaard Hansen skriver om "A Context Awareness Model"[17] hvor man vejer præcision og pålidelighed mod den handling og konsekvensen af en forkert beslutning mod hinanden. Udvidelserne her er lav på præcision og pålidelighed - hvorfor man klart skal overveje konsekvensen hvis man vælger at bruge dem i sin løsning.

SIS Protokollen

Kommunikation mellem server og klient skal vi have defineret. I vores valg af protokol har vi lagt vægt på:

- Nem implementering
- Så lille et overheat som muligt
    - Bitflytning er dyrt
- Læsbar data
    - Erfaringsmæssigt er det noget nemmere at debugge noget man kan læse.
- Simpelt at parse
    - Vi kan ikke være sikker på at klienterne har et bibliotek der understøtter dette.

Som følge af vores valg af .NET 4.0 platformen på serversiden har vi fået foræret understøttelse for JSON eller XML til vores RESTfull webservice.
Til kommunikation mellem server og klient har vi valgt at bruge JSON. Primært fordi overheat er mindre end i XML. Vi har to objekt typer der skal kunne flyttes i protokollen. Net og InformationsPakke.

Net

Vores net ser således sådan ud

```
{
   "Type":"<typen af net der er fundet>",
   "Value":"<Værdien af søgeresultatet>"
```

```
}
```

Eksempelvis hvis vi finder en bluetooth enhed med MAC adressen AA:BB:CC:DD:EE:FF blive til {"Type":"bt","Value":"AA:BB:CC:DD:EE:FF"}

InformationsPakke

En informationspakke vil se således ud

```
{
   "Information": {
      "Id": <Valgfrit database id>,
      "Titel": "Overskrift",
   &amp;nbsp; &amp;nbsp;  "ValidUntil&amp;amp;quot;: "<Optionel udløbsdato>"
    },
    "InformationsData": [
       {
          "Data": "<Data for information>",
          "Description": "<Beskrivelse af information>",
          "Tags": [
             {
                "Tag": "<Tag - bemærk, kan være flere>"
             }
          ],
          "Type": "<Typen af infomration>",
          "ValidUntil": "<Udløbs data, ligeledes optinal>",
          "id": <Valgfrit database id>
       }
    ],
    "Lokationer": [
       <JSON net typer>
    ]
}
```

Database ID'erne er nødvendige i det tilfælde at vi skal understøtte opdateringer er data.

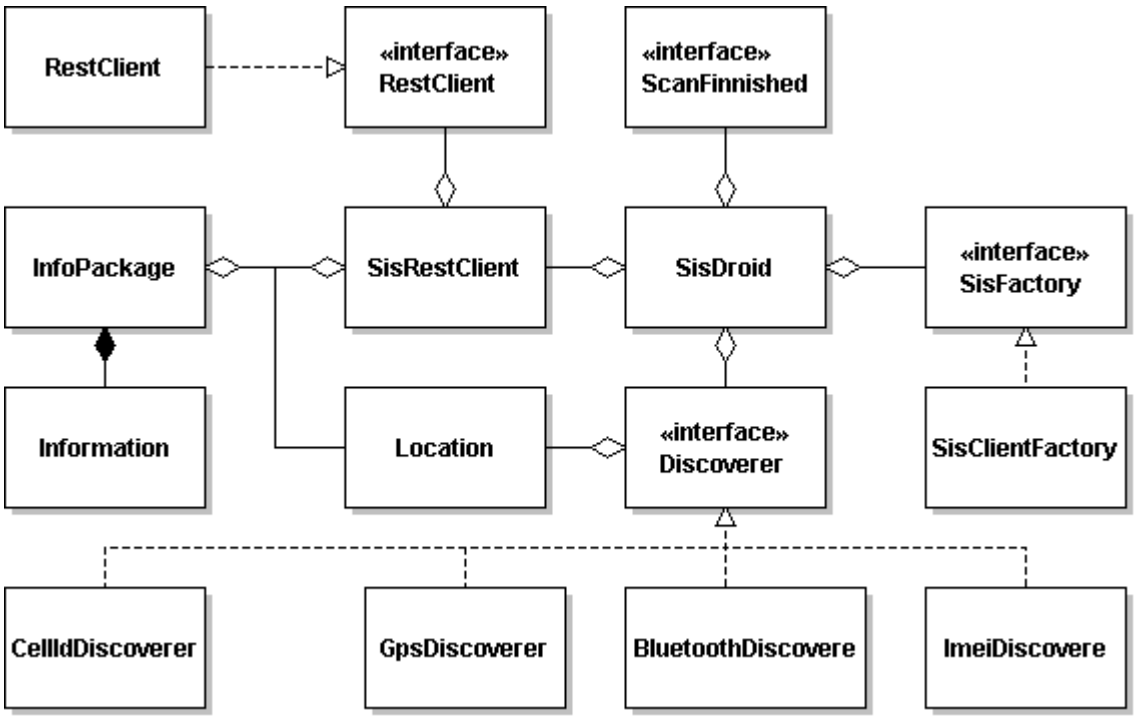
Design og implementering af mobilklient frameworket (SisDroid)

På klient siden har vi valgt at lægge os på Android. Hensigten er at holde klienten så tynd og så lille som muligt, dels for at sikre mindst muligt porteringsarbejde, og for at spare mest muligt på behandlingstiden på mobil siden.

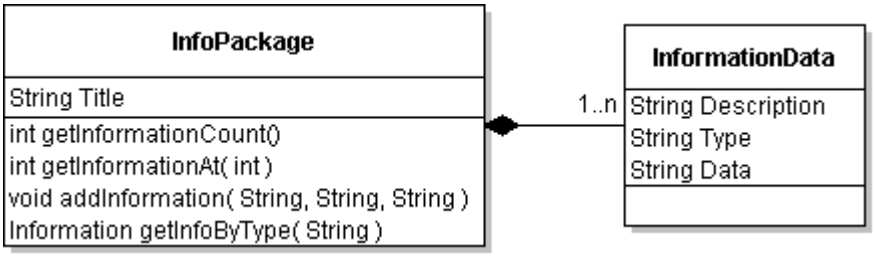
Overordnet skal klienten skanne området for alle de informationer der er tilgængelige/konfigureret. Sender dem til en server. Og præsentere resultatet til brugeren.

> Note: Generelt i vores UML klassediagrammer gælder det at alle properies der er skrevet i de enkelte klasser, er tilgængelige via get/set metoder.

Frameworket er designet som:



InfoPackage beskriver en samling informationer, som kan være knyttet til en eller flere lokaltioner. *InfoPackage* har en eller flere *Information*. Modellen ligger altså op til to niveauer i informationerne Eksempelvis et CV, der ville selve CV'et være repræsenteret i en *InfoPackage* - mens personens hjemmeside ville være en *Information.*



I en *Information* er Type en indifikation for klienten om hvilken type data instansen har. Description er en beskrivelse, og Data er det egentlige data. Man kunne

eksempelvis forstille sig følgende { Description := "Min personlige hjemmeside", Type := "URL", Data := "http://daimi.au.dk/~wegelbye" }

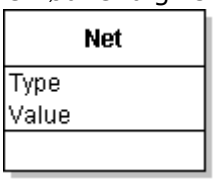
En alternativ model blev diskuteret hvor *Information* var specialiseret ud i mere specifikke klasser - eksempelvis en HTTP URL klasse, en Billede klasse.

Men ovenstående datamodel blev valgt for at give størst fleksibilitet for klienten - frameworket er så at sige ikke konkret i forhold til, hvad det er for noget information der er gemt, blot at der er information.

Vi kunne godt have undværet type, og så overladt det til klienten selv at finde ud af hvad de enkelte *Information* holder. Men med den valgte struktur, kan vi lave en getInfoByType metode, som virker meget overbevisende for klienten.

Net

Net er en klasse, der beskriver de netværk, som vores sensors har fundet. Navnet Net er ikke super, da navnet foregiver, at det er et Netværk af en eller anden slags, man har fundet, men det er det ikke nødvendigvis.



Propertien Type er en beskrivelse af hvilken slags information der er fundet, og Value er en beskrivende værdi.
Ideen er at den samme Value kan forfindes i forskellige Type. Eks. en MAC adresse på en Bluutooth adapter er lidt anderledes, hvis det er din egen adaptor eller din adapter er skannet en en anden applikation.
Begge værdierne er typen String, og på serversiden kan vi så beslutte om nogen *Net* skal have special håndtering. Hvis man eksempelvis scanner med aktiv wifi og GPS kunne man eksempelvis få et resultat.

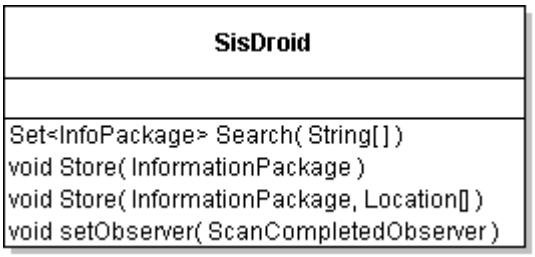
Type	Value
GPS	56.474758/10.045699
WIFISSID	HOTSPOT
WIFIMAC	AA:BB:CC:DD:EE:FF
WIFIMAC	11:22:33:44:55:66

Enheden har altså plottet sig på en GPS koordinat. Plus den kan se to accesspoints, som er på samme trådløse netværk.
I forbindelse med vores design af systemet har vi diskuteret frem og omkring denne struktur, specielt GPS koordinatet, hvor vi har brug for at lave en magisk formateret

streng. Men den valgte struktur har vundet for at holde klienten, som har de begrænsede resourcer, så let og simpel som muligt.

SisDroid

SisDroid er en Facade til frameworket, alle, for klienten, funktioner af frameworket er eksporneret her.



Search er metoden til at søge i det område man er. Store giver mulighed for at gemme information. Der er to udgaver af Store() en med en liste Location og en uden. Forskellen er om klienten selv vil bestemme hvilken kontekst der skal bruges, eller om informationen skal bindes mod den kontekst som alle de aktive sensors kan finde på det tidspunkt Store bliver kaldt. Endeligt har vi setObserver som kan tilføje en observer ind i frameworket.

Intentionen med en Facade er at gøre tilgangen til et mere kompleks kodestruktur simpel. Det sker gennem at Constructoren, får sat SIS frameworket op til en kørende tilstand, plus den eksponere metoderne til at kalde instanserne af de enkelte klasser i frameworket.

RestClient

RestClient er en wrapper omkring en http client. Som er i stand til at lave http/post og http/get - som er de metoder, der skal bruges til at kommunikere med en RestFull webservice. Http/put og Http/delete er også en del af Restfull webservices, men dem bruger vi ikke i vores kommunikation.
RestClient er i øvrigt implementeret mod et interface. Selv om i produktion vil det altid være den samme klasse, der bliver instansieret. Dette er valgt for, at vi i vores testkode kan indsætte en test stub, hvor vi har kontrol over resultaterne.

SisRestClient

SisRestClient er igen en wrapper omkring en *RestClient* - den giver mulighed for at udføre en Store og Search i mod vores server.
Det er også her at objekter af typen *Net* og *InfoPackage* kan blive pakket ind i JSON.

Der findes et JSON bibliotek, som er tilgængeligt på android telefonen. Men en performance undersøgelse har vist, at det er mange gange hurtigere, at bruge en StringBuilder og så selv bygge datastrukturen op.
Undersøgelsen bestod i at lave en simpel klasse med to getMetoder, instansiere en liste af typen herefter encode dem til JSON. Koden for dette eksperiment er i projektet jsonlibperformancetest. Herefter følger en tabel med test resultaterne. (Testen er udført på en Google G1 telefon - vi ser ingen grund til, hvorfor forholdet skulle være anderledes på en anden model - de enkelte test er udført 5 gange og de angivende tider er gennemsnitstider)

Antal objekter	com.sun.JSON (ms)	StringBuilder (ms)
10	92	18
100	821	216
10000	92181	14430

Da denne encoding vil komme til at være en operation der udføres ofte i vores framework, har vi, af hensyn til mobiltelefonens begrænsede ressourcer, valgt selv at implementere JSON syntaxen.

ScanFinnishedObserver

Denne klasse er en observer, som bliver kaldt hver gang en skanning er færdig.

Når frameworket er færdig med en skanning, lige før der skal kommunikeres med serveren, bliver metoden scanCompleted kaldt. Hvis metoden returnerer false fortæller det frameworket, at brugeren af en eller anden grund har vurderet at kommunikationen ikke er nødvendigt.

Konkret fik vi behov for denne observer, da vi skrev en applikation til opsamling af data. (WarDriver) - der ville vi ikke sende data, med mindre vi havde en wifi forbindelse.
Det ligger lidt uden vores frameworks virkeområde, at beslutte om det er aktuelt at søge efter data, derfor har vi introduceret inversion-of-control her.

Referencen til en assigned observer ligger i Facade objektet. I constructoren af SisDroid sættes referencen til et annonymet NullObject[18].

```
_scanCompletedObserver = new ScanCompletedObserver()
{
   @Override
   public boolean scanCompleted(Set<Net> data)
   {
      return true;
   }
};
```

På den måde undgår vi, at vi ved hver eneste kald til observeren bliver nødt til, at teste om _scanCompletedObserver er null.

Sensor

Sensor er et interface, der beskriver vores standard sensore. Det eneste interfacet kræver er at implementeringen implementere en metode getNetIds som returnere et *java.util.set<Net>.*
Systemet bliver leveret med en række standard sensors. Generelt vil frameworket ikke selv tænde for nogen sensors, det er op til brugeren.

Note: Det er ikke alle Sensors herunder, der er listet i klassediagrammet tidligere.

BluetoothSensor

BluetoothSensor er en klasse, der finder relevant information omkring BlueTooth enheder i området nær sensoren.
I Android forgår en skanning via et asynkront kald. Klienten skal så subscribe til events for, at få oplysninger omkring hvilke enheder, der findes i ens område.

CellIdSensor

CellIdSensor bruges til at finde informationer omkring de gsm/cdma celler telefonen kan se. Dels er det muligt at få et event hver gang enheden skifter sendemast, og dels er det muligt, at få en liste af celler der kan ses.
Den celle enheden er parret med lige nu er markeret specielt.

CompositeSensor

CompositeSensor er egentligt ikke en sensor i sig selv. Men giver mulighed for at kombinere søgeresultatet fra 2 eller flere sensors.

EmptySensor

EmptySensor er en sensor der aldrig finder noget, den returnere altid en tom liste. Den tjener som vores standard sensor.

GpsSensor

GpsSensor kan finde GPS positionen, hvis et GPS signal er tilgængeligt. Hvis GPS modtageren er slukket, eller ikke kan finde satelitter nok, så returneres en tom liste.

ImeiSensor

ImeiSensor bruges til at sende oplysninger omkring imei nummeret i telefonen. Tanken er at denne information kan bruges som identifikation af de enkelte klienter.

WifiSensor

WifiSensor kan bruges til at finde oplysninger omkring de trådløse netværk der er. Både MAC på access points, og SSID på wifi nettet.
Derudover sendes der en speciel type for den MAC og SSID der har størst signal styrke.

Opsummering over understøttede Sensorer

Følgende tabel beskriver hvilke type af information vi som standard understøtter.

Type	Class	Eksempel data	Beskrivelse
bt	BluetoothSensor	AA:BB:CC:DD:EE:FF	MAC adressen på enheder som er fundet i den seneste bluetooth scanning.
ownbt	BluetoothSensor	FF:EE:DD:CC:BB:AA	MAC adressen på den bluetooth enhed der er brugt til at skanne med. Typisk vil det i praksis være en unik identifikation

			af den mobile enhed.
CellGSM	CellIdSensor	1234/5678	gsm cellid og gsm location area code på tilgængelig GSM sendemaste.
CellUMTS	CellIdSensor	2345	Primary Scrambling Code for en UMTS
Ccell	CellIdSensor	3456/2345	Den cell som enheden lige nu er parret med
gps	GpsSensor	52.1234/9.4352	Længdegrad og bredegrad for nuværende possition
imei	ImeiSensor	35-209900-176148-1	Imei nummeret på enheden som er i brug
wifimac	WifiSensor	11:22:33:44:55:66	Mac adressen på et synligt accesspunkt
wifissid	WiFiSensor	MyHotspot	SSID på et net som er er synligt per nuværende
swifimac	WifiSensor	22:33:44:55:66:77	MAC adressen på det access-point som har størst sende effekt.
swidissid	WifiSensor	TheOtherSpot	SSID på på det access-point som har størst sende effekt

SisFactory

Er et interface der beskriver de ting, der er nødvendige for at sætte frameworket op.

To objekter er nødvendige at få oprettet i vores system for at få systemet til at fungere, en RestClient og en Sensor.

SisDroidFactory

SisDroidFactory er en standard implementering af SisFactory. Ud over at implementere interfacet *SisFactory* så giver den mulighed for at aktivere de sensors der er angivet som standard.
Dette gøres gennem metoden setEnabledStandardSensors(int). Parameteren er en bitvector sammensat af de konstanter der er defineret i SisDroidFactory klassen.

```
public final static int SENSOR_BLUETOOTH = 1;
public final static int SENSOR_WIFI = 2;
public final static int SENSOR_CELLID = 4;
public final static int SENSOR_IMEI = 8;
public final static int SENSOR_GPS = 16;
```

Hvis der ønskes at aktiveres mere end en, gøres det med bytewise or. Eksempelvis hvis en klient vil aktivere cellid og bluetooth er det:

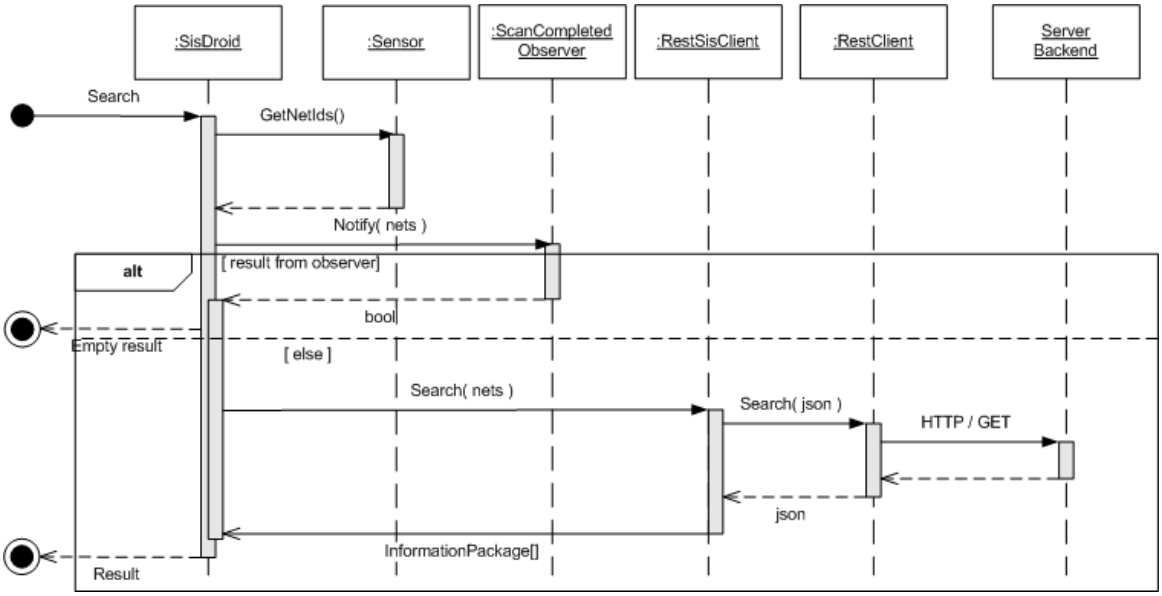
```
factory.setEnabledStandardSensors( SENSOR_BLUETOOTH | SENSOR_CELLID );
```

Hvis en klient skulle have brug for, at indsætte en brugerdefineret sensor. Kunne denne klient også bruge *SisDroidFactory* ved hjælp af metoden

```
public void addCustomSensor( Class<? extends Sensor> newClass )
```

Sekvensdiagram

Diagrammet giver et overblik over flowet i frameworket.



Klassen *Server Backend* er hele den del af systemet der forgår inde i maven på serveren.

Fordelen ved dette layout er

- Vores kommunikations lag kender ikke noget til forskellen i de enkelte Sensors.
- Der kan nemt udvikles nye sensore, og alt frem og tilbage vil virke som det skal.

Reflektion over udviklingen af SisDroid frameworket

Det miljø[19] som Android projektet tilbyder er meget velfungerende. Det API som bliver stillet til rådighed virker gennemført. På trods af at vi har haft brug for at lave nogle ret hardware nære operationer, så var der altid noget under android pakken der kunne hjælpe.
Erfaringen i gruppen fra tideligere context-aware computing projekter er, at sådan nogle ting mange gange var skjulte af api'et.

Riva og Kangasharju[20]skriver, at man skal være forsigtig, specielt i forhold til begrænset RAM, processorkraft og båndbrede, når man udvikler mod mobile enheder. Men SmartPhone området er i rivende udvikling, rygterne siger, at Motorola vil producere en 2Ghz model[21] inden udgangen af 2010. En HTC Desire telefon har 512 mb ram.
Hvis man læser Androids eget notat omkring programmering for performance[22], kan man læse, at man altid skal prioritere gode algoritmer og struktur.
Med de ting i baghovedet, mener vi at de vigtige begrænsninger i context-aware computer programmering i dag mest er relateret til varierende netværks forbindelser og batterikapacitet.

Forskelligheder i serverside / klientside model

Som en konsekvens af at vi har valgt på serversiden at lave special-håndtering af resultatet af GPS sensor, er der opstået noget navneforvirring hvis man sammenligner kode fra serveren versus klienten.
Tabellen herunder giver et overblik over hvilke typer der svarer til hvad og hvor.

Klient	Server	Rest	Database	Beskrivelse
Net	Lokation	Lokation	N/A	Svarer til den rå data som er opfanget af en sensor.
N/A	GpsPossition	N/A	gpsPositioner	Er de nets der kommer fra klienten som kan identificeres som et GPS koordinat
N/A	Net	N/A	Nets	Er de nets der kommer fra klienten, som ikke kan identificeres som et GPS koordinat

Test Driven Development i klient frameworket

TDD på android platformen er egentligt godt understøttet, det er muligt, fra et Android test projekt at instrumentere en Android enhed ret langt ud i detaljen.
I test driven development arbejder man efter forskellige patterns[23] herunder "test first" - som foreskriver, at man skal teste sin næste implementering før man programmerer den.
Over 50% af klient koden er relateret til aktivering af forskellige sensors.
De to ting er ikke nemme at kombinere, og et forsøg på dette vil hurtigt resultere i, at testen omhandler test af Android SDK'en mere end test af SIS Frameworket.
I den anden ende af SIS klient frameworket har vi RestClienten, TDD i dette område ville hurtigt blive til en test af apaches HttpClient - og helt ærligt, det tror vi altså Apache Software Foundation har styr på.
Det primære der har været muligt at pakke godt ind i nogle unit tests, er området hvor Net og Informationer bliver enten pakket ind eller ud i JSON. Altså klassen *SisRestKlient*.

Evaluering

Ud fra den erfaring vi har fået ved, at komme til den nuværende iteration af vores framework samt prototyper, har det vist sig muligt at lave et framework der væsentligt forbedrer en udviklers muligheder for at lave informations deling i et intelligent lokations netværk.

I følgende afsnit beskriver vi nogle af de prototyper, som vi har lavet for at teste/ evaluere SIS frameworket.

Brug/test af frameworket

I følgende kapitel vil vi afprøve SIS frameworket. Afprøvningen foregår ved, at vi laver nogle mindre programmer, som gør brug af SIS frameworket. I vores evaluering vil vi fokusere på hvor lang tid der bruges på integration af frameworket. Programmerne vil ikke blive til færdige programmer, men blive implementeret med henblik på integration af SIS frameworket.

BlueSocial

BlueSocial er et program, der er tiltænkt som et socialt netværksskabende program til en Android telefon.
Tanken er at man via sin telefon kan være i stand til at oprette en profil, skrive lidt kort tekst om sig selv, tilføje nogle informationer, såsom profil billede, hjemmeside og/eller facebook bruger detalier osv. Denne information vil andre brugere, hvis de er i umiddelbar nærhed af dig, kunne se.
Tanken er at man så i offentlige fora, i toget, venteværelset ved lægen og lignende, kan se lidt omkring de folk du er i nærheden af, og på den måde kunne skabe nye sociale relationer.

Implementere en ListActivity, fylde denne med et søgeresultat med en bluetooth sensor. Er kort sagt det, der er brug for.

Brugsmønstres

Navn: Søg efter information via synlige bluetooth enheder

Succes kriterier:
1. Brugeren trykker på søg knappen.
2. Klienten beder SIS frameworket om, at hente al information der kan findes via bluetooth
3. SIS Frameworket returnerer en liste over information der er fundet.

4. Brugeren klikker på en af informationerne
5. Der vises en liste over de data, der er tilknyttet den information.

Undtagelser:
2a.
1. Bluetooth er ikke tændt.
2. Frameworket returnere en tom liste.

Navn: Opret ny information

Succes kriterier:
1. Brugeren trykker på opret knappen.
2. Klienten beder SIS frameworket om at hente al information, der kan findes via bluetooth sensoren
3. SIS Frameworket returnerer en liste over synlige bluetooth enheder, der er fundet.
4. Brugeren vælger de lokationer, som informationen skal kobles på. Der kan vælges mere end en, men der skal vælges mindst en.
5. Brugeren vises en skærm, hvor han kan indtaste stamdata for informationen, han vil gemme.
6. Brugeren kan herefter tilføje informations data til informationen ved at klikke på et plus.
7. Brugeren vises en skærm til indtastning af stamdata for informations dataerne.
8. Via et plus på denne skærm kan brugeren tilknytte tags til hver enkelt informations data pakke.
9. Når bruger trykker ok, tilføjes informations data til informations pakken og vises i en liste.
10. Brugeren trykker opret og informations pakke sendes til SIS frameworket sammen med lokationerne og informations dataene.
11. SIS Frameworket returnerer en informations pakke hvor der er blevet tilknyttet database ID'er.

Undtagelser:
2a.
1. Bluetooth er ikke tændt.
2. Frameworket returnerer en tom liste.

Navn: Opdatere information.

Succes kriterier:
1. Brugeren trykker på søg knappen.
2. Klienten beder SIS frameworket om, at hente al information der kan findes via bluetooth.
3. SIS Frameworket returnerer en liste over information der er fundet.
4. Brugeren klikker på en af informationerne.
5. Hvis informationen er oprettet på brugerens telefon, vises der en rediger knap.
6. Brugeren klikker på rediger knappen.

7. Brugeren får vist stamdata skærmen fra oprettelse og kan nu redigere data.

Undtagelser:
2a.
1. Bluetooth er ikke tændt.
2. Frameworket returnerer en tom liste.

Koden som BlueSocial skal bruge for at søge efter omkring værende klienter er således.

```
SisClientFactory fac = new SisClientFactory( this );
fac.setEnabledStandardSensors( SENSOR_BLUETOOTH );
SisDroid sisdroid = new SisDroid( fac );
HashSet<String> s = new HashSet<String>();
s.add( "bluesocial" );
for ( InfoPackage info : sisdroid.Search( s ) )
{
   ...
```

Det ses tydeligt, at det er relativt lidt kode for at

- Aktivere sensors
- Skanne området
- Oprette forbindelse til server på internettet
- Søge informationer frem

BlueSocial præsenterer søgeresultatet i en Android ListActivity. Hvilket kan gøres ved hjælp af et Set af HashMaps. Derfor skal søgeresultatet nu blot mappes ind i sådan en datastruktur.

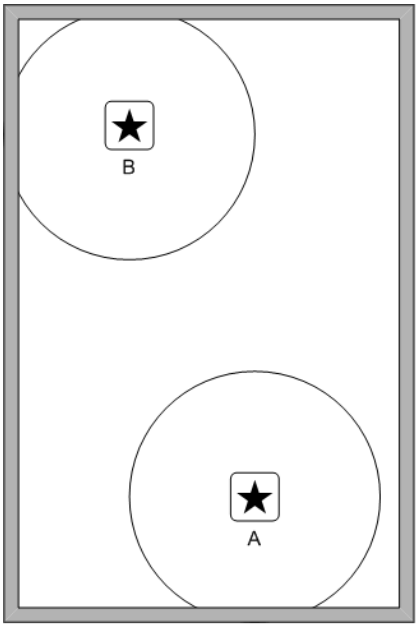
```
   ...
   HashMap<String, String> newEntry = new HashMap<String, String>();
   newEntry.put( "header", info.getTitle() );
   newEntry.put( "teaser", info.getInfoByType("teaser").getData( ) );
   list.add( newEntry );
}
```

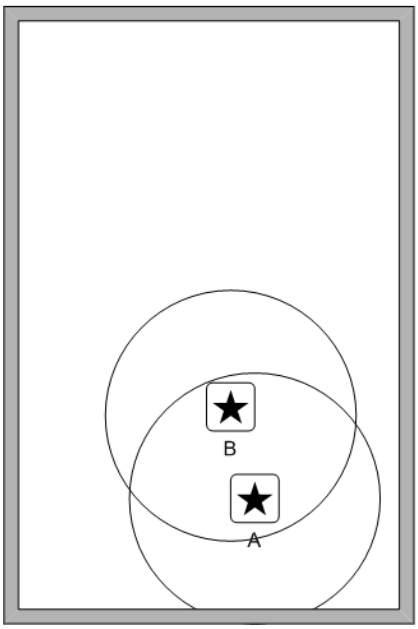
For at gemme data, skal følgende kode bruges

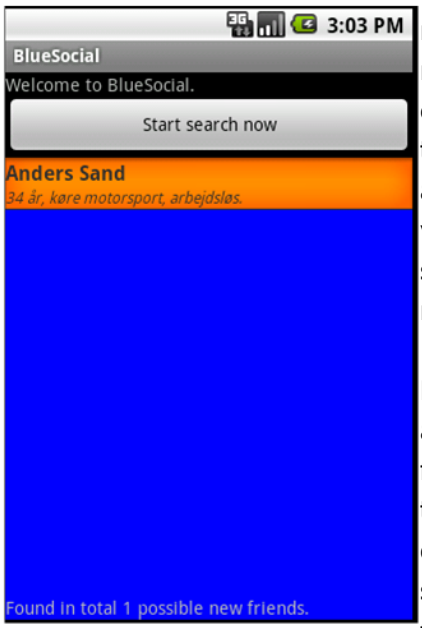
```
InfoPackage ip = new InfoPackage();
ip.addInfofmationData("teaser", "En mand, siger nogen", "Just for fun" );
ip.addInfofmationData("www", "http://www.folketinget.dk", "Min hjemmeside" );
ip.setTitle( "Morten Wegelbye Holm" );
```

```
ip.setTag( "bluesocial" );
Set<Net> s = new HashSet();
s.add( new Net( "bt", "AA:BB:CC:DD:EE:FF" ) );
sisdroid.Store( ip, s );
```

Når der gemmes data, har klienten to muligheder, enten skal denne selv angive hvilke nets der skal bruges, eller lade være med at angive nogle nets. Hvis der ikke angives nogen net. Så vil SIS Frameworket automatisk gemme den nye information på alle de nets som de aktive sensores kan se.







Integrationen af frameworket gav i dette eksempel understøttelse for lokaltion til data på mindre end en time. (Naturligvis hjulpet på vej af at udviklerne også havde lavet S:I:S frameworket)

Prototypen her er tænkt som en tjeneste, der kan bruges til at skabe nye sociale relationer, hvor de eksisterende sociale tjenester mere arbejder med, at vedligeholde de sociale relationer man har.

BlueSocial er som applikation langt fra færdig. Men alle de ting der skal virke omkring de tjenester som S:I:S frameworket tilbyder, er afprøvet og virker som forventet.

Bluetooth som sensor

Vi er klar over, at bluetooth har nogle begrænsninger i forhold til skanning af tilgængelige resourcer - dels er der tidsforbruget og dels er det en strømkrævende process.

Strømforbruget er så højt at reference telefonen, en G1, tømte et 1150mAh batteri i løbet af 6 timer. Hvis der blev udført en skanning hvert 3. minut.

I version 4[24] af Bluetooth specifikationen er der indført en ny teknologi til enheder, der kræver lang batteri levetid istedet for overførsels hastighed og rækkevidde. De kalder det meget originalt *"Bluetooth* low energy technology" og det skulle bruge mellem ½ og 1/100 del strøm af, hvad en standard klasse 2 enhed bruger(2,5 mW).

Strukturen i bluetooth discovery process tager mindst 10,26 sekunder[25] - I ren ventetid på en applikation, er det meget lang tid.

Det er tilladt at aborte en discovery process, hvis man har fundet den enhed man søger. Men da vores system ikke kender noget om de enheder, der har vedhæftet information, er det ikke en mulighed.

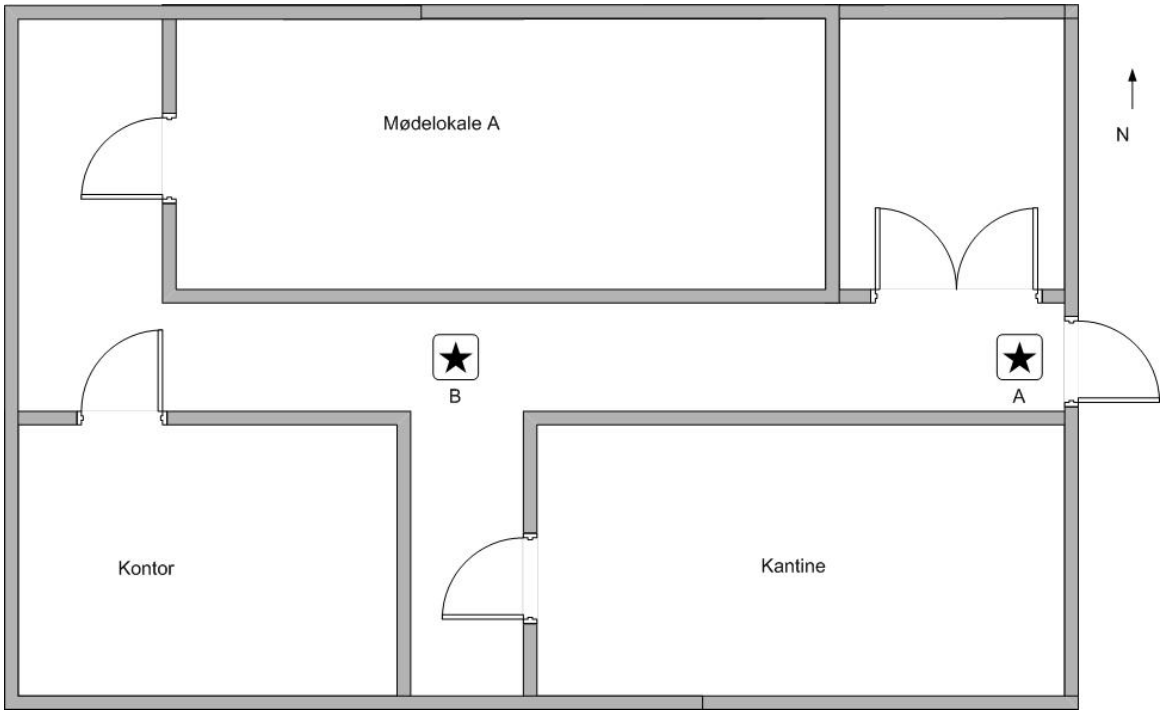
Vi overvejede kort, om vi kunne lave det om således, at der kun blev udført en skanning, hvis vi kunne registrere ændringer på nogle af de andre nets. Men præcist for bluetooth kan det være andre enheder, der er blevet flyttet i retning af dig.
En anden løsning kunne være at lade systemet teste om nogle andre klienter kunne have bevæget sig ind i nærheden af dig - men det ville kræve at systemet er deployed bredt - med andre ord man ville ikke kunne udnytte eksisterende infrastruktur.

Bluetooth som punkt for din localtion

På den platform vi har valgt, har det vist sig, at en bluetooth enhed ikke er discoverable som standard. Hvis man i android ønsker, at noget skal etablere en bluetooth forbindelse til din enhed, skal du manuelt aktivere dig selv som discoverable. Det er ikke hensigtsmæssigt for vores system, da vi ikke ønsker, at skulle køre nogen service, ej heller nødvendigvis installere noget på informations punkter.

Indoor PathFinder

Indoor pathfinder er tænkt som et alternativ til eksempelvis indendørs skiltning i sygehuse og virksomheder. Et klassisk eksempel hvor GPS må se sig slået.
Ved at knytte informationer til den eksisterende trådløse infrastruktur, og tilføje en retnings sensor vil man kunne bruge vores framework til at guide folk rundt i bygninger. Kunne eksempelvis være som vejviser i et konference center eller et sygehus.
Til udviklingen af prototypen arbejder vi ud fra et tænkt eksempel.



Ved punkt A og B er der installeret et access-point, som begge er oprettet i S:I:S databasen. Dertil er knyttet data således, at hvis man står i nærheden af punkt B og kigger mod vest vil man så kunne se at det er retning mod Mødelokale A og kontor, kigger man mod syd, vil man se at det er retningen mod kantinen.
Står man ved punkt A, og kigger mod vest, vil det ud over mødelokale A og kontor også være retningen mod kantinen.

For at lave denne klient, er det nødvendigt at indsætte en brugerdefineret sensor. Til formålet er der blevet lavet et kompas, der kan fortælle, om man kigger mod nord, syd øst eller vest.

Et object kan så registrere sig som observer på kompasset, og hver gang man skifter retning vil vores server blive kontaktet, og man vil på displayet kunne se hvad man er i retning mod.

Systemet her forsøger at løse det samme problem som Lodestar projektet ved

Alexsandra instituttet[26]. Men hvor Lodestar baserer sig på installation af en ny infrastruktur, er Indoor PathFinder baseret på eksisterende infrastruktur. Indoor PathFinder kræver dog at de besøgende har en kompatibel mobil enhed og applikationen installeret. Applikationen kunne f.eks. installeres ved at man havde en 2D stregkode siddende ved indgangen, som linkede til Android Marked.

Et praktisk krav til infrastrukturen ville være, at man har et access point på de steder hvor brugerne skal beslutte sig for at gå til højre eller venstre.

Modsat eksisterende offline løsninger vil man her kunne flytte om på aktiviteter meget hurtigt.

For at lave indoor pathfinder kræves lidt mere end ved BlueSocial, da man her vil indsætte en brugerdefineret sensor.

```
public class SimpleCompass implements Sensor
{
    ... /*En masse kode til at aktivere compass sensor*/
    @Override
    public Set<Net> getNetIDS()
   {
        Set<Net> data = new HashSet<Net>();
        if ( this.currentDirection != -1 )
        {
            String heading = null;
            switch ( this.currentDirection )
            {
                case N: heading = "North"; break;
                case E: heading = "East"; break;
                case S: heading = "South"; break;
                default: heading = "West"; break;
            }
            data.add( new Net("SimpleCompass", heading ) );
        }
        return data;
     }
    ...
```

I ovenstående kode har vi fjernet alt der ikke er relevant for vores framework. Efterfølgende skal den nye sensor indsættes i frameworket (Linie 3 i koden herunder).

```
SisClientFactory factory = new SisClientFactory( this );
factory.setEnabledStandardSensors( SisClientFactory.SENSOR_WIFI );
```

```
factory.addCustomSensor( mogo.sisdemo.SimpleCompass.class );
_droid = new SisDroidImpl( factory );
Set<String> filter = new HashSet<String>();
filter.add( "Indoor pathfinder" );
Set<InfoPackage> result = _droid.Search(filter);
```

Http body vil eksempelvis nu se således ud (Resultatet fra den indsatte sensor, er markeret med rødt):

[{"Value":"North","Type":"SimpleCompass"},
{"Value":"monitnet","Type":"wifissid"},
{"Value":"00:1d:6a:86:23:3c","Type":"wifimac"},
{"Value":"TDC-AD88","Type":"wifissid"},
{"Value":"00:17:9a:2e:4f:c1","Type":"wifimac"},
{"Value":"monitnet","Type":"swifissid"},
{"Value":"00:17:9a:2e:4f:c1","Type":"swifimac"}]

Eksemplet her er en noget mere kompleks udnyttelse af SIS frameworket, i og med at der har været behov for at oprette en brugerdefineret sensor. Vi ser også at frameworkets primære funktion stadig er lukket land. Klienten har ikke brugt meget krudt på, at knytte information til kontektsten.

Erfaringer med andre prototyper

Vi lavede en simpel prototype uden GUI, der ikke gjorde andet end at skanne efter lokationer både GPS, bluetooth, WIFI osv. Derefter lavede den en søgning på de lokationer den fandt. Derved blev lokationerne oprettet i databasen og relateret til hinanden. Vi kunne se på de GPS positioner der blev oprettet, at selvom vi stod stille på samme sted, så var de aldrig ens. Om det så skyldes, at modtagerene i vores test telefoner var dårlige eller om der var andre forhold der gjorde det, så kunne vi ret hurtigt konkludere, at det ville være umuligt at lave en søgning på lokation ud fra GPS postionerne. Man ville aldrig kunne stille sig på samme sted hvor informationen var oprettet.

Vi fandt derfor ud af, at ved oprettelse af informationer hvortil der var koblet en GPS lokation så skulle der også kobles en radius på den GPS position. F.eks. kunne man angive at alle positioner inden for 10 meter af den oprettede position skulle finde informationen. Det åbner også op for andre muligheder. F.eks. kunne man oprettet information omkring en by ved at stå i by midten og så oprette informationen med en radius på f.eks. 10 km.

På den anden side ville det også give nogle store udfordringer omkring sikkerhed og brugerstyring. Man vil ikke kunne give tilladelse til, at alle kunne oprette information med så stor radius, ellers ville søgninger hurtigt blive oversvømmet af "spam" pakker.

Mere om det i fremtidigt arbejde.

Konklusion

Vi har gennem vores analyse og design fase undersøgt tidligere arbejde på context-aware computing fronten, samt fået klarlagt de muligheder/begrænsninger der ligger i moderne telefoners SDK'er. Vi har gennem en iterativ process med prototyper fået opbygget et framework, der kan håndtere sensorer og dermed kontekst på en moderne platform.

Vores nuværende iteration af frameworket, består af et Android framework(klient), og en transparent backend. Det nuværende setup gør det muligt for applikationer fra forskellige domæner at gemme informationer i samme database. Ideen er at de forskellige applikationer er fælles om at opbygge en omfangsrig kontekst database, der via vores selvlærende nets vil gøre det muligt at søge information på en lokation/ kontekst frem selvom den søgende enhed ikke har de samme sensor muligheder som informationen er gemt med.

Vi har også opdaget, at når man udnytter nogle teknologier til noget de ikke er designet til, kan det medføre nogle problemer. Eksempelvis har vi problemet med WiFi og GSM Cell der flytter sig. Et andet eksempel er Androids ide om automatisk at slukke for sig selv som discoverable.

Vi har ikke haft mulighed for at teste vores system i stor skala, og har heller ikke fået sikkerheds aspektet med i nuværende iteration, hvorfor systemet nok heller ikke pt egner sig til deployment til mange brugere/applikationer.

Det centrale i vores hypotese har vi opnået, i og med det har vist sig muligt, at lave et framework der forsimpler det at knytte information til ens omgivelser, ud fra den eksisterende infrastruktur og de sensorer der findes i mange telefoner idag.

Vi har også opnået den egenskab, at finde de gemte informationer frem igen. Vi har afprøvet det specielt med BlueSocial prototypen.

Vi har opnået, at kunne gemme informationer på bestemte områder, uden at have behov for en lokaltions sensor, og uden behov for at geocode den kontekst brugeren befinder sig i. Dette er specielt afprøvet i eksemplet med Indoor pathfinder.

Vi har lavet et framework der fra en bruges synspunkt vil kunne levere data tilknyttet brugeren kontekst, uden at brugeren skal gøre noget aktivt for at fortælle systemet omkring konteksten.

Fremtidig arbejde

I det følgende afsnit kigger vi på fremtidigt arbejde. Hvad der rører sig ude i verden, samt hvad der mangler på SIS Frameworket for, at det ville kunne bruges i praksis.

Hos Intel Labs arbejder[27] de lige pt. med automatiske twitter/facebook status lignede opdateringer via sensorer fra ens telefon. Nogle af problematikkerne går på, at ikke alt det data der kan opfanges via sensorer vil være interesant for ens venner og familie. En af løsningerne på det problem er, at kategorisere sensor data, så noget data skal trækkes(pull) mens andet skubbes ud (push). Data der trækkes vil også have en begrænset levetid. F.eks. er det ikke interessant for ret mange, om man ligger på sofaen. Det er måske kun interessant for konen/kæresten, der evt. kunne trække den information og så ringe hjem og brokke sig over at man ligger på sofaen istedet for at rydde op, som man havde lovet og derudover så er den information ikke interesant 2 timer senere. Et andet problem er hardwaren. Hvis en telefon skal være tændt med alle dens sensorer aktive så ville dagens telefoner ret hurtigt løbe tør for strøm. To ting er helt sikre, sociale netværk, samt context-aware computing er kommet for at blive og de bliver helt sikkert videreudviklet til ting, vi slet ikke kan forestille os idag.

Problemet med batterikapacitet er ikke nyt - vi har selv oplevet det med reference telefonen - den løber tør for strøm hvis for mange sensor er aktive på samme tid. Problemet er så udbredt at de fleste smartphones i dag, har en widget på skrivebordet til at tænde og slukke forskellige sendere og modtagere. Vi kan kun håbe at den teknologiske udvikling snart får fokus på det problem igen.

Jævnfør sofa problemstillingen ovenfor så skal man i en fremtidig iteration også kunne håndtere information med begrænset levetid. F.eks. vil en information der kobles på en mobil bluetooth enhed, og som er lokations bestemt, ikke være relevant så snart enheden har forladt lokationen. Et eksempel kunne være, at man står foran Eifeltårnet og opretter et link til et fotogalleri med billleder af tårnet. Linket kobles selvfølgelig på lokationen via GPS, men man vælger også en bluetooth enhed man kan se. Det viser sig så at være en turist fra Tysklands telefon. Tilbage i Tyskland 2 uger efter, giver det ikke meget mening med koblingen af Eifeltårns galleriet, som måske endda også indeholder lidt andre ferie billeder med de personer på, som oprettede linket og som ingen relation har til tyskeren, til tyskerens mobiltelefon.

Sikkerhed og brugerkontrol

Sikkerhed og bruger kontrol er også en ting man skal have i et offentligt tilgængeligt database system. F.eks. i relation til Intels forudsigelser så skal man kunne styre den gruppe af venner/familie, der skal have lov til at trække information ud som automatisk registreres af sensorerne. Det er nok heller ikke alle man vil skubbe informationer ud til. Sikkerhed kunne også være at X id'er fra en lokations samling

skal være tilstede for at informationen er tilgængelig. F.eks. kunne man koble information sammen med 2 personers mobiltelefoner samt gps position eller et wifi netværk. Så ville den information kun være tilgængelig, når begge var tilstede på det oprettede sted. Man kunne forestille sig et regnskab eller lignende kun måtte åbnes når x antal direktører var tilstede.

Tilgængelighed har vi ikke taget højde for, i praksis gør SIS frameworket det samme hvis

- Ingen sensors er er tændt
- Ingen sensors fanger nogen information
- Der ikke er nogen netværksadgang
- Serveren er ikke tilgængelig
- Der ikke blev fundet nogen information knyttet til de sensor resultater vi har fundet

I en endelig løsning, vil en klient nok reagere forskelligt på de 5 forskellige fejlsituationer.

Autentiteten af vores data bliver ikke verificeret. En person med onde intentioner kunne relativt enkelt lave sin egen sensor, der sender forkerte GSM celler med til de forkerte WiFi hotspots.

Vi har ingen mulighed for at kontrollere at den klient type der henter informationer ud, også bør have adgang til disse data.

Udvidelse af klientens konfiguration muligheder

På serversiden har vi to tilføjelser, som vi kan ligge ind foran en søgning.
Dels muligheden for at tilføje et GPS punkt til en søgning, hvor klienten ikke har en GPS modtager.
Og dels muligheden for at definere, at en type opfanget net altid vil være dækket af en anden type, eksempelvis WiFi er altid dækket af GSM celler.
Disse to egenskaber kunne vi bygge ind i SIS protokollen, således at klienten kunne aktivere og konfigurere disse to egenskaber.
Det ville også give mulighed for at en klient, som har lavet en brugerdefineret sensor, kunne kæde denne sensor sammen med andre sensor resultater.

Optimering af GPS geocoding

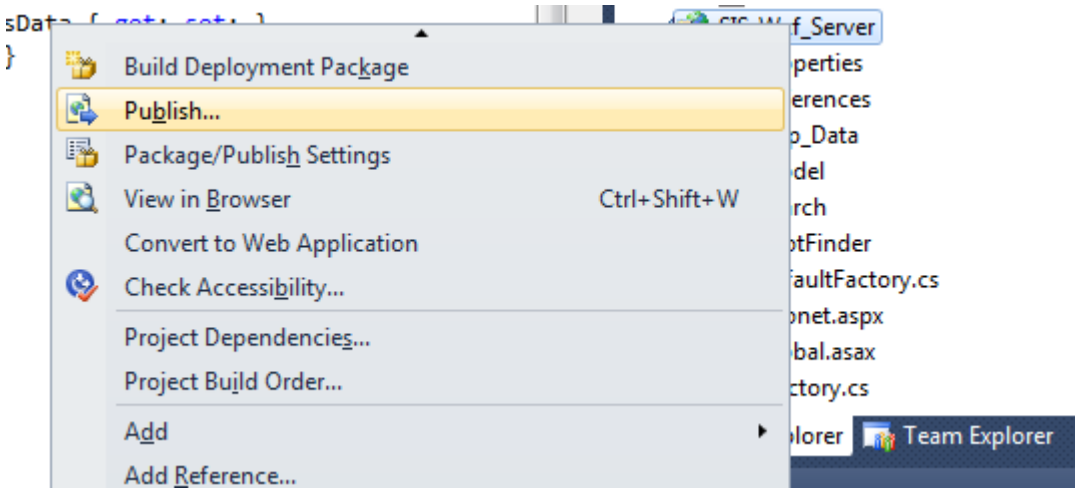
I afsnittet "GeoCoding af nets" har vi beskrevet en ret simpel metode til, at raffinere netværks data til GPS positioner ud fra indkommende sensor resultater. Vi har også illustreret et problem, der er med at vores algoritme for beregning af senderens placeringer flytter placeringen i retning af det sted, hvor der er mest trafik.
Et andet problem, som vi vil forvente ville dukke op er, at nogle sendere som vi betegner som rimeligt stationære, måske ikke er så stationære. Eksempelvis har

man i Android 2.2 introduceret en funktion[28] der i praksis gør Android telefoner til mobile WiFi access-points. Algoritmen bør altså udvides med en metode til at opdage og blokere for disse.

Appendix A - installation og brug af SIS

Installation af Server backend

Installation af serveren er ret simpel. Først publiceres server projektet til et bibliotek. Højreklik på projektet og vælg publish.

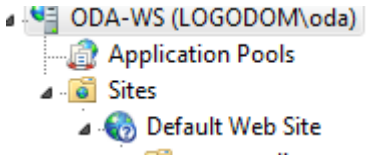


I den næste dialog vælges metode: File System, og der angives en sti hvor filerne skal gemmes.

Vi viser hvordan serveren kan installeres på en IIS 7.5 (Windows 7 og 2008 server). Det kan lade sig gøre, at installere det på en IIS 6, men det kræver lidt flere indstillinger. Bla. vil IIS 6 ikke automatisk acceptere REST urls. Det er samme problematik som hvis man skulle installere en ASP.NET MVC side.

Tilbage til IIS 7.5.

Højreklik på default website og vælg add virtual directory. Giv den et navn(Url) og vælg den sti hvor filerne blev eksporteret til.



Herefter åbnes web.config og connectionstring rettes til, så den passer med ens database server.

Vil man have databasen liggende i App_Data folderen under ens virtuelle sti, kræver

det en masse sikkerhedsopsætninger, som vi ikke vil komme ind på her.

Brug af klient framework

Android har ikke noget shared libary koncept.
Den nemmeste måde at bruge SIS frameworket på, er ved at importere koden i en eget kodetræ.

Specielt vedr. Andorid manifest

Android programmer består ud over kode også af et AndroidManifest.xml som blandt andet bruges til at fortælle hvilke ressourcer et program ønsker adgang til. Følgende kan indsættes som som <user-permissions /> hvis man vil bruge alle de sensors der er leveret som standard.

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.BLUETOOTH" />
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE"
/>
<uses-permission android:name="android.permission.CHANGE_WIFI_STATE" />
<uses-permission
android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"
/>
<uses-permission android:name="android.permission.ACCESS_COARSE_UPDATES"
/>
```

For en detaljeret beskrivelse af de enkelte permissions, se dokumentationen for Andorid SDK.

Tilpasning af rest endpoint

Hvis man vil ændre koden således det er en anden server man kommunikere med, skal man rette i sisdroid.framework.rest.SisRestClient, for der er en konstant REST_ENDPOINT som skal pege på hosten.

Appendix B - Reference hardware

SmartPhone

Vi har haft lånt to G1 telefoner af Århus Universitet til denne opgave, vi ønskede dog at udnytte de nyeste features der er i Android 2.1 (Android 2.1 var nyeste version, da vi startede på forløbet) - G1 telefonenerne er fra fabrikkens side installeret med Android 1.0, seneste officielle opdatering går til version 1.5.
Derfor har vi installeret en Custom Rom KiNgxKxlick-AOSP2.1 findes nemmest vha. google.com.

Serverside

Vores test backend blev installeret på en Windows 2008 Standard, med .NET 4.0 og en Microsoft SQL 2008 standard SQL.

Appendix C - Det der er i zipfilen

Med dette dokument følger en zipfil, indenholdende de vigtigste ting for opgaven.

Folder	Indenhold
URL	Er en gemt udgave af de kilder vi har angivet på internettet.
SisFramework	Kildekoden til frameworket, inclusive eclipse projekt filer
SisServer	Kildekoden til serveren
database	SQL script til oprettelse af databasen
prototype/BlueSocial	Koden til bluesocial
prototype/ SimpleCompass	Custom sensor fra indoor pathfinder eksemplet

Reference liste og ordbog

[1] Frank Allan Hansen og Niels Olof Bouvin - Context-aware Hypermedia in the Wild

[2] Anind K. Dey - Understanding and Using Context

[3] http://code.google.com/p/iphone-wireless/wiki/Apple80211Functions

[4] http://msdn.microsoft.com/en-us/library/ff402551%28v=VS.92%29.aspx

[5] http://martinfowler.com/bliki/InversionOfControl.html

[6] http://www.cisco.com/web/about/ac123/ac147/archived_issues/ipj_8-1/wireless_networks.html

[7] http://www.cisco.com/web/about/ac123/ac147/archived_issues/ipj_11-4/114_wifi.html

[8] Jeffrey Hightower og Gaetano Borriello - Location Systems for Ubiquitous Computing

[9] http://violet.sf.net/

[10] http://projects.gnome.org/dia/

[11] Internet Information Services (Tidligere Internet Information Server) - http://www.iis.net/

[12] ORM: Object Relational Mapping - http://msdn.microsoft.com/en-us/library/bb399567.aspx

[13] Plain Old CLR Object - http://msdn.microsoft.com/en-us/library/dd456853.aspx

[14] Text Templates - http://msdn.microsoft.com/en-us/library/bb126445.aspx

[15] Gamma et al, Design Patterns Elements of Reusable Object-Oriented Software p. 175

[16] Gamma et al, Design Patterns Elements of Reusable Object-Oriented Software p. 139

[17] Thomas Rissgaard Hansen: "Note about context-awareness (2008)"

[18] Henrik Bærbak Christensen, Flexible, Reliable Software p, 325

[19] http://developer.android.com/sdk/index.html SDK / plugin og installations vejledning til eclipse

[20] Riva & Kangasharju- Challenges and Lessons in Developing Middleware on Smart Phones

[21] http://www.engadget.com/2010/06/10/motorola-wants-a-2ghz-android-by-years-end-so-do-we

[22] http://developer.android.com/guide/practices/design/performance.html

[23] Henrik Bærbak Christensen, Flexible, Reliable software 2010

[24] http://www.bluetooth.com/SiteCollectionDocuments/Core_V40.zip

[25] "Specification of the Bluetooth system Part B: Baseband specification" The bluetooth special interest group. grouper.ieee.org/groups/802/15/Bluetooth/profile_10_b.pdf

[26] http://www.alexandra.dk/dk/projekter/Sider/Lodestar.aspx

[27] http://cs.unc.edu/~julia/accepted-papers/chi2010_microblogging_workshop_subramanian_march.pdf

[28] Portable hotspot - http://developer.android.com/sdk/android-2.2-highlights.html