CHALMERS

Parans Solar Lighting Fjärrkontroll

Android applikation för enklare styrning och felsökning av ljusinsamlingspaneler

Examensarbete inom Data- och Informationsteknik

HENRIK NUMÉ
TOBIAS NIELSEN

Institutionen för Data- och Informationsteknik
CHALMERS TEKNISKA HÖGSKOLA
Göteborg, Sverige 2016

EXAMENSARBETE

Parans Solar Lighting Fjärrkontroll

Android applikation för enklare styrning och felsökning av ljusinsamlingspaneler

HENRIK NUMÉ
TOBIAS NIELSEN

Institutionen för Data- och Informationsteknik
CHALMERS TEKNISKA HÖGSKOLA

Göteborg 2016

Parans Solar Lighting Fjärrkontroll
Android applikation för enklare styrning och felsökning av ljusinsamlingspaneler
HENRIK NUMÉ
TOBIAS NIELSEN

Examinator: Peter Lundin

Institutionen för Data- och Informationsteknik
Chalmers Tekniska Högskola
412 96 Göteborg
Telefon: 031-772 1000

Omslag:
Takmonterade ljusfångare av modellen SP3

Institutionen för Data- och Informationsteknik
Göteborg 2016

Parans Solar Lighting Fjärrkontroll
Android applikation för enklare styrning och felsökning av ljusinsamlingspaneler

HENRIK NUMÉ
TOBIAS NIELSEN
Institutionen för Data- och Informationsteknik, Chalmers Tekniska Högskola

Examensarbete

SAMMANFATTNING

Parans Solar Lighting är ett företag som levererar helhetslösningar för att leda in solljus in i byggnader. Detta gör de med hjälp av mottagare placerade på tak, kallad SP3 hädanefter, vilka följer solen under dagtid och sedan ställer sig riktade mot den del av horisonten där solen väntas att stiga. Dock kan SP3 tappa positionsdata. Då detta händer kommer den inte längre följa solen optimalt och kan i vissa fall förbli orörlig. Att felsöka och ta fram loggar ur en SP3 kräver viss tekniskt kunnande och det händer att det är Parans egna tekniker som måste åka ut för att extrahera dess data. För att underlätta service av dessa SP3 har Parans framfört en önskan om att få en fjärrkontroll utvecklad. De krav som finns är att den ska kunna utföra samtliga basinställningar av enheten samt kunna skicka log-data till Parans support. De krav som ställdes på enheten utöver funktionalitet var att hårdvaran skulle vara baserad på en Android-lösning, alternativt Raspberry Pi med embedded Linux, eller en likvärdig lösning. Det som projektet har resulterat i är en lösning baserad på Android.

Nyckelord: Android, seriell kommunikation, CP2102, fjärrkontroll

ABSTRACT

Parans Solar Lighting is a company that delivers complete solutions for leading sunlight into buildings. They achieve this by using roof mounted receivers, hereafter called SP3, which tracks and follows the sun during daytime and positions themselves facing the horizon where the sun is expected to rise the next day. However, the SP3 can lose its position settings. When this occurs it can no longer track the sun properly or it might stop functioning completely. To troubleshoot and extract logs from the SP3 requires a lot of technical knowledge and this forces the company to send their own technicians for support. To facilitate the maintenance of their product, Parans has inquired to have a remote control developed. The requirements include that the remote should be able to perform all the standard-configuration tasks and be able to send log data to Parans support department. The hardware should be based on either a Raspberry Pi or an Android device. The project resulted in a working solution based on an Android smartphone.

Keywords: Android, serial communication, CP2102, remote control

FÖRORD

Examensarbetet har utförts under våren 2016 för Data- och informationsteknik.

Vi skulle vilja framföra tack till Parans samt vår handledare där, Simon Larsson, för att vi fick möjlighet att utföra vårt examensarbete hos dem. Även väldigt tacksamma för den hjälp Simon har bidragit med vad gäller testning av applikationen samt värdefull återkoppling. Vi vill även tacka vår handledare på Chalmers Lars Svensson som har varit tålmodig med våra långa perioder av tystnad när vi har varit fullt fokuserade på projektet. Ett stort tack riktas också till Sakib Sistek som har varit behjälplig när vi har behövt.

Henrik Numé och Tobias Nielsen

Göteborg, Lindholmen, juni 2016.

BETECKNINGAR

AsyncTask – En Abstrakt klass i Android vilken utnyttjas för trådning.

CLI - Command-Line interface.

Espresso - Ramverk för automatiserad testning av användargränssnitt.

Junit - Testningsramverk för java klasser

Scrum - En agil arbetsmetod.

SP3 - Parans solpanel, tredje generationen.

USB - Universal Serial Bus

USB Host - USB enhet som agerar värd. Varje USB-anslutning måste ha exakt en värd.

USB OTG - USB On The Go, kontakt med extra pin som tillåter USB-enheten att agera Värd

VID/PID - Vendor ID och Product ID, används för att identifiera okända USB enheter.

Innehållsförteckning

1. Inledning

1.1 Bakgrund

Parans Solar Lighting är ett företag som säljer helhetslösningar för att leda in naturligt solljus i kontorslokaler. Detta sker med hjälp av en typ av optiska solfångare vilka är kopplade via fiberoptiska kablar till inomhusbelysningen.

I detta projekt behandlas företagets tredje generation solfångare av typen SP3 [1]. Dessa består i grund och botten av en uppsättning linser, monterade på ett motordrivet stativ som skall se till att linserna alltid är riktade mot och följer solens position på himmelen. Stativet kontrolleras av en inbyggd mikrokontroller som beräknar solens förväntade position utifrån aktuell tid och GPS-positionsdata.

För att koppla in sig till en SP3-enhet och konfigurera denna krävs det i dagsläget ett relativt stort tekniskt kunnande. Man behöver t.ex. en laptop med speciella USB-drivrutiner samt en terminalemulator installerad för kunna ge styrkommandon till enheten. Om t.ex. GPS-koordinater skall matas in får dessa sedan hämtas från en karttjänst och skrivas in manuellt via ett CLI.

Service av dessa SP3-enheter anses därför vara onödigt komplicerat. Det krävs ofta att Parans behöver skicka tekniskt kunnig personal för att felsöka och konfigurera felaktiga enheter även för relativt enkla ingrepp. Det kan t.ex. handla om att mata in rätt tid och position efter att denna har korrumperats på enheten.

Parans har uttryckt en önskan om att ta fram en lösning som förenklar interaktionen mellan slutanvändaren och dessa SP3-enheter. Detta är utgångspunkten för projektet.

1.2 Syfte

Syftet med projektet är att förenkla konfigurering och felsökning av SP3 enheter och därmed minskar behovet av att skicka tekniskt kunnig personal till kunder. Detta förväntas leda till minskade kostnader för support och underhåll för företaget.

1.3 Mål

Målet med detta projekt är att utveckla en fjärrkontroll som uppfyller krav på både användarvänlighet och funktionalitet. Kontrollen ska bland annat kunna visa enhetens status, aktuella inställningar för datum, tid och position samt spara och skicka händelseloggar till Parans. Den ska även kunna användas till att manuellt justera orienteringen av SP3 om den har ställt in sig felaktigt.

1.4 Avgränsningar

Inom ramen för detta examensarbete kommer inte andra enheter än Raspberry Pi eller Android att utvärderas som lämplig enhet för kontrollen [2][3]. Enheten ska kunna använda sig av GPS-data samt, kunna skicka information tagen ur SP3 till Parans support.

1.5 Begränsningar

För att kunna utnyttja alla önskade funktioner måste enheten vara mobil, ha en GPS-modul samt kunna skicka loggar, som är extraherade från en SP3, till Parans support. Kommunikation mellan SP3 och fjärrkontrollen kommer endast att ske via USB då andra anslutningsmöjligheter saknas.

1.6 Frågeställningar

Vilken hårdvara är mest lämplig för uppgiften när hänsyn tas till utvecklingstid, användarvänlighet, mobilitet samt pris? Hur genomförbart är det att upprätta kommunikation över USB mellan SP3-panelen och de olika hårdvarualternativen?

2. Teknisk bakgrund

2.1 Plattformar

2.1.1 Android

Android är ett operativsystem för telefoner och surfplattor, baserat på Linux kärnan [4]. Operativsystemet utvecklas av Android Inc, vilket är ett dotterbolag till Google Inc. Det har snabbt blivit det största operativsystemet inom smarttelefon marknaden sedan det släpptes 2008. Genom att vara baserat på öppen käll-kod har Android lyckats locka flera tillverkare till att anamma produkten. De får därmed möjligheten att tillföra egna ändringar av operativsystemet, ofta i formen av proprietär mjukvara. För att få tillgång till Google Play, vilket kräver proprietär mjukvara från Google, så måste tillverkaren uppfylla vissa krav.

Utvecklingsspråk för Android är främst Java och XML, men även C/C++, om behov skulle finnas.

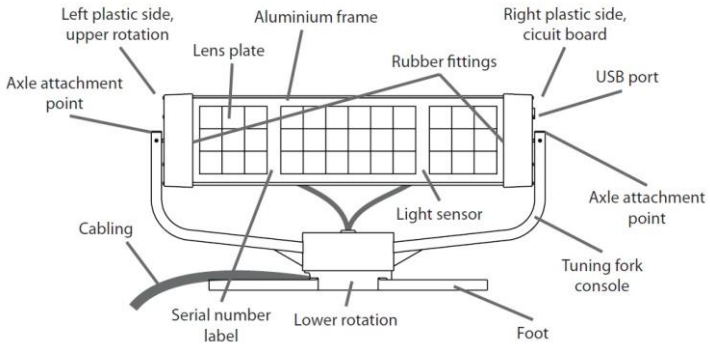
2.1.2 Raspberry Pi

Raspberry Pi är en enkortsdator utvecklad av Raspberry Pi Foundation. Den stöds av flera Linux-distributioner och Windows 10 IoT Core. Raspberry Pi är modulär och kan anpassas genom att utnyttja moduler för att bygga ut dess funktionalitet [5][6]. Ett populärt språk att använda för utveckling till Raspberry är Python.

2.1.3 SP3

SP3 är namnet på den tredje generationen av Parans' system med solpaneler. Dessa monteras på en ställning på byggnadens tak och fångar in ljus med hjälp av stora linser. Ljuset leds sedan in i byggnaden via kablar av optiska fibrer. SP3:an styrs av en mikrokontroller vilken är monterad på ett kretskort under enhetens kåpa. Denna styr bl.a. enhetens rotation och vinkling av linshöljet så att enheten kan följa solens position på himmelen.

Mikrokontrollern kan kommunicera via en USB-drivare som också är monterad på kretskortet. Denna USB-drivare är av modellen Silicon Labs CP2102 och har ett uttag av typen USB Type B [7]. För att kommunicera med SP3:an krävs det därför att man kopplar in sig med en enhet som agerar värd samt att enheten har drivrutiner som stöder CP2102.



Figur 2.1 SP3 enhetens delar

2.2 Val av plattform

De plattformar som var aktuella för projektet var alltså Android och Raspberry Pi. Dessa är inte helt jämförbara då Android är ett operativ för mobila enheter och Raspberry Pi är en enkortsdator. Det har tagits i beaktning att Android enheter har varierande hårdvaruspecifikationer och att Raspberry Pi är modulär och kan anpassas med de moduler som behövs för att uppfylla kraven från Parans.

Android fördelar och nackdelar:
En enhet kommer som en komplett produkt och behöver ej monteras utan endast installera vår programvara för att fungera. Vissa modeller kan sakna stöd för USB-OTG men då stöd har funnits inbyggt för Android sen version 3.1 är detta ovanligt [8]. Minimum versionen för att köra vårat program är 4.4, även känt som Kitkat.

Raspberry fördelar och nackdelar:
Det finns möjlighet att installera den officiella drivrutinen från Silicon Labs. Raspberry är modulär och det finns moduler för de funktioner som är önskade av Parans. Dock så måste Raspberry enheten monteras och få ett operativ installerat vilket inte Android behöver.

Efter att ha övervägt fördelar och nackdelar för respektive plattform samt testat bibliotek för serieportskommunikation för Android, så blev detta den plattform som fokus lades på i projektet.

2.3 Testning

För att säkerställa att applikationen fungerar korrekt samt att den gör det som förväntas av användaren har följande testmetoder lyfts fram som lämpliga i projektet.

2.3.1 UI Exerciser / Monkey

’UI/Application Exerciser Monkey’ är ett verktyg gjort för att utföra automatiserad testning av användargränssnitt. Verktyget kan utföra flera tusen kommandon inom loppet av några sekunder. I händelse av att ett fel påträffas avbryts testen och det som finns på stacken skrivs ut. Detta är ett bra stresstest för att se om användargränssnittet kan hantera stora mängder slumpvalda händelser utan att fastna [9].

2.3.2 Espresso

Espresso är ett Android-specifikt verktyg gjort för att testa användargränssnitt. Verktyget kan simulera användarens interaktion med applikationen och därefter ge möjlighet att testa enskilda UI-komponenters state eller metoder från involverade klasser [10].

2.3.3 Manuell

Testning av applikationen kommer även ske manuellt genom att gå igenom applikationens alla funktioner. Fokus för testarna, kommer ligga på att testa kända undantagsfall. Bland dessa finns fall då man försöker använda GPS utan att först fått tillåtelse från användaren. Det är även viktigt att testa vad som händer när USB-kabeln kopplas till eller från vid oväntade tillfällen.

2.4 Bibliotek

För att utveckla androidapplikationen har följande fristående bibliotek utnyttjats.

2.4.1 UsbSerial

UsbSerial är ett fristående bibliotek skapat av Felipe Herranz för att hantera serieportskommunikation i Android.

Biblioteket innehåller en UsbSerialDevice klass vilken kan ta emot Androids UsbDevice objekt och sedan jämföra PID/VID-nummer mot en lista med alla enheter som stöds och motsvarande interface-klasser. I vårt fall är CP2102:s VID/PID-nummer intressant. Klassen kan då skapa ett objekt utifrån rätt interface. Detta objektet används sedan för att skriva och läsa på USB porten. Man får även tillgång till att modifiera serieportens konfigurering vilket innefattar baudrate, paritet, flödeskontroll m.m.

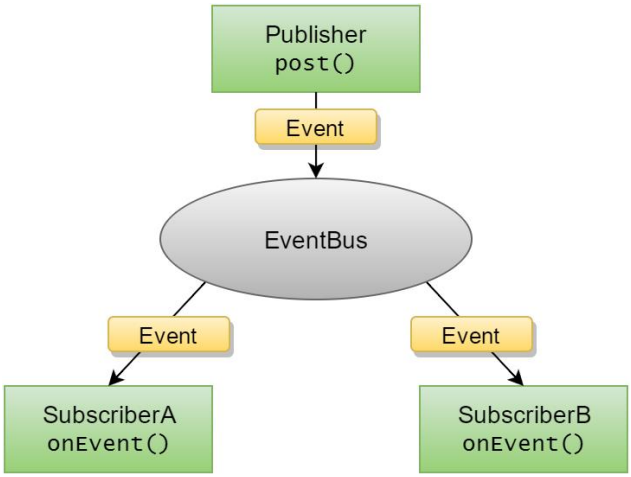
UsbSerial implementerar Producer/Consumer mönstret internt. Alla skrivningar till porten läggs i en buffer och skickas kontinuerligt. Skrivningar kan även utföras från flera trådar samtidigt utan att fel uppstår. Inkommande data från porten anländer via ett callback interface vilket gör att man kan undvika polling [11].

2.4.2 Eventbus

Greenrobot's EventBus är ett bibliotek som underlättar kommunikation mellan klasser genom att tillämpa publisher/subscriber mönstret på ett smidigt sätt. Biblioteket är ett mer lättanvänt alternativ till Androids BroadcastReceiver, speciellt när man vill skicka events från en bakgrundtråd, t.ex. en service, till flera olika View-komponenter, se Fig 4.3.

För att använda eventbussen behövs det först skapas en eventklass som representerar eventobjekten som skickas. Dessa kan vara rena java-klasser vilket ger stor frihet vid implementering. Därefter skall mottagarklassen registreras och avregistreras till eventbussen. Om mottagaren är en Android komponent, t.ex. ett Fragment, så måste

hänsyn tas till Androids livscykel. Sedan behöver man bara deklarera en onEvent(SomeEvent e) metod som annoteras med @Subscriber. Denna metod kommer sedan ta emot alla event som matchar event typen. För att posta ett event räcker det sedan med att man hämtar en referens till den statiska eventbussen och sedan kallar på post(new SomeEvent()). Detta går att göra från i princip var man vill i koden och alla subscribers som är vid liv kommer ta emot eventet [12].



Figur 2.2 Greenrobot EventBus flödesschema

3. Metod

3.1 Scrum

Den arbetsmetod som kommer att användas är Scrum. Det är en metod som lämpar sig väl för projekt där resultatet är viktigare än hur man kom dit. I det agila-manifestet så nämns några ledord för vad som är viktigast för Scrum [13]. Dessa är följande:

- Individer och anpassning framför processer och verktyg.
- Fungerande programvara framför omfattande dokumentation.
- Kundsamarbete framför kontraktsförhandling.
- Anpassning till förändring framför att följa en plan.

För att applikationen ska anses vara klar finns det ett antal uppsatta kriterier vilket skall uppfyllas när projektet är färdigt. Varje sprint är cirka en vecka lång, med utvärdering av föregående vecka i början av följande vecka. Det finns en produktägare, vilken även är vår handledare på Parans.

3.2 Versionshantering

Versionshantering kommer ske med Git, vilket arbetsgruppen redan har erfarenhet av [14]. Övrig dokumentation delas genom Google Drive [15].

3.3 Utvecklingsverktyg och testning

Utveckling av programmet kommer att ske med Android Studio vilken är Googles officiella IDE för Androidutveckling [16]. Android Studio har är i grund och botten Intellij IDEA med inbyggt stöd för Androidspecifika verktyg som t.ex. en hårdvaruemulator och testningsverktyg [17]. Det finns även likt Intellij, bra grafiskt stöd för vanliga Git-operationer.

Testning kommer initialt att ske manuellt då det krävs att Androidtelefonen är inkopplad till SP3-enheten för att kunna få intressanta resultat. Därmed är telefonens USB-uttag upptaget vilket förhindrar oss ifrån att köra tester från IDE:n. Senare när användargränssnittet är mer utbyggd, kan test skrivas som endast eller till stor del bara testar användargränssnittet. Några intressanta ramverk för detta är Espresso och UI Exerciser Monkey.

4. Genomförande

4.1 Kravspecifikation

De initiala kraven från Parans var relativt övergripande om vad som skulle omfattas rörande appens funktionalitet. Fokus lades på att skapa en lösning som underlättar service och support av SP3-enheter. Efter diskussioner med vår handledare på företaget kom vi fram till att följande funktioner skulle ingå:

- Skicka loggar till Parans support
- Ställa in GPS-position automatiskt
- Ställa in tid och datum automatiskt
- Ett intuitivt användargränssnitt

Det skall även finnas funktioner som motsvarar alla standard-kommandon i figuren nedan. Undantaget är altitude, azimuth, zeropos, setup samt restart, då dessa anses tillhöra avancerade inställningar.

Command	*Action*	*Comment*
info	Show info about the system. Eg. SW version and number of days with sun	
logga	Toggle between showing the numbers and not showing the numbers	
date	Show the current date and time	
date YYYY-MM-DD HH:MM:SS	Set the time and date	The time should ALWAYS be 24h-time and GMT
lat or latitude	Show the current latitude	
lat *ll.lll* or latitude *ll.lll*	Set the current latitude	3 or more decimals should be used
lon or longitude	Show the current longitude	
lon *ll.lll* or longitude *ll.lll*	Set the current longitude	3 or more decimals should be used
run u	Run the unit up	
run d	Run the unit down	
run r	Run the unit right	
run l	Run the unit left	
run stop	Stop the unit's movement	
run auto	Run the unit automatically	
altitude	Show the current altitude position	
azimuth	Show the current azimuth position	
zeroposx	Show the current sensor zero position	
zeroposy	Show the current sensor zero position	
setup	Setup the unit for a new installation or full reset	
restart	Equivalent to plugging out and in the SP3 receiver.	

Figur 4.1 SP3 kommando lista

4.2 Utvecklingsprocessen

Projektet delades upp i ett flertal sprintar, vilka var cirka en vecka lång. Nedan presenteras sprintarna samt resultat från respektive sprint. Prototypen är det som beaktas som första iterationen i projektet, varje efterföljande sprint betraktas som en ny iteration, då ofta inte bara ny funktionalitet har införts utan kod har även blivit refaktorerad.

4.2.1 Upprätta kommunikation

Då målplattformen var bestämd, var det första målet att upprätta kommunikationen mellan SP3 och en Android-telefon. Hela projektet var beroende på att kommunikationen mellan enheterna var stabil och tillförlitligt.

Vid eftersökningar fanns några bibliotek på GitHub som var intressanta för uppgiften. Efter vidare efterforskningar kunde det konstateras att Felipe Herranz bibliotek, UsbSerial, var det som passade bäst för de krav som var uppsatta [18]. Det största argumentet för detta var att det finns en Terminal-app, DroidTerm, utvecklad av Felipe Herranz som använder sig utav samma bibliotek [19]. Denna app kunde vid testning kommunicera med SP3-enheten. Appen var dock inte open-source däremot fanns det ett bra demonstrationsexempel man kunde utgå från.

4.2.2 Prototyp

Med hjälp av detta bibliotek och exempel-appen började arbetet med att utveckla en första prototyp vilken skulle kunna känna igen SP3-enheten och skicka styrkommandon.

Målet med denna prototyp, förutom att tidigt demonstrera funktionalitet, var att bli familjära med utvecklingsmiljön till Android och SP3:ans CLI.

Det konstaterades även att UsbSerial-biblioteket krävde att nivån på 'minimum target SDK' behövde höjas till 19, från det initiala 17, för vara kompatibel med applikationen. Detta gör att applikationen kräver att telefonen kör Android version 4.4 Kitkat eller nyare vilket omfattar cirka 76% av alla Android enheter på marknaden [20].

Figur 4.2 Prototyp

Prototypens användargränssnitt håller i detta skede en textvy, vilken visar de log-meddelanden SP3 kontinuerligt sänder över bussen. Knapparna implementerades i detta moment som dummies för att senare koppla dessa till metoder.

4.2.3 Implementering av styrkommandon

Tack vare det färdiga UsbSerial-biblioteket var det relativt enkelt att skicka kommandon till SP3-enheten. För att läsa och skriva till telefonens USB-port skapades en UsbService som förlänger Androids egna Service klass. Denna Service bands sedan till en

MainActivity vilken håller knappar för respektive kommando. När en knapp trycks ner kallas UsbServicens skrivmetod med själva kommandosträngen som argument, se Fig 4.3. Svaret från servicen skickades sedan med hjälp av en BroadCastReceiver tillbaka till ett textfält i MainActivity. Detta fungerade, dock visade sig en bug som gjorde att svaren var opålitliga. Det hände t.ex. ofta att svaren blev av typen "Unknown command" trots att rätt kommando var angivet. Källan till buggen visade sig finnas i SP3-enhetens läsbuffer vilken var känslig mot störning och fylldes ofta upp med skräptecken mellan läsningar.

För att lösa detta problem skickades helt enkelt samma kommando igen ifall svaret innehöll ”Unknown command” som substräng. Denna lösning var acceptabel till prototypen men till den slutgiltiga versionen skapades sedan ett bättre system med en egen klass ‘CommandHandler’ som hanterade alla kommandon och svarsmeddelanden, se Fig 4.4.

```
public void onInfoButton() {
    String command = "info\r";
    if (usbService != null) { // if UsbService was correctly binded, Send command
        logTextView.append("# " + command + "\n");
        usbService.write(command.getBytes());
    }
}
```

Figur 4.3 Info-command före CommandHandler

```
infoButton.setOnClickListener((v) -> {
        Log.d(TAG, "infobutton pressed");
        EventBus.getDefault().post(new CommandEvent("info",
                CommandEvent.RESPONSE_TYPE_INFOEVENT,
                CommandEvent.TARGET_INFO_FRAGMENT));
});
```

Figur 4.4 Info-command med CommandHandler

4.2.4 Förbättrad GUI och refaktorering

En del av målet var att skapa en lättanvänd applikation som gör det möjligt för både kunder och supporttekniker att på ett enkelt sätt interagera med SP3-enheterna efter att eventuella fel har inträffat. Därför har mycket fokus lagts på att göra alla funktioner så enkla och lättanvända som möjligt.

För att uppnå målet planerades det tidigt att försöka abstrahera bort så mycket som möjligt av det tidigare CLI-baserade användargränssnittet. I samarbete med vår handledare på Parans togs det fram ett antal huvudanvändarfall. Vilka motsvarar de vanligaste operationerna som tidigare har behövts utföras på SP3-enheterna. Dessa huvudanvändarfall fick sedan stå som grund för resten av applikationens design och uppbyggnad. Varje användarfall fick bli en egen aktivitet med en egen layout som bara visar just den informationen och de knappar som krävs för att användaren skall kunna utföra uppgiften.

De fyra huvudanvändarfallen är följande:

Control - Användaren vill kunna manuellt styra SP3-enhetens riktning, samt sätta enheten i "run auto" läge.

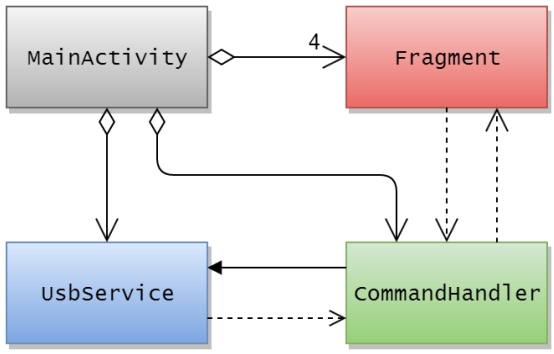
Settings - Användaren vill kunna inspektera och ändra enhetens aktuella tid, datum och GPS-positions inställningar.

Log - Användaren vill kunna inspektera systemloggar och sedan skicka dessa tillsammans med all annan tillgänglig enhetsinformation till Parans supportavdelning.

Info - Användaren vill kunna inspektera enhetens aktuella status.

Dessa användarfall realiserades genom att i huvudaktiviteten (MainActivity) införa en ViewPager-komponent innehållandes fyra fragment. Varje fragment är en vy som är ansvarig för respektive användarfall och innehåller alla nödvändiga knappar och textfält. ViewPager är en Android-komponent som på ett smidigt sätt låter användaren bläddra mellan fragment, antingen genom att svepa i sidled eller att trycka på flikarna i toppen. En sak som dock måste tas i beaktning är att det aktuella fragmentets grannar också återupptar sin livscykel även om dom är ur fokus.

Det skapades även en CommandHandler klass vars uppgift var att agera mellanhand mellan UsbService och alla fragment. Detta extra lager minskar även kopplingen mellan vyerna och service-delen, se Fig 4.5. En av fördelarna med detta är att UsbService blir helt oberoende av fragmentens livscykler vilket minskar risken för buggar vid oförutsedda händelser.



Figur 4.5 Förenklat UML-diagram över applikationen

Kommunikationen mellan fragmenten och CommandHandler och UsbService sker via en eventbuss vars flöde beskrivs i senare kapitel.

Huvudaktiviteten är också som tidigare ansvarig för att starta en UsbService som kan känna igen när USB-enheter ansluts eller kopplas ifrån. När något av detta sker visas Toast-meddelanden med servicens status. CommandHandler uppdateras även med en ny referens till UsbService.

Även andra delar av appen blev refaktorerade i detta steg av utvecklingen. Lokalisering, vilket initialt hämtade positionering på huvudtråden, refaktorerades för att utnyttja AsyncTask klassen och dess metoder för att utföra positionering i bakgrunden.

```
setPosButton.setOnClickListener((new View.OnClickListener() {

    @Override
    public void onClick(View v) {
        logTextView.append("<setPosButton>\n");
        checkPermission();
        getLocationManager();
        getLocation();
        if(!setPosButtonBoolean){
            showRequestDialog();
        }else{
            onLocationChanged(location);
            logTextView.append("lat: " +latitude + "\n");
            sendCommand("lat " +latitude);
            logTextView.append("lon: " +longitude + "\n");
            sendCommand("lon " +longitude);
        }
    }
}));
```

Figur 4.6 Positionering första implementering

```
syncButton.setOnClickListener((v) → {
        Log.d(TAG, "syncButton pressed");

        Toast.makeText(fcontext, "Syncing the device, please wait until completed ...", Toast.LENGTH_SHORT).show();
        if (setPosButtonBoolean && buttonAvailable) {
            buttonAvailable = false;
            new AsyncLocation().execute("");
        }else{
            Log.d(TAG, "AsyncLocation is currently running");
        }
});
```

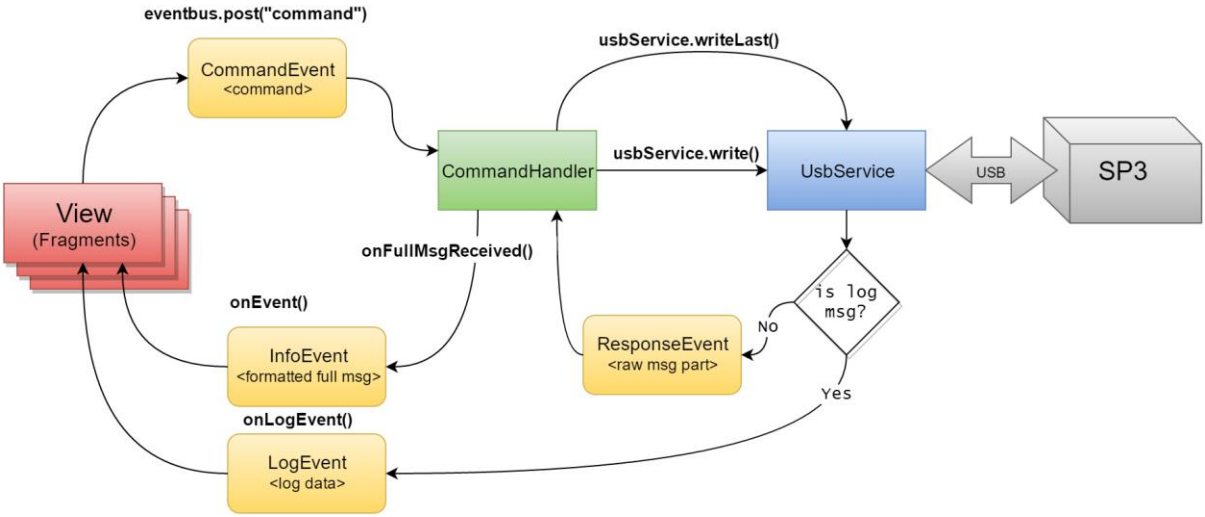
Figur 4.7 Positionering efter refaktorering

Den refaktorerade metoden för lokalisering, skapar en temporär lokaliserings-manager vilken hämtar platsdata och därefter avslutar managern. I postExecute uppdateras fält i vyn och metoder för synkning med SP3 startas.

4.3 Stödklasser

4.3.1 CommandHandler och events

CommandHandler fungerar som mellanhand mellan fragmenten och UsbService. Dess huvuduppgift är att ta emot CommandEvents från Fragment och vidarebefordra dessa till UsbServicen och sedan vänta på ett helt svarsmeddelande. Ett CommandEvent-objekt består av en konstant 'responseTarget' som indikerar målfragmentet för svaret samt en sträng med själva SP3-kommandot som skall köras.



Figur 4.8 CommandHandler Flödesschema

Ett svarsmeddelande från SP3:an består av en sträng vars slut indikeras med ett '#' tecken. All data från SP3 skickas till UsbService i form av en byteström. Denna kan delas upp i ett eller flera bytearrayer när den tas emot av UsbService beroende på hur långt meddelandet är. Dessa bytearrayer skickas sedan som ett ResponseEvent till CommandHandler som i sin tur väntar på att den får ett ResponseEvent som innehåller ett '#'-tecken. Då vet CommandHandler att ett helt svarsmeddelande är mottaget.

Då SP3 även kan skicka loggar till UsbService måste de särskiljas från ett vanligt svarsmeddelande. Detta uppnås genom parsning av dataströmmen och när UsbService har identifierat meddelandet som antingen en log eller ett svars -meddelande passerar den resultatet antingen direkt till LogFragment eller till CommandHandler.

Om svarsmeddelandet innehåller substrängen "Unknown command" försöker CommandHandler skicka samma kommando en gång till vilket hjälper i fallet då SP3:an har skräp i sin inbuffer och därmed inte känner igen kommandot trots att det är korrekt.

När ett helt giltigt meddelande är mottaget i CommandHandler skapas sedan ett sista [fragmentnamn]Event från 'responseTarget' som är skräddarsytt för det Fragment som tar mot det. I figuren ovan kommer t.ex. InfoEvent tas emot av InfoFragmentet. Dessa [fragmentnamn]Event är också ansvariga för att parsa ut all information ur svaren och presentera den på ett smidigt sätt via egna get-metoder.

4.3.2 GpsLocation

Denna klass håller de metoder som har med positionering att göra i applikationen. I fragmentet Settings så skapas ett objekt av GpsLocation tillsammans med en LocationListener, vilken sedan passas in till GpsLocationen objektet, när användaren vill hämta nuvarande position. När dessa steg har skett så begär enheten en positionsuppdatering från GPS. När positionsdata har mottagits och hanterats, så dödas både GpsLocation objektet och LocationListener objektet. Detta för att minska onödigt

batteridränage. Ytterligare för att minska mängden data som applikationen begär från GPS så använder vi oss av det mindre precisionssäkra requestSingleUpdate(), vilket vid testning gett en varians på 50-70m.

4.3.3 Sp3Model

En klass med statiska metoder för att temporärt lagra data från SP3. Denna klass håller info, tid och position i form av strängar samt log data som objekt. Valet att hålla log data som objekt är på grund att det gav en bättre implementering när man skulle visualisera det i log vyn.

4.4 Testning

Testning av applikationen skedde i samband med att den andra iterationen av mjukvaran blev färdig.

4.4.1 UI Excerciser / Monkey

Testen som genomfördes skickade från 500 till 5000 kommandon under loppet av några sekunder. Detta test började köras när det nya grafiska gränssnittet var klart.

4.4.2 Manuell testning

Testarna har fokuserat på att testa de kända fall som kan orsaka krascher. Bland dessa så har fel rörande, avbrott av anslutning under pågående transaktion samt fel när användaren har givit tillstånd för GPS med lokalisering avstängt, blivit funna och åtgärdade.

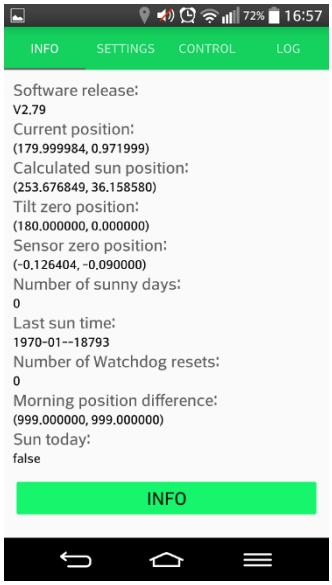
5. Resultat

I beaktning till de mål som sattes upp i början av projektet resulterade examensarbetet i en applikation vilket har ett minimalistiskt gränssnitt och delar upp funktionalitet i flera vyer.

5.1 Vyer

5.1.1 Info

Info är startskärmen för applikationen. Här får användaren grundläggande information om enheten man är ansluten till. Synkning med SP3 sker med ett knapptryck på infoknappen. Data som inhämtas från SP3 lagras även temporärt i en modell av SP3, för att kunna visa samma data vid återskapande av vyn som den hade vid innan.

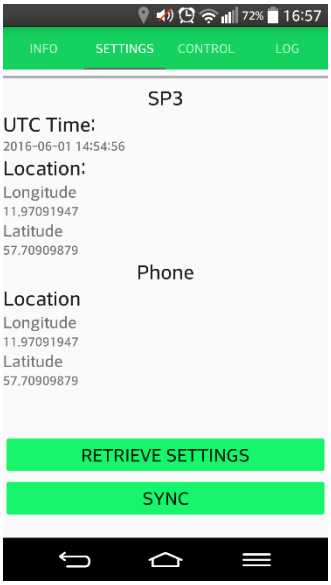


Figur 5.1 Info Fragment

5.1.2 Settings

Denna vy visar SP3:ans aktuella tid i UTC format och dess aktuella GPS position. Informationen hämtas genom att trycka på 'Retrieve settings'-knappen. Då startas en bakgrundstråd som skickar ett antal `CommandEvents` med "date", "lon" och "lat" kommandon.

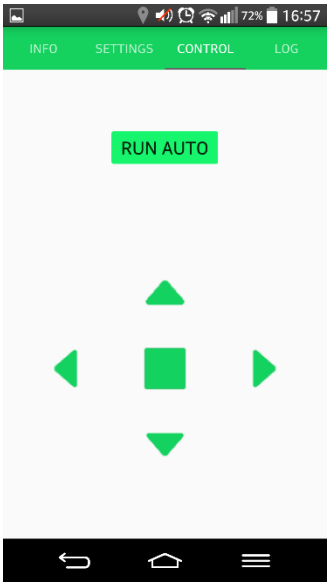
Det finns även möjlighet att överföra telefonens aktuella tid och GPS information till SP3:an. Detta sker genom att trycka på Sync-knappen. Då startas en överföringsprocedur vilken håller användaren informerad om hur det går med hjälp av en Android progressbar placerad i den övre delen av skärmen. När synkroniseringen är klar visas en kort Toast-notifikation.



Figur 5.2 Settings Fragment

5.1.3 Control

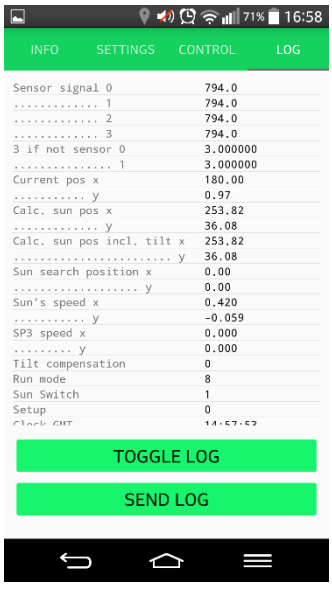
Denna vy låter användaren ta direkt kontroll över SP3. Detta låter användaren att manuellt justera SP3 riktning så den står riktad mot solen. När korrigering väl är skedd så trycker man på ‘Run auto’ för att SP3 ska återgå till auto läget. Knappen i mitten är en Stopp-knapp som avbryter både manuella kommandon och ”Run auto” läget.



Figur 5.3 Control Fragment

5.1.4 Log

Visar log data från enheten då logkommandot är aktivt. Från denna vy kan man även skicka data från SP3 till Parans support. Viktigt är att logkommandot stängs av om man ska utföra andra kommandon på SP3, då log flödet från SP3 kan störa övriga funktioner.



Figur 5.4 Log Fragment

5.2 Testning

Applikationen har testats manuellt och med UI exerciser, dock har ej espresso testning hunnit genomföras inom utsatt tid. För de manuella testerna så har testning av kända extremfall utförts. En UI exerciser har stress-testat användargränssnitt och har körts med 5000 slumpade kommandon, utan att några problem uppstod.

Det har konstaterats under projektets gång att Android enheter är väl lämpade för det planerade användningsområdet. Med tanke på Androids stora marknadsandel, vilken ligger på runt 83%, kommer den tilltänkta användarna med stor sannolikhet redan ha tillgång till Android-telefoner [21]. Användarna är också vana vid att både installera och nyttja tillhörande mjukvara. Detta gör att tröskeln för nya användare blir låg vilket har varit huvudmålet för projektet.

En nackdel är dock som tidigare nämnt att alla Android-telefoner inte har stöd för USB-Host. Även om själva Android-OS versionen är tillräckligt hög, 3.1 eller högre, så kan telefonens hårdvara vara icke-kompatibel. Efter att ha undersökt diverse användarrapporterade listor över Host-kompatibla enheter anser vi dock detta vara relativt ovanligt. I princip alla populära modeller finns rapporterade som fungerande [22][23].

I verkligheten testades applikationen på fyra olika telefonmodeller varav en inte fungerade trots att den var listad som fungerande. Felet förklarades dock med att telefonen, som legat oanvänd en längre tid, hade en äldre Android version.

6. Slutsatser

6.1 Kritisk diskussion

Vi har utvecklat en applikation till Android med vilken man kan utföra de vanligaste inställningarna Parans samt deras kunder brukar behöva utföra på SP3. Den hade initialt även andra funktioner som diskuterades och är sådant som ligger kvar som potentiell vidareutveckling av applikationen. Utvecklingen gick ungefär som planerat, åtminstone fram tills tredje iterationen skulle implementeras. När vi arbetade med de tillägg och ändringar som skulle införas till den iterationen, fann vi det lämpligt att även arbeta med hur applikationen hanterar kommunikationen med SP3.

För att återvända till frågeställningarna vi ställde i början av rapporten, så fann vi att om än enhetskostnaden kan vara högre för en Android enhet, så kommer kostnaden för montering samt installation av operativsystem till en Raspberry Pi sannolikt överskrida detta i längden. Med detta så är en lösning till Android bättre ekonomiskt gångbar då installation av ett program tar betydligt mindre tid.

Utveckling till Raspberry respektive Android betraktar vi som likvärdig. Vad gäller Raspberry finns det bibliotek som är tillgängliga från Silicon Labs för deras CP2102 krets, men tredjepartsutvecklare har även implementerat stöd åt Android. Vi har visserligen inte någon erfarenhet av att utveckla till Python sen innan men, eftersom det är ett objektorienterat språk samt har en relativt låg inlärningskurva, skulle det inte utgjort något problem.

För mobilitet så är Android segraren i vår mening då en Raspberry med moduler samt skal inte får samma snäva dimensioner som den integrerade lösningen som en Android enhet erbjuder. Dessutom så är en telefon något som man nästan alltid har på sig och igång medans en specialiserad enhet som Raspberry Pi skulle vara något man endast tog fram vid behov.

I användarvänlighet så måste vi göra några antaganden: applikationerna är likvärdiga i funktionalitet, och att enheten används ej samtidigt till andra ändamål än det som är avsett för projektet. Med dessa antaganden är plattformarna nästan likvärdiga. Dock ter sig inte verkligheten så. Android-lösningen går att implementera på kundens telefon till skillnad från Raspberry lösningen vilket levereras på egen hårdvara.

Kommunikation var den del vilken var mest kritisk för Android. Vi visste redan att kommunikation skulle fungera med Raspberry, då det finns officiellt stöd för Linux distributioner. Dock så fann vi tidigt under inledande undersökningar för projektet att det fanns applikationer vilka påstod att de hade den funktionalitet vilken vi önskade. Efter att vi fick kommunikation att fungera med DroidTerm, så visste vi att Android skulle fungera för vår tänkta lösning och det blev den plattform vilket vi la vårt fokus på.

Den planering vilken vi arbetade efter har visat att vi har underskattat tidsåtgången för testning och rapportskrivning. Det är något vi finner tråkigt men, tid har lagts åt att testa de kritiska delarna och de fel vi har funnit har blivit åtgärdade. Något vi kommer att göra till nästa gång är nog att planera utvecklingen med testning i större fokus hädanefter. Vi har funnit att vi har lärt oss mycket om Android och hur utveckling till plattformen kan

ske. Med det sagt så finns det fortfarande oerhört mycket kvar att lära sig om utveckling till denna plattform.

6.2 Vidareutveckling

Det finns flera funktioner som kan läggas till eller som kan utvecklas vidare i appen. En funktion som önskades från Parans var möjligheten att även kunna skriva in mer avancerade kommandon. Detta kan lösas genom att införa en terminal-liknande aktivitet där användaren är fri att skriva in kommandon och ta emot svar från SP3 i råtext-format.

En annan önskvärd funktion är att vid uppstart notifiera användaren ifall telefonen inte stöder USB-Host. I dagsläget händer ingenting ifall en icke-kompatibel telefon ansluts till SP3-enheten, vilket tyvärr inte kan anses vara särskilt användarvänligt.

Det skulle även gå att göra positionsvisningen tydligare genom att använda Google Maps karttjänst. Användaren kan då se var denne befinner sig på kartan och därmed verifiera manuellt att den inhämtade positionen är korrekt innan den skickas till SP3-enheten.

Till sist kan det även undersökas ifall det är möjligt att uppdatera SP3-enhetens firmware direkt från applikationen.

6.3 Miljöaspekter

Ur en miljösynvinkel kan applikationen göra viss skillnad genom att minska antalet service-resor som krävs för att underhålla SP3-enheter. T.ex. kan log data bli förmedlad från kundanläggningar direkt till Parans, utan att en tekniker behöver åka ut. Detta kan spara in på koldioxidutsläpp, särskilt om det är långa flygresor som kan förhindras. Distribution av applikationen kan även göras via existerande infrastruktur, Google Play eller genom att man skickar en installationsfil, vilket har en minimal inverkan på miljön.

7. Litteratur referenser

[1] Parans Solar Lighting, “Paranssystemet,” 2016. [Online]. Available: http://www.parans.com/swe/sp3/. [Accessed: 15-Apr-2016].

[2] “Raspberry Pi,” 2016. [Online]. Available: https://www.raspberrypi.org/. [Accessed: 04-Apr-2016].

[3] “Introduction to Android,” 2016. [Online]. Available: https://developer.android.com/guide/index.html. [Accessed: 04-Apr-2016].

[4] “Wikipedia Android.” [Online]. Available: https://en.wikipedia.org/wiki/Android_(operating_system). [Accessed: 08-Jun-2016].

[5] Microsoft, “Windows 10 IoT Core,” 2016. [Online]. Available: https://developer.microsoft.com/sv-se/windows/iot/iotcore. [Accessed: 08-Jun-2016].

[6] Raspberry-Pi Foundation, “Raspberry-Pi OS Images,” *2016*. [Online]. Available: https://www.raspberrypi.org/downloads/. [Accessed: 08-Jun-2016].

[7] “CP210x USB to UART Bridge VCP Drivers,” 2016. [Online]. Available: https://www.silabs.com/products/mcu/Pages/USBtoUARTBridgeVCPDrivers.aspx. [Accessed: 04-Apr-2016].

[8] Android Inc., “Android 3.1 API.” [Online]. Available: https://developer.android.com/about/versions/android-3.1.html. [Accessed: 20-May-2016].

[9] Google, “UI/Application Exerciser Monkey,” 2016. [Online]. Available: https://developer.android.com/studio/test/monkey.html. [Accessed: 08-Jun-2016].

[10] Google, “Espresso Testing,” 2016. [Online]. Available: https://developer.android.com/training/testing/ui-testing/espresso-testing.html. [Accessed: 08-Jun-2016].

[11] Felipe Herranz, “UsbSerial: A serial port driver library for Android v3.0,” 2015. [Online]. Available: https://felhr85.net/2014/11/11/usbserial-a-serial-port-driver-library-for-android-v2-0/. [Accessed: 20-May-2016].

[12] “EventBus: Events for Android | Open Source by greenrobot,” 2016. [Online]. Available: http://greenrobot.org/eventbus/. [Accessed: 18-May-2016].

[13] Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, and Dave Thomas, “Manifest för Agil systemutveckling,” 2001. [Online]. Available: http://www.agilemanifesto.org/iso/sv/. [Accessed: 20-May-2016].

[14] S. Chacon and B. Straub, *Pro Git*. Berkeley, CA: Apress, 2014.

[15] “Wikipedia, Google Drive,” 2016. [Online]. Available: https://en.wikipedia.org/wiki/Google_Drive. [Accessed: 08-Jun-2016].

[16] Google Inc, “Android Studio,” 2016. [Online]. Available: https://developer.android.com/studio/index.html. [Accessed: 16-Jun-2016].

[17] H. D’Arcy, “IntelliJ IDEA,” *RefCardz*, 2009.

[18] Felipe Herranz, “UsbSerial,” 2014. [Online]. Available: https://github.com/felHR85/UsbSerial. [Accessed: 29-Mar-2016].

[19] Felipe Herranz, “DroidTerm,” *2016-05-25*, 2016. .

[20] Android inc., “Android Dashboards,” *2016-05-02*, 2016. [Online]. Available: https://developer.android.com/about/dashboards/index.html. [Accessed: 03-May-2016].

[21] IDC, “Smartphone OS marketshare,” *IDC*, 2016. [Online]. Available: http://www.idc.com/prodserv/smartphone-os-market-share.jsp. [Accessed: 25-May-2016].

[22] “List of OTG Supported Devices — SYMLIS.” [Online]. Available: http://www.symlis.com/blog/2015/1/12/list-of-otg-supported-devices. [Accessed: 30-May-2016].
[23] “OTG Compatibility List.” [Online]. Available: http://www.corsair.com/en-us/landing/otg-compatibility-list. [Accessed: 30-May-2016].

Bilaga

Parans kravspecifikation.

FJÄRRKONTROLL FÖR STYRNING AV SOLLJUSSYSTEM

Sammanfattning

Framtagning av fjärrkontroll för solljusenhet bestående av användarinterface med visning av information och möjlighet att kontrollera och göra inställningar på enheten. Plattform beslutas efter utvärdering; alternativ inkluderar Android och inbäddade Linuxsystem.

Beställare

Parans Solar Lighting är ett företag som tillverkar solljussystem för belysning. Systemen består av en linsenhet som monteras på taket och följer solens rörelse under dagen. Linserna fokuserar solljuset mot fiber som leder ljuset in i byggnaden där fiberändarna är monterade i armaturer och sprider ljuset i rummet.

Bakgrund

Service av Parans enheter har historiskt krävt specifik mjukvara och drivrutiner samt mycket god datorvana, vilket har gjort inlärningskurvan för service av dem onödigt hög. Även för enklare ingrepp har man behövt skicka tekniskt kunnig personal, vilket är dyrt både i arbetstid och transporter.

Syfte och mål

Syftet med detta projekt är att ta fram en fjärrkontroll för att underlätta styrning av enheterna för slutanvändare. Denna kontroll skall ha någon form av skärm för indikering av enhetens status, och knappar eller touchskärm för att skicka kommandon och inställningar till enheten. Den skall också vara utrustad med GPS för att automatiskt ställa in tid och position på enheten. Kontrollen ansluts med USB till enheten.

Kontrollen skall antingen vara baserad på Android vilket ger slutanvändare möjlighet att använda valfri Andoid-baserad enhet, eller på en inbäddad plattform med antingen touchskärm eller skärm och fysiska knappar.

Kontaktpersoner

Simon Larsson (simon.larsson@parans.com)

Karl Richard Nilsson (karl.nilsson@parans.com)