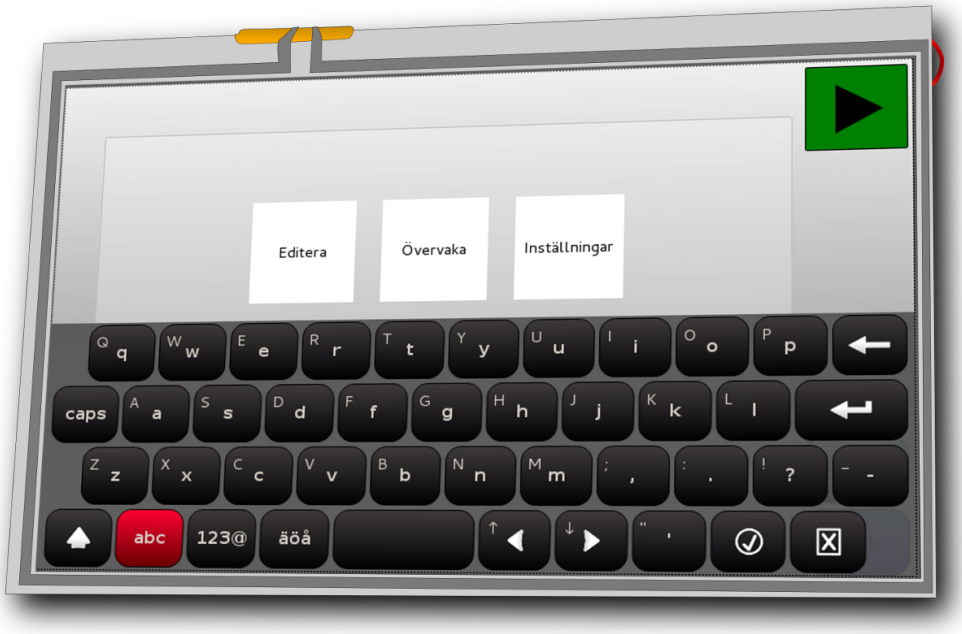
CHALMERS



Utveckling av programvara för kontrollenhet till en industriell bläckstråleskrivare

Kandidatarbete inom data- och informationsteknik

ROBERT COLLIN-KARLSSON
FRITIOF HEDMAN
OLA JEPPSSON

Institutionen för data- och informationsteknik

CHALMERS TEKNISKA HÖGSKOLA
Göteborg, Sverige 2012
Kandidatarbete/rapport: 2012:002

Abstract

This report discusses the development process of a computer program for controlling an industrial inkjet system. The project is based on outcome targets specified by the originator, Inkit AB. These include the functionality that the software should have. Requirements also stated that the software should have support for internationalization, and that it should have a user-friendly user interface. The software is targeted for an ARM-based embedded system. A specification was developed for the software, where demands were made for defined parts of the software architecture. The requirements were implemented by code, preferably based on C + +. The choice of development environment Qt Quick for the user interface enabled internationalization. The requirement regarding ease of use were met by the application of the principles of coherence and consistency. The software was given experimental support for the ARM structure through the choice of development environments, but the report does not cover the implementation of the software on the embedded system.

Sammanfattning

Denna rapport behandlar utvecklingen av en programvara för styrning av ett industriellt bläckstråleskrivarsystem. Projektet är baserat på effektmål ställda av beställaren Inkit AB. Dessa omfattar funktionalitet som programvaran ska ha. Krav fanns även på att programvaran skulle ha stöd för internationalisering, samt att den skulle ha ett användarvänligt gränssnitt. Det inbyggda systemet som ska köra programvaran har en ARM-arkitektur. En kravspecifikation togs fram för programvaran, där krav ställdes på definierade delar i programvarans arkitektur. Kraven implementerades med hjälp av kod, företrädesvis baserad på C++. Valet av utvecklingsmiljön Qt Quick för användargränssnittet möjliggjorde internationalisering. Kravet gällande användarvänlighet uppfylldes genom att principerna om konsekvens och samstämmighet applicerades. Programvaran gavs experimentellt stöd för ARM-arkitekturen genom val av utvecklingsmiljö. Dock omfattar inte rapporten implementering av programvaran på det inbyggda systemet.

Förord

Rapporten är ett kandidatarbete vid Institutionen för Data- och informationsteknik, Chalmers Tekniska Högskola, och utfördes våren 2012. Projektgruppen önskar att tacka Lennart Hansson för den tid han lagt ner med att handleda arbetet i rätt riktning. Projektgruppen önskar också att tacka Per Nilsson och de övriga anställda vid Inkit AB.

Kandidatgruppen
Göteborg 2012

Innehåll

Förkortningar

ACID Atomicity, Consistency, Isolation, Durability

API Application Programmers Interface

ARM Advanced Risc Machine

ASCII American Standard Code for Information Interchange

BSD Berkeley Software Distribution

COLIBRI Qt Quick COmponent LIBRary

FIFO First In, First Out

GNU GNU is not Unix

GPL GNU Public License

GUI Graphical User Interface

HTA Hierarchical Task Analysis

HTTP HyperText Transfer Protocol

JSON JavaScript Object Notatation

LGPL Lesser GNU Public License

LVDS Low Voltage Differential Signaling

PLC Programmable Logic Controller

QML Qt Meta Language

REST REpresentational State Transfer

RPC Remote Procedure Call

USB Universal Serial Bus

WYSIWYG What You See Is What You Get

XML eXtensible Markup Language

Definitioner

Beställare Inkit AB

Driftstörning Strömavbrott, felaktig avstängning och dylikt.

GNU/Linux Ett operativsystem baserat på en kärna skriven av Linus Torvalds, med program ifrån GNU-projektet.[1]

Inbyggt system Det inbyggda systemet utgörs av ARM-baserad dator med en pekskärm

Kontrollenhet Utgörs av det inbyggda systemet i kombination med programvaran

Public Domain Allmän egendom

[1] www.gnu.org

1 Inledning

Dagens samhälle är i en mängd olika sammanhang beroende av information. Exempelvis behövs tydlig information för att logistiken av en vara i alla led mellan producent och konsument ska vara effektiv. Ett steg är att märka varorna så de lätt kan identifieras. En av de tekniker som används inom industrin för att lösa behovet av märkning är bläckstråleskrivare.

Bläckstråleskrivartekniken bygger på att små bläckdroppar sprutas mot ytan som ska märkas. Inom industrin finns flera tekniker för att göra just detta. En av dem är *drop-on-demand*, där varje enskild droppe genereras direkt när den ska skjutas mot ytan.

1.1 Bakgrund

Inkit AB är ett ungt företag som har varit aktivt i drygt två år, men med personal som sedan tidigare har en bred erfarenhet av branschen och framförallt den svenska marknaden. Företagets huvudsakliga verksamhet är i dagsläget tillverkning av industriellt bläck.

Sedan drygt ett år tillbaka har företaget, för att bredda sin verksamhet, arbetat med att utveckla en egen bläckstråleskrivare. Skrivaren bygger på drop on demand-tekniken, och mer specifikt *valvejet*-tekniken. Valvejet-tekniken bygger på att bläcket hålls tryck-satt. Sedan används ventiler för att generera droppar som skjuts mot ytan som ska märkas.

Egenskaper som prioriterats i produktutvecklingen är att anläggningen ska vara underhållsfri, samt att pumpsystemet ska vara av sugande typ. Det medför lägre krav på emballeringen av bläcket eftersom denna inte måste stå emot tryck. Detta möjliggör i sin tur bland annat effektivare logistik.

1.2 Syfte

Kandidatprojektet syftar till att integrera företagets hårdvarukomponenter till ett funktionellt bläckstråleskrivarsystem, samt möjliggöra styrning av detta genom att tillhandahålla ett grafiskt användargränssnitt.

1.3 Mål

Projektets mål är att utveckla en programvara för systemets kontrollenhet. Denna programvara ska kunna styra övrig hårdvara samt ha ett användarvänligt gränssnitt som erbjuder en återkoppling av vad som ska skrivas ut.

Med hjälp av den konstruerade programvaran ska en användare kunna spara, öppna och skriva ut meddelanden. Projektet avser använda en mjukvaruarkitektur som medger utbyggnad enligt företagets önskemål.

1.4 Avgränsningar

På grund av att projektet är avgränsat, både i tid och resurser, kommer projektets genomförande endast omfatta funktionalitet för att spara, ladda in och skriva ut ett meddelande som består av en statisk text. Projektet avgränsas ytterligare, genom att enbart experimentellt stöd ges för det inbyggda systemet. Dessutom utgörs den visuella återkopplingen av utskriften endast av en statisk bild. Indata får endast bestå av ASCII-tecken och den genererade bitmappen måste rymmas i driverboardets FIFO. Implementeringen har ingen hantering av driftsstörningar.

1.5 Metod

Effektmålen nås genom att en systemarkitektur tas fram baserad på de förutsättningar som ges i kapitlet *Förutsättningar*. Därefter bryts effektmålen ned till funktionella krav som appliceras på systemarkitekturens delar. Dessa krav ligger sedan till grund för implementeringen. Sedan genomförs implementeringen parallellt för att slutligen sättas samman till ett system som uppfyller effektmålen. Lösningens funktionalitet jämförs med projektets mål genom verifiering på systemets hårdvara.

1.6 Disposition

I kapitlet *Förutsättningar* behandlas projektets uppbyggnad, genom att bland annat tekniska förutsättningar och effektmål tas upp. I kapitlet *Genomförande* beskrivs hur projektet genomförts från grundläggande arkitektur, via kravspecifikation till hur denna funktionalitet implementerats. Kapitlet *Resultat* beskriver projektets resultat och hur de verifierats. I *Diskussion* diskuteras projektets upplägg och måluppfyllelse samt resultatets säkerhet. Slutligen sammanfattas i kapitlet *Slutsats* hur väl projektets syfte uppnåtts.

2 Förutsättningar

Arbetet med att konstruera programvaran för kontrollenheten har, som de flesta projekt, ett antal förutsättningar. Dessa omfattar, förutom de rent tekniska förutsättningarna, även miljön där produkten ska användas, användaren och användningsmönstret. Projektet har varit en del i ett större projekt, vilket även det har varit en förutsättning att förhålla sig till.

Förutsättningarna har sammanställts tillsammans med Inkit AB och beskrivs nedan mer utförligt i separata avsnitt. Slutligen redovisas även de effektmål som beställaren definierat för produktens funktionalitet.

2.1 Projektförutsättningar

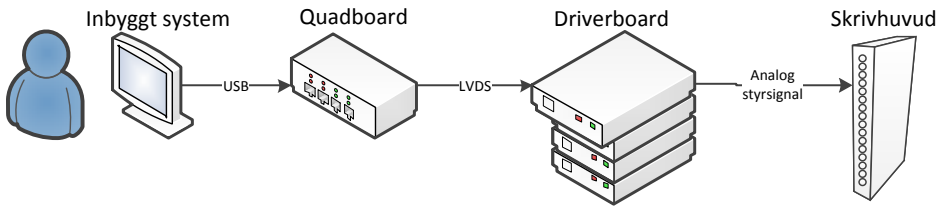
Projektet som kandidatgruppen drivit är en del i en produktutveckling av ett större system. Det innebär att de tekniska förutsättningarna varit föränderliga. Arbetet med att sammanställa projektets förutsättningar skedde iterativt, och under en längre tid eftersom varje ny förutsättning ledde till nya frågor.

2.2 Tekniska förutsättningar

De tekniska förutsättningarna omfattar förutom det inbyggda systemet som ska exekvera projektets programvara även de enheter som ska styras.

2.2.1 Hårdvara

Eftersom projektet omfattar utvecklingen av programvaran till kontrollenheten, omfattar de tekniska förutsättningarna förutom själva kontrollenheten även de enheter som kontrollenheten ska styra. All hårdvara som ingår i projektet har tillhandahållits av beställaren.



Figur 1: Översikt över systemets hårdvara

Hårdvaran består av enheterna *inbyggt system*, *quadboard* och *driverboard* vilka presenteras noggrannare nedan. Systemet där projektets framställda programvara ska användas är ett inbyggt system. Till detta är ett quadboard kopplat via USB[1]. Quadboardet är i sin tur kopplat till driverboardet i enlighet med figur 1.

Inbyggt system

Det inbyggda systemet består av en ARM[2]-baserad dator med en 7” stor resistiv pekskärm. Skärmens upplösning är 800x480 pixlar. Det inbyggda systemet är anpassat för att köra GNU/Linux, vilket gör att alla drivrutiner redan finns tillgängliga.

Quadboard

Quadboardet är i grund och botten en USB↔LVDS[3]-adapter. Enheten presenterar sig som en seriell port för det inbyggda systemet. Kommunikation sker genom att textkommandon skickas till quadboardet som översätter textkommandona till LVDS som sedan skickas till driverboardet. Quadboardets gränssnitt mot det inbyggda systemet och dess funktionsprincip är beskrivna i dokumentationen[4].

Driverboard

Driverboardet är den enhet som styr och driver själva skrivhuvudet. På driverboardet finns anslutningar för takometer, fotocell, skrivhuvud och ett LVDS-gränssnitt för anslutning till quadboard, se figur 1.

[1] Universal Serial Bus
[2] Advanced Risc Machine
[3] Low Voltage Differential Signaling
[4] Quadboard interface description, internt dokument, Inkit AB

För utskrift använder driverboardet en monokrom bitmap som lagras i en FIFO[5]-buffert. Driverboardet har också inställningsregister för bl.a. frekvensdelning av insignalen från takometern, ställtid för fotocellen och hur länge en ventil ska vara öppen för att generera en droppe.

När en utskrift triggas, vilket normalt sker genom att fotocellens stråle bryts, taktar driverboardet ut bitmappen kolumnvis till skrivhuvudet som genererar bläckdroppar, vilka träffar objektet som ska märkas.

Driverboardets gränssnitt finns specifierat i ett internt arbetsdokument[6] som även behandlar hur systemet i helhet fungerar. Driverboardet är designat på ett sådant sätt att inga hårda realtidskrav ställs på kontrollenheten.

2.2.2 Mjukvara

Beställaren hade önskemål om att programvaran skulle vara baserad på C++ och Qt. Vidare skulle systemets operativsystem vara GNU/Linux. Önskemål fanns från företaget att de mjukvarubibliotek som användes skulle ha licensvillkor motsvarande licenserna LGPL[7], (Free Software Foundation, Inc, 2007) BSD[8] (The Linux Information Project, 2005) eller Public Domain.

2.3 Miljö

Beställaren avser att lansera produkten globalt. Systemet används i industrimiljö, vilket medför att ljud och ljusförhållanden kan vara ansträngande.

2.3.1 Driftsstörningar

Systemet ska kunna hantera en driftstörning, exempelvis ett strömavbrott eller en felaktig avstängning, på ett tillförlitligt sätt. Efter att driftsstörningen är åtgärdad ska systemet återgå till normal drift.

[5]First In, First Out
[6]Driverboard interface description, internt dokument, Inkit AB
[7]Lesser GNU Public License
[8]Berkeley Software Distribution

2.4 Användare och användning

Den typiska användaren är enligt Inkit AB[9] en industriarbetare som huvudsakligen gör andra saker än att handha datorer. Den typiska användningen under produktion är att användaren väljer ett av flera fördefinierade meddelanden som innehåller specifika inställningar för just den typen av utskrift som ska ske. I normalfallet är dessa meddelanden skapade av andra personer än de som vanligtvis hanterar själva utskriften. Användandet i produktionsskedet sker uteslutande i samband med märkning av längre serier. Det innebär att operatören typiskt spenderar liten del av sin tid till handhavandet. I detta skede kan mindre korrigeringar av meddelanden ske. Hanteringen av meddelanden som ska skrivas ut sker vanligtvis då produktionsprocessen inte behöver styras av operatören, vilket medför att mer tid finns för att redigera meddelandena.

2.5 Effektmål

Effektmålen har definierats som funktionella krav, vilket är den funktionalitet som kontrollenheten ska klara av att utföra. Dessa funktionella krav har sammanställts i en lista, se bilaga A.

[9]Möte med Per Nilsson, VD Inkit AB, 2012-01-24

3 Genomförande

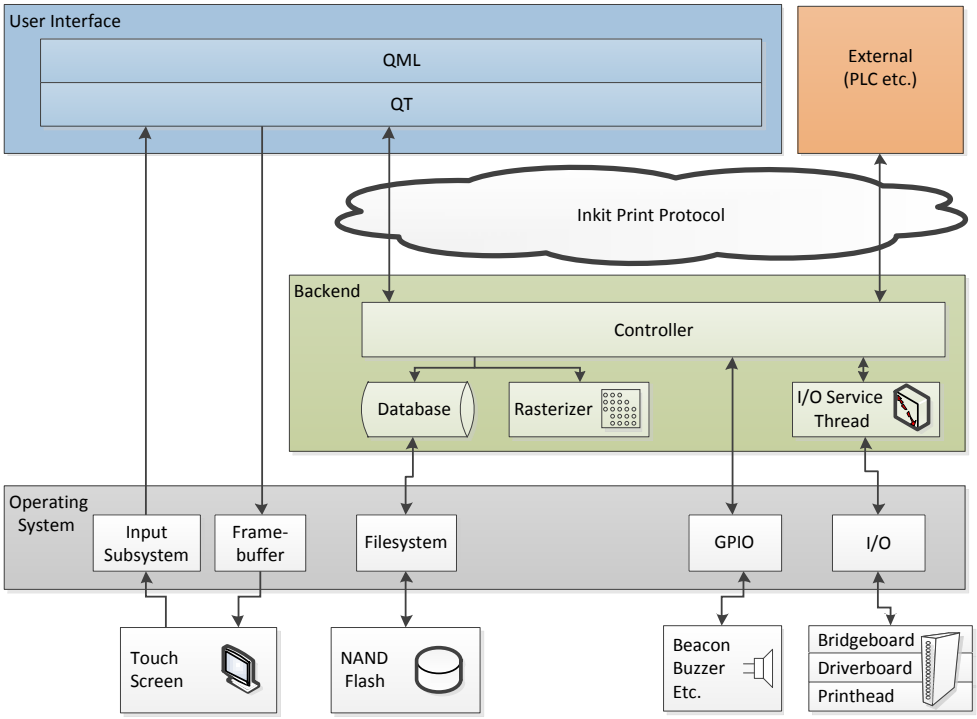
Baserat på effektmålen som definierats i bilaga A och övriga förutsättningar som redovisas i kapitel 2, gjordes en grov uppdelning i en arkitektur. Därefter sammanställdes en kravspecifikation med krav på arkitekturens ingående delar utefter tidigare nämnda krav. Slutligen implementerades kraven på arkitekturens delar.

3.1 Systemarkitektur

Då ett av önskemålen var att systemet ska vara åtkomligt även ifrån andra enheter än kontrollerenheten så behövde någon form av server-klient-kommunikation implementeras. Kandidatgruppen beslutade att programvaran skulle delas upp i två delar; *user interface* och *backend*. User inteface inriktar sig på interaktionen med användaren medan backend innehåller all logik samt interaktionen med hårdvaran. Dessutom måste delarna kommunicera mellan varandra.

De olika delarna är skrivna i olika språk. User interface är skrivet i QML[1], vilket är en del av biblioteket Qt. C++ har använts i en liten del av user interface. Som kommunikation mellan de olika delarna valdes XML[2]-RPC[3] över HTTP[4]. Detta beskrivs mera i detalj i sektion 3.1.3. Delsystemens förhållande till varandra redovisas i figur 2.

[1] Qt Meta Language
[2] eXtensible Markup Language
[3] Remote Procedure Call
[4] HyperText Transfer Protocol



Figur 2: Översikt över systemets arkitektur

3.1.1 User interface

Eftersom användargränssnittet enbart ska användas till interaktionen mellan människa och maskin valdes Qt Quick, med programspråket QML som grund. QML är ett CSS- och Javascriptliknande språk. Utvecklings-plattformen Qt Quick innehåller ett grafiskt designverktyg som möjliggör att användargränssnittets komponenter snabbt kan placeras och ges grundläggande funktionalitet. Qt Quick är uppbyggt för att möjliggöra återanvändning av kod i flera projekt. Dessutom finns stöd för internationalisering, genom att text kan ha variabelt språk samt att användargränssnittet kan väljas att ritas både från vänster till höger och från höger till vänster.

Analys av användargränssnitt

Utefter kravspecifikationen utfördes en HTA[5], se bilaga D. Denna analys definierar hur systemet ska användas för att nå det avsedda målet för användningen, och utgör kravspecifikation för användargränssnittets funktionalitet. Förutom uppgifter redovisade i HTA ska användargränssnittet även ha en funktion för att starta och stoppa skrivaren. Denna knapp ska alltid vara på samma ställe i användargränssnittet och ska alltid vara åtkomlig. Dessutom ska en knapp alltid vara tillgänglig som tar användaren till hemskärmen.

3.1.2 Backend

Backend innehåller själva logiken för kontrollenheten. Det är även backend som förser driverboardet med den bild som ska skrivas ut och konfigurerar dess inställningsregister. Eftersom backend har mycket kommunikation med hårdvaran krävdes ett språk som körs nativt på plattformen, utan att blanda in virtuella maskiner. Valet föll på C++. Backend kontrollerar också all data som exempelvis meddelanden som ska skrivas ut, räknare och övriga inställningar.

Allt som ska skrivas ut måste först renderas, då driverboardet förväntar sig en färdig bitmap som indata. Detta sker i rasteriseraren. Utöver detta har backend hand om server-delen i systemet.

Lagringsmedia

Ett effektmål är att det ska vara möjligt att spara och hämta meddelanden. Det ställer i sin tur krav på det inbyggda systemets lagringsmedia.

Enligt systemspecifikationen, se bilaga C, utgörs lagringsminnet av ett 256 MB SLC NAND flash-minne med 16 kilobyte stora *erase-blocks*. Flashminnen klarar bara ett

[5]Hierarchical Task Analysis

visst antal skrivningar per block innan de blir otillförlitliga. För minnen som bygger på SLC-teknik brukar tillverkaren garantera 100 000 skrivningar per block.

Enklare beräkningar enligt bilaga E visar att den förväntade livslängden blir åtminstone 10 år. Beställaren anser att det är tillräckligt[6].

Databas

Eftersom programmet ska hantera driftsstörningar [7] måste det alltid hålla en aktuell bild av sitt interna tillstånd sparat i icke flyktigt lagringsminne. Om den egenskapen uppfylls kan programmet återgå till normal operation när driftsstörningen har åtgärdats.

Ofta kommer mer än en av programmets interna tillståndsvariabler förändras samtidigt. Det inträffar till exempel när en utskrift triggas om antalet räknare i aktiva meddelanden är mer än ett. Alltså krävs *atomicitet* när de uppdaterade variablerna ska sparas till lagringsminnet för att inte riskera att hamna i ett inkonsistent tillstånd.

Enligt ovanstående resonemang samt slutsatserna ifrån 3.1.2 drogs slutsatsen att en databas med ACID[8]-egenskaper krävdes för att uppfylla kraven.

Några vanligt förekommande databaslösningar som uppfyller kravet om ACID utvärderades, bl.a. SQLite, MySQL och Firebird. Slutligen valdes SQLite för dess bevisade stabilitet. SQLite är vanligt i mobila och inbäddade system. Exempelvis används SQLite i både iOS-baserade (Apple Inc. (2012)) enheter ,såsom iPhone och iPad och Android-baserade enheter (Google Inc. (2012)). Dessa system är utsatta för diverse störningar, särskilt problem med strömförsörjning vilket är relevant då kandidatprojektets system utsätts för liknande driftsstörningar.

3.1.3 Protokoll

På grund utav att mjukvarusystemet är separerat i två delar måste ett protokoll specificeras och implementeras för att de olika delarna ska kunna kommunicera med varandra. Protokollet bör ha följande egenskaper:

- Textbaserat protokoll. Eftersom användargränsnittet till stora delar är implementerat i ett högnivåspråk (QML) underlättar det med ett textbaserat protokoll.
- Nätverksstöd. Beställaren har talat om att eventuellt bygga in fjärrstyrning för övervakning och support.
- Externa enheter (PLC[9] m.m) ska enkelt kunna anslutas.

[6] Telefonmöte, Per Nilsson, VD Inkit AB, 2012-03-20
[7] Möte, Per Nilsson, VD Inkit AB, 2012-02-07
[8] Atomicity, Consistency, Isolation, Durability
[9] Programmable Logic Controller

- Då det inte är klarlagt exakt vilken funktionalitet den färdiga produkten kräver måste protokollet vara lätt utbyggbart.
- Hantering av internationella tecken. Eftersom en viktig marknad är tänkt att vara Asien måste protokollet kunna hantera "internationella" tecken.

Några befintliga TCP/IP-baserade protokoll utvärderades, och det ansågs att HTTP uppfyller samtliga nämnda krav. HTTP är det protokoll som används till att hämta webbsidor och har, enligt Fielding et al. (1999), stöd för internationella tecken om teckenkodningen som används anges.

3.2 Kravspecifikation

Utifrån effektmålen som tillhörde förutsättningarna, se bilaga A, sammanställdes en kravspecifikation. Kravspecifikationen återfinns i bilaga B. Kraven delades in i funktionella krav, som baserats på effektmålen. Därefter bröts de funktionella kraven ned till funktionella följdkrav på de olika delarna i arkitekturen.

3.3 Utvecklingsmiljö

Mjukvaran är utvecklad med avseende att användas på ett Linux-baserat system. Merparten av koden är dock portabel men i vissa hörnfall har Linux-specifika API[10]:er använts.

Projektgruppen har använt versionshanteringsprogrammet *GIT* med ett centralt förråd, för hantering av källkod. Det webbaserade projektlednings- och ärendehanteringssystemet *TRAC* upprättades även i ett tidigt stadium.

Det grafiska användargränssnittet är nästan uteslutande skrivet i QML. Ungefär 90 % av koden är skriven i QML. De återstående 10 % är kod skriven i C++ som kopplar ihop det virtuella tangentbordet med gränssnitt som endast kan nås med C++. Utvecklingen av användargränssittet har skett i *IDE*:t Qt Creator.

Backend är skrivet helt i C++ där bibliotek från BOOST användes i omfattande grad för att lösa vanligt förekommande problem.

GDB, *Valgrind*, *strace* och restriktiva kompilatorflaggor är de verktyg som användes mest för felsökning och diagnostisering av programvaran.

[10] Application Programmers Interface

3.3.1 Riktlinjer

Fokus har lagts på korrekthet och läsbarhet framför prestanda. Följande principer användes för att fånga programmeringsfel så tidigt som möjligt i utvecklingsprocessen:

- Så få delade variabler mellan klasser och trådar som möjligt. I de fall där det ändå krävs, använd trådsäkra, referensräknande pekare.
- Alla indata-funktionsparametrar bör deklarerats som *const*.
- Pass-by-value i stället för pass-by-reference i så stor utsträckning som möjligt
- Funktionalitet som inte har några logiska sidoeffekter bör brytas ut i separata funktioner

3.3.2 Kodstil

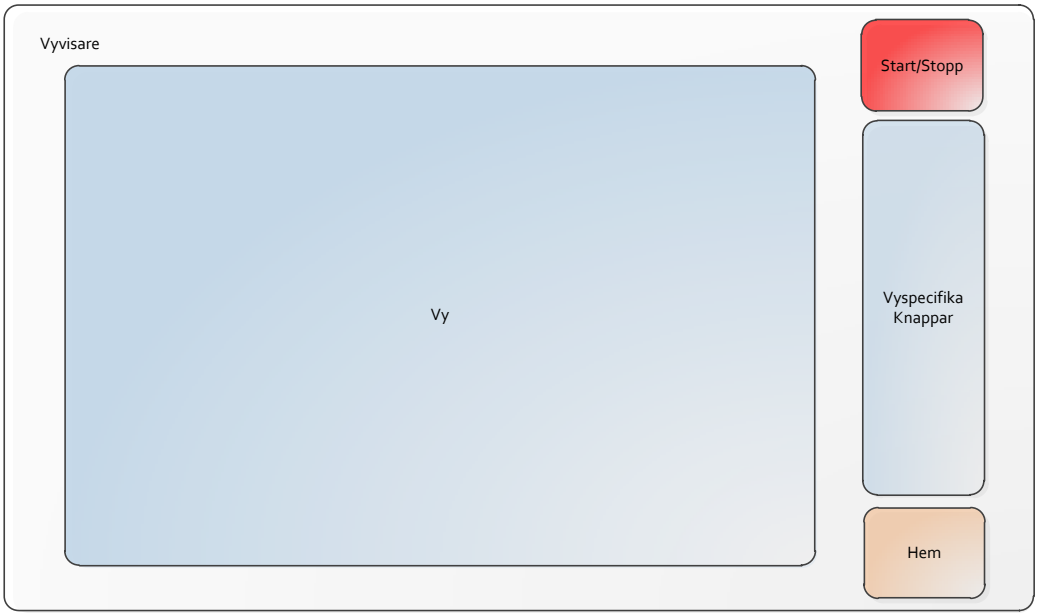
Eftersom projektet består av två separata delar, som dessutom är skrivna i olika programspråk, var det svårt att hålla en helt enhetlig kodstil. Därför har användargränssnittet och backend haft olika riktlinjer. För backend användes Googles riktlinjer för C++-kod enligt Weinberger, G. et al (2011). Användargränssnittets kodstil definierades för QML enligt Qt project hosting (2011).

3.4 Implementering

Implementeringen av kraven i kravspecifikationen har genomförts genom konstruktionen av programvara. Detta har omfattat modellering och programmering.

3.4.1 Användargränssnitt

Användargränssnittet ska enligt Bligård (2011) bland annat ha en konsekvent och samstämmig utformning. Detta ligger till grund för användargränssnittets grundkomponent. Denna ska innehålla en start/stopp-knapp som alltid är i översta, högra hörnet. Den största delen av skärmen tas upp av en komponent som innehåller den skärmbild som användaren har valt att visa, vilket är *Prio 2 Normal* i bilaga D. Skärmens högra sida ska innehålla en panel med knappar, vilka är beroende av och har ett sammanhang med den skärmbild som visas för tillfället. Slutligen ska det längst ned till höger alltid finnas en knapp som tar användaren tillbaka till startbilden, där användaren kan välja en ny skärmbild för visning. En modell över hur detta ska implementeras presenteras i figur 3.



Figur 3: Översikt över användargränssnittets grundläggande struktur

Virtuellt tangentbord

Användargränssnittet är förutom det virtuella tangentbordet implementerat helt ifrån grunden. Det virtuella tangentbordet behövs eftersom systemet saknar ett fysiskt tangentbord. Ursprungligen kommer tangentbordet från Nokias COLIBRI[11]-projekt . Tangentbordet är placerat i den nedre halvan av skärmen, och visas enbart då användaren har möjlighet att skriva in text.

Vyvisare

Vyvisaren är komponenten som visar vald vy. Komponenten har fyra tillstånd som representerar de fyra möjliga vyerna. Utgångstillståndet, som är det som är valt vid start av systemet, är hemskärmen. Hemskärmen är centrum för navigeringen, då den består av knappar som styr navigeringen till någon av de tre andra vyerna.

[11]Qt Quick COmponent LIBRary

Vy

Vyerna presenterar de huvudsakliga möjligheterna till interaktion mellan människa och maskin. Detta sker dels i huvudpanelen där den största delen av innehållet finns, och i sidopanelen på höger sida av skärmen där kontextberoende knappar visas.

Fasta komponenter

Längst upp i högra hörnet finns alltid en knapp som har två lägen; om skrivaren är i aktiverat läge är knappen röd och om skrivaren inte är i aktiverat läge är knappen grön. Ett klick på grön knapp aktiverar skrivaren och möjliggör utskrift medan ett klick på en röd knapp stoppar pågående utskrift och spärrar utskrift. Längst ned i högra hörnet finns alltid en knapp som styr visningen av skärmbild till hemskärmen, där användaren kan välja någon av de andra skärmbilderna.

Hemskärm

Navet i navigeringen mellan sidorna är hemskärmen. Den innehåller knappar som ändrar vyvisarens tillstånd, och således ändrar detta skärmens innehåll.

Redigeringsskärm

Denna vy hanterar interaktionen gällande datan som ska skrivas ut. Den består av en komponent där användaren skriver in texten som ska skrivas ut. Vidare finns en knapp som styr uppdateringen av data till backend. I sidopanelen finns två knappar som möjliggör att användaren kan spara text och hämta sparad text.

Övervakningsskärm

Övervakningsskärmen består av en vy med dynamiskt innehåll som visar den utskrift som pågår just för tillfället. Förutom detta finns en extra knapp i sidopanelen som ger användaren tillgång till parametrar som styr utseendet på utskriften.

Inställningsskärm

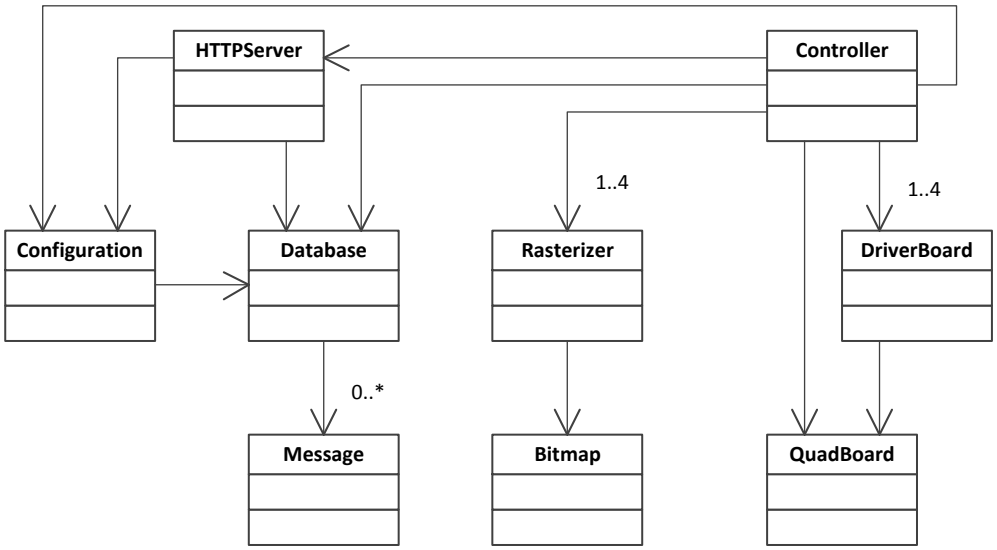
Inställningsskärmen ger möjlighet att påverka samtliga globala inställningar. Dessa är indelade i utseendespecifika inställningar och hårdvaruinställningar. För möjliga inställningar, se bilaga D.

Kommunikation

Kommunikationen med backend sker genom API:et XMLHttpRequest som definieras i W3C (2012). Läsning av data från backend sker genom *GET*-anrop, medan skrivning av data sker genom *POST* av XML-data som genereras med hjälp av Javascript.

3.4.2 Protokoll

Kommunikationen är implementerad med HTTP som protokoll. Stor vikt har lagts på att implementationen ska följa REST[12]-modellen enligt Fielding (2000). Till hjälp användes Zülke (2012).



Figur 4: Förenklat UML-diagram över Backend

3.4.3 Backend

Backend är skrivet i C++. Bibliotek från BOOST som beskrivs i Dawes (2012) har använts i så stor utsträckning som möjligt. Undantag har dock gjorts där BOOST har saknat den funktionalitet som projektet krävde. För XML-manipulation används pugi XML (Kapoulkine, 2012), för databasen SQLite (SQLite.org, 2012) och för HTTP-servern pion-net (Atomic, Inc., 2012).

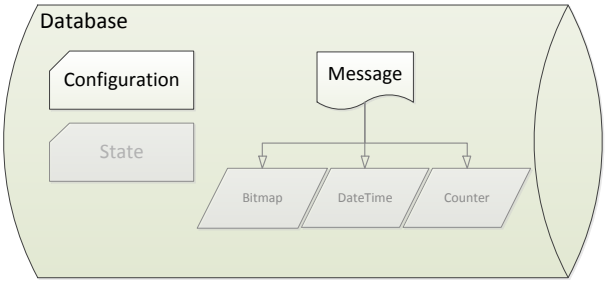
[12]REpresentational State Transfer

Controller, *QuadBoard* samt *HTTPServer* exekveras i separata trådar. För intern kommunikation mellan komponenterna i backend används en kombination av signaler/slots [13] och UNIX pipes.

Figur 4 beskriver i stora drag hur nedanstående komponenter förhåller sig till varandra. Se bilaga F för ett fullständigt UML-diagram.

Controller

Controller är systemets knutpunkt. Klassen sköter initiering och uppstart av övriga komponenter i backend. För att kunna kommunicera med användaren och övrig hårdvara ansluts vid uppstart *slots* till signaler på HTTPServer och Quadboard. När ett slot blir anropat uppdaterar Controller programmets interna tillstånd och utför nödvändiga operationer.



Figur 5: Översikt över databasernas struktur (gråa fält är inte med i implementationen)

Database

Databasen används för att spara och läsa inställningar, meddelanden och objekt i enlighet med figur 5.

Databasklassen är implementerad som ett skal runt SQLite. Dess syfte är att dela databasen mellan programmets trådar. Det uppnås genom att först initiera SQLite och därefter tillhandahålla en pekare till ett SQLite-handtag (en. handle).

Configuration

Configuration innehåller programmets globala inställningar. Klassen använder internt en XML-struktur. Övriga klasser kan söka bland inställningarna med hjälp av XPath.

[13] Boost.Signals2

HTTPServer

Backend har en serverkomponent som kommunicerar med användargränsnittet. För att förenkla utvecklingsprocessen har biblioteket pion-net som beskrivs av Atomic, Inc. (2012) använts.

Quadboard

Denna klass är en implementation av det fysiska quadboard på mjukvarunivå. Eftersom serieports-gränssnittet som tillhör det fysiska quadboardet misstänkes vara långsamt och utgöra en flaskhals exekveras klassen i en separat tråd och använder buffrad kommunikation. Trådsäkerhet har uppnåtts genom ett statuslås som förhindrar läsning av status då denna är under uppdatering samt ett accesslås som hanterar andra trådars åtkomst. Implementationen av quadboard har stöd för både synkron åtkomst och burst mode. I burst mode går det bara att skriva och det går inte att ta reda på om ett skickat kommando mottagits korrekt. I gengäld är burst mode mycket snabbare än synkron åtkomst. Klassen vidarebefodrar även avbrott som fångas upp från det fysiska quadboardet.

Driverboard

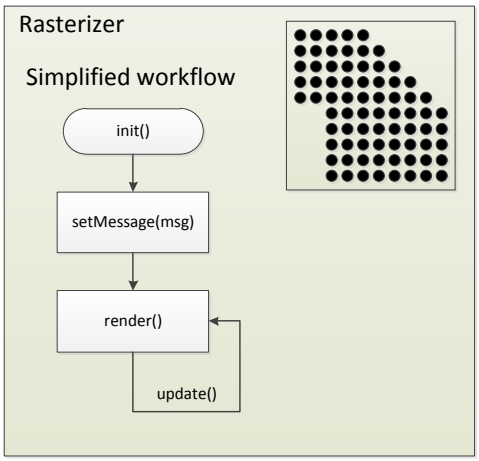
Klassen är en implementation av det fysiska driverboardet på mjukvarunivå. Kommunikationen med det fysiska driverboard sker genom quadboard-klassen. Implementationen använder burst mode för att överföra bitmappar, men synkron kommunikation för enstaka inställningar.

Bitmap

Klassen är programmets interna representation av en bitmap. Den stödjer enkla operationer såsom rotation, klippning, storleksändring, överlagring samt serialisering. Internt använder klassen BOOST::dynamic_bitset för att förenkla åtkomst till enskilda bitar.

Message

Ett *Message* innehåller referenser till den text och bilder som skapats i användargränsnittet. Message lagras som ett XML-dokument i databasen.



Figur 6: Rasteriserare

Rasterizer

Rasteriseraren får som indata ett *Message* och renderar en *Bitmap* utifrån de element som ingår. Klassen använder internt FreeType för att hantera typsnitt samt rendera text. Arbetsgången för rasteriseraren beskrivs i figur 6.

4 Resultat

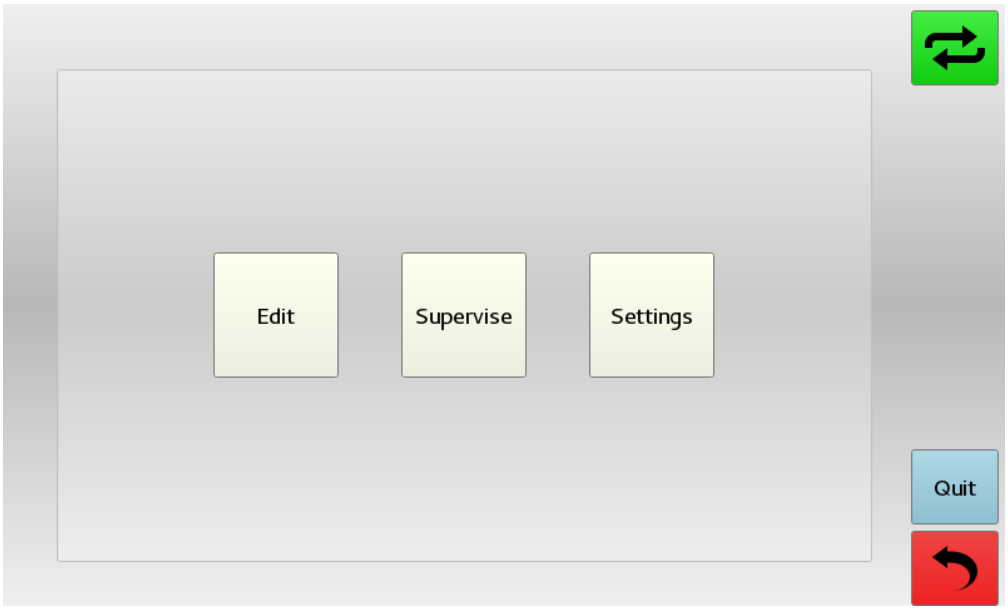
Kandidatgruppen har under projektets gång utvecklat mjukvara samt nödvändiga protokoll för att klara av målsättningen med projektet och dessutom verifierat dess funktionalitet. Nedan beskrivs hur dessa delar tillsammans uppfyller rapportens mål. Detta verifieras därefter i avsnittet *Verifiering*

4.1 Mjukvara

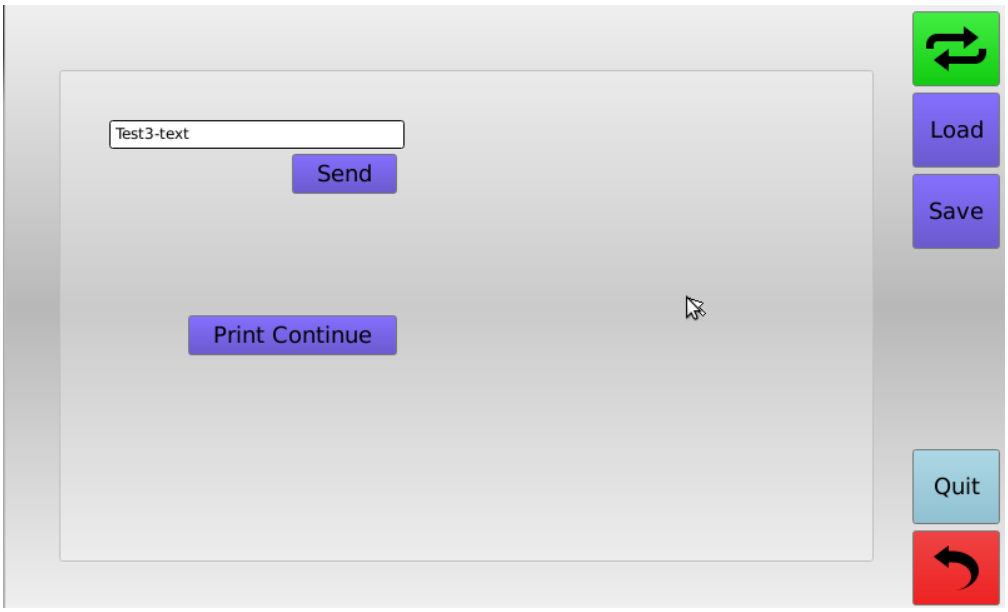
Som det har beskrivits tidigare är systemet uppdelat i två distinkta program, ett för användargränssnittet och ett för backend. Nedan redovisas den funktionalitet som används i respektive del för att nå projektets mål.

4.1.1 Användargränssnitt

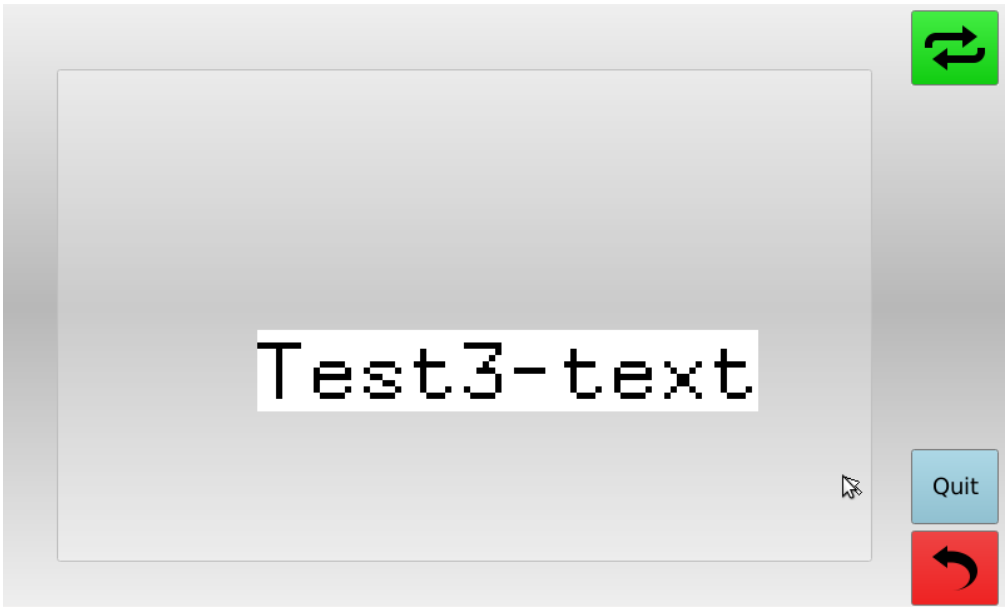
Användargränsnittet har ett genomgående utseende och har möjlighet till att skapa texter som kan skickas till backend. Figur 7 visar hur startskärmen ser ut. Användaren har här möjlighet att klicka på det val som denne är intresserad av och får då fram en en ny vy med de verktyg som krävs. Figur 8 visar redigeringsskärmen, där användaren ges möjlighet att redigera meddelandens text. Denna vy ger även möjlighet att spara meddelanden i den databas som finns i backend samt hämta tillbaka sparade meddelanden. Övriga knappar används för att styra utskriften. Knappen längst upp i högra hörnet sätter skrivaren i ett aktivt läge, där meddelandet skrivs ut när en triggsignal ges. Knappen *print continue* används för att generera en triggning. Övervakningsskärmen, som kan ses i figur 9 , visar en bild som beskriver den renderade bitmapp som ska skrivas ut. I figur 10 visas inställningsskärmen där möjlighet ges till inställning av parametrar som påverkar utskriften.



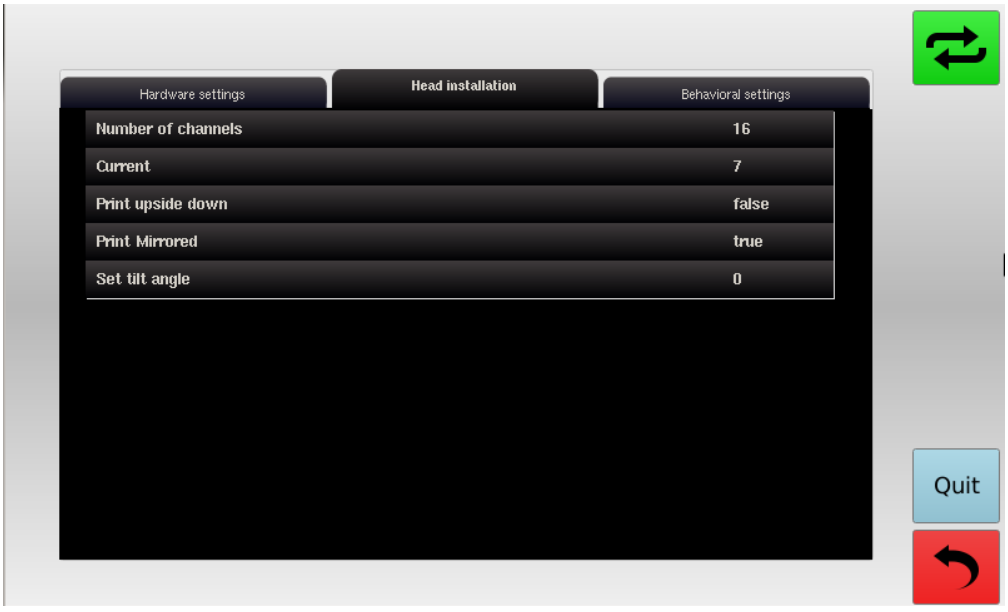
Figur 7: Startskärm



Figur 8: Redigeringsskärm



Figur 9: Övervakningsskärmen



Figur 10: Inställningsskärmen

4.1.2 Backend

Backend hanterar via *HTTPServer* förfrågningar från användargränssnittet och vidarebefodrar förfrågningarna till *Controller*, som utför de åtgärder som efterfrågats. Backend rasteriserar också den text som ska skrivas ut, skickar den genererade bitmappen till driverboardets FIFO, samt konfigurerar driverboardets styrparametrar för utskrift. Eftersom Backend använder SQLite för att lagra all information så sker det på ett säkert sätt, vilket efterfrågats i kravspecifikationen.

4.2 Verifiering

Kandidatgruppen har verifierat den utvecklade programvaran under arbetets gång. I inledningsfasen verifierades delsystemen var för sig, men i takt med att de började integreras med varandra skedde mer omfattande tester där till slut hela systemet verifierades samtidigt.

4.2.1 Backends delkomponenter

En viktig del av utvecklingsprocessen har varit att parallellt med implementeringen av backends klasser, samtidigt skriva upprepbara testfall för dessa klasser. På så sätt har programmeringsfel och regressioner kunnat fångas upp i ett så tidigt i stadie som möjligt.

4.2.2 Kommunikation mellan användargränssnitt och backend

Verifieringen har skett löpande under arbetets gång och har testats genom att göra förfrågningar mot backend, kontrollera svaret från backend samt undersöka vad backend har vidtagit för åtgärder vid varje förfrågning. Den funktionalitet som redovisats gällande användargränssnittet har också verifierats med denna metod. Undantaget är inställningsskärmen, där all funktionalitet inte är implementerad i backend.

4.2.3 Kommunikation mellan backend och hårdvara

Det första som implementerades i backend var synkron kommunikation mellan backend och quadboard. Därefter utfördes omfattande stresstester där det genom skrivning och återläsning av register i quadboard har kunnats verifieras att kommunikationen fungerar.

Eftersom kandidatgruppen inte har haft tillgång till ett skrivhuvud under utvecklingen har ingen verifiering av utskriften kunnat göras kontinuerligt. Dock har det kunnat

observeras att den renderade bitmappen laddats in i driverboardets FIFO. Sedan har det kunnat konstaterats att FIFOt är fyllt med det antal kolumner som bitmappen betingar. Vid manuell uttaktning från FIFOt har kunnat observeras att FIFOts fyllnadsgrad har minskat med förväntad takt. Kandidatgruppen har på så sätt kunnat verifiera att en korrekt sekvens av styrkommandon utförts för att genomföra en utskrift.

Verifiering har också gjorts av de styrparametrar som har möjlighet att ställas in. Data har då skickats till driverboardet, och sedan återlästs och det har då konstaterats att rätt svar har skickats tillbaka.

4.2.4 Verifiering av en hel utskrift

Försök gjordes att verifiera en utskrift med all hårdvara inkopplad. Dock orsakade det inkopplade skrivhuvudet att driverboardet förstördes. Resultatet blev att en hel utskrift inte kunde verifieras. Inkit AB konstaterade att orsaken till att driverboardet förstördes berodde på ett logikfel i hårdvaran.

5 Diskussion

I detta kapitel diskuteras resultatets säkerhet, viktiga val under projektets gång, samt förutsättningar som påverkat projektets genomförande.

5.1 Resultatets säkerhet

Uppfyllelsen av projektets mål kan diskuteras, då en verifiering av en hel utskrift med all ingående hårdvara inte kunde göras. Kandidatgruppen anser dock att eftersom det kunde observeras att FIFOt kunde fyllas, samt att det kunde observeras att FIFOt tömdes i överensstämmelse med manuell uttaktning via användargränssnittet, kan det göras sannolikt att en utskrift genom projektets mjukvara är möjlig om all hårdvara uppfyller specifikationerna.

Verifiering av driverboardets utsignaler kunde ha genomförts genom mätning av dessa med hjälp av oscilloskop eller logikanalysator. Alternativet utvärderades av kandidatgruppen, men bedömdes för svårt att genomföra på grund av bristfällig dokumentation av driverboardets utgångar. Hade denna verifiering kunnat genomföras skulle mätdata kunnat jämföras med den renderade bitmappen och ett säkrare resultat hade därmed kunnat uppnås.

5.2 Tidsbegränsning

På grund av att projektets hela omfattning ej var känd från början fick avgränsningarna ändras under projektets gång. I ett tidigt skede bedömdes beställarens mål som rimliga att genomföra inom projektets tidsramar, men när alla förutsättningar blev kända framkom det snabbt att projektet var för omfattande. Därför gjordes ytterligare avgränsningar.

I balansgången mellan beställarens mål och kandidatgruppens resurser prioriterades både viss funktionalitet och visst analysarbete bort. En djupare analys gällande programvarans användarvänlighet planerades i ett tidigt skede, men prioriterades bort till förmån för arbete med implementering. Valet av att ge användargränssnittet en konsekvent och

samstämmig utformning anses tillräckligt för att användargränssnittet ska vara användarvänligt med avseende på den funktionalitet som implementerats.

Tidsbegränsningen gjorde även att stödet för det inbyggda systemet prioriterades ned. Eftersom viss programvara som är förinstallerad på det inbyggda systemet är av en äldre version än vad som utlovades i början av projektet skulle detta stöd ha krävt tid och resurser som inte finns. Valen av mjukvarubibliotek är dock gjorda med tanke på det inbyggda systemet och bör därför fungera då rätt programvara finns installerad.

5.3 Programmeringsspråk och plattform

Redan innan kandidatprojektets påbörjande har kontrollenhetens hårdvaruplattform varit fastställd. Således har kandidatgruppen inte haft någon möjlighet att påverka kontrollenhetens hårdvara och har helt enkelt varit tvungen att förhålla sig till den.

5.3.1 Valet av C++ för utveckling av backend

Som nämnts tidigare i kapitlet förutsättningar hade beställaren önskemål om att mjukvaran skulle skrivas i Qt och C++. Trots detta utvärderade kandidatgruppen andra alternativ men fann att C++ långsiktigt var det bästa valet för backend.

Det går att argumentera för att ett högnivåspråk som t.ex Python hade varit fullgott för kandidatprojektets omfattning. Om kandidatgruppen istället för C++ hade valt att använda ett högnivåspråk för implementeringen av backend hade sannolikt ett mer omfattande resultat uppnåtts.

Kandidatgruppen valde dock att ta hänsyn till beställarens långsiktiga mål: eftersom det inbyggda systemets hårdvara är relativt begränsad innebär ett högnivåspråk en mer överhängande risk att senare stöta på prestandamässiga flaskhalsar som begränsar hela systemets funktionalitet.

5.3.2 Valet av Qt Quick för utveckling av användargränssnitt

Vad gäller användargränssnittet har kandidatgruppen resonerat precis tvärtom. Eftersom användargränssnittet inte kan anses utgöra någon prestandamässigt kritisk del av systemet kan med fördel ett högnivåspråk användas. Valet av Qt Quick låg nära till hands, dels eftersom Qt har ett bra stöd för inbyggda system enligt Qt project hosting (2012) och dels eftersom Qt var ett önskemål från beställaren redan från början.

Tesen om snabbhet från idé till funktionalitet bevisades, då kandidatgruppen i ett tidigt skede hade en prototyp med viss funktionalitet. Även i det fortsatta utvecklingsarbetet upplevdes QML och Qt Quick som en utvecklingsmiljö som var enkel och tillräckligt

kraftfull för syftet. Integreringen av det virtuella tangentbordet, som är en extern komponent gick smidigt. Vissa mindre anpassningar gjordes, men på det stora hela visade det ändå att Qt Quicks filosofi om återanvändningsbara komponenter fungerar.

Vad gäller kommunikation så uppstod problem eftersom QML bara har ett reducerat stöd för XML DOM. QML's DOM-implementation har tillräcklig funktionalitet för att läsa XML-data men saknar tillräcklig funktionalitet för manipulation av densamma. Lösningen projektgruppen har valt är att XML-data som ska skrivas sätts samman i ett javascript vilket är tidskrävande både för utvecklaren och systemet. Projektgruppen bedömer dock att påverkan från detta på resultatet är försumbart.

5.4 Representation av användargenererad data

Kandidatgruppen planerade först att utforma ett domänspecifikt språk, vilket beskrivs i Mernik et al. (2005), för representation av användargenererad data. När så omfattningen av beställarens slutgiltiga målsättning blev mer känd framgick det att problemet var mer komplext än vad som först antagits. Därför fick idén om ett anpassat format ge vika för ett mer generellt, utbyggbart format . Valet stod då mellan XML och JSON. Slutligen valdes XML eftersom Qt Quick enligt föregående resonemang utlovade stöd för det.

5.5 Framtidssäkring av backend

Beställarens önskemål om att koden ska vara utbyggbar och att effektmål i senare nivåer ska hållas i åtanke har medfört merarbete främst då det gäller utvecklingen av backend. Där återfinns stöd för funktioner som inte innefattas av effektmålen, *Prio 1 Basic* så som de är definierade i bilaga A. Dessa funktioner har inneburit att tid har fått tas från annat arbete inom projektet, men kandidatgruppens bedömning är att det har bidragit till förståelse för hur systemet ska fungera när samtliga delar av produktutvecklingen är klar. Detta har lett till att kandidatprojektet har kunnat implementera funktionalitet som motsvarar effektmålen, men som samtidigt är utbyggbar.

6 Slutsats

I detta kapitel redovisas slutsatser med utgångspunkt från detta projekt. Vidare presenteras även förslag på framtida arbete.

6.1 Slutsatser

Utifrån projektets genomförande och resultat kan följande slutsatser dras:

- Projektets syfte att kunna styra bläckstråleskrivaren genom ett grafiskt användargränssnitt uppnås, med viss osäkerhet på grund av hårdvarans tillgänglighet.
- Projektet tog längre tid att genomföra än vad som avsågs från början. Det beror på att projektets förutsättningar var tidskrävande att utvärdera.
- Programvaran är användarvänlig, i avseende att den har en konsekvent och samstämmig utformning.
- Användandet av XML över HTTP för kommunikation mellan programmets delar möjliggör utbyggnad av systemet för att omfatta ytterligare funktionalitet.
- Qt Quick som utvecklingsmiljö gav förutom effektiv utveckling av användargränssnittet även stöd för internationalisering av text och orientering.

6.2 Framtida arbete

För att föra detta projekt vidare föreslår kandidatgruppen att arbete med implementation av ytterligare funktionalitet enligt bilaga A, nivå två och tre. Vidare anser kandidatgruppen att en större studie över användargränssnittets användarvänlighet genomförs, som omfattar dess slutgiltiga funktionalitet.

Litteraturförteckning

Apple Inc. (2012). *Data Management - Apple Developer.* Webbsida: http://developer.apple.com/technologies/ios/data-management.html Läst 2012-05-09.

Atomic, Inc. (2012). *Pion Network Library - Overview.* Webbsida: http://www.pion.org/projects/pion-network-library Läst 2012-05-07.

L.-O. Bligård (2011). *Utvecklingsprocessen ur ett människa-maskinperspektiv.* Göteborg: Chalmers Tekniska högskola.

B. e. a. Dawes (2012). *BOOST C++ libraries.* Webbsida: http://www.boost.org Läst 2012-05-07.

R. Fielding, et al. (1999). *Request For Comments 2616 - Hyper Text Transfer Protocol – HTTP/1.1.* Webbsida: http://www.w3.org/Protocols/rfc2616/rfc2616-sec3.html#sec3.4 Läst 2012-05-02.

R. T. Fielding (2000). *Architectural Styles and the Design of Network-based Software Architectures.* Ph.D. thesis, Irvine: University of California.

Free Software Foundation, Inc (2007). *GNU LESSER GENERAL PUBLIC LICENSE.* Webbsida: http://www.gnu.org/copyleft/lesser.html Läst 2012-05-07.

Google Inc. (2012). *Data Storage - Developer Pages.* Webbsida: http://developer.android.com/guide/topics/data/data-storage.html Läst 2012-05-09.

A. Kapoulkine (2012). *pugixml.* Webbsida: http://pugixml.org Läst 2012-05-07.

M. Mernik, et al. (2005). 'When and how to develop domain-specific languages'. *ACM Comput. Surv.* **37**(4):316–344.

Qt project hosting (2011). *QML coding conventions.* Webbsida: http://qt-project.org/doc/qt-4.8/qml-coding-conventions.html Läst 2012-05-07.

Qt project hosting (2012). *Support for embedded linux.* Webbsida: http://qt-project.org/wiki/Support_for_Embedded_Linux Läst 2012-05-25.

SQLite.org (2012). *C-language interface för SQLite*. URL: http://www.sqlite.org/ Läst: 2012-05-09.

The Linux Information Project (2005). *BSD License Definition*. Webbsida: http://www.linfo.org/bsdlicense.html Läst 2012-05-07.

W3C (2012). *XMLHttpRequest Level 2*. Webbsida: http://www.w3.org/TR/2012/WD-XMLHttpRequest-20120117/ Läst 2012-05-07.

Weinberger, G. et al (2011). *Google C++ style guide*. Webbsida: http://google-styleguide.googlecode.com/svn/trunk/cppguide.xml Läst 2012-05-08.

D. Zülke (2012). *Designing HTTP Interfaces and RESTful Web Services*. Webbsida: http://www.slideshare.net/Wombert/designing-http-interfaces-and-restful-web-services-phpday11-20110514/ Läst: 2012-04-18.

A Effektmål

Version 1							
Prio 1 Basic	*Prio 2 Normal*	*Prio 3 Advanced*		*Inställningar/Print Settings*	*Lokal*	*Redigerbar Default*	*Globalt*
X			Appearence	Set font type	X	X	
X			Appearence	Set font size	X		
		X	Appearence	Special character	X		
		X	Appearence	Set Narrow font	X		
	X		Appearence	Set Wide	X		
		X	Appearence	Set Wide light	X		
		X	Appearence	Set inverted text	X		
X			Appearence	Set character spacing	X		
X			Appearence	Column width	X	X	
X			Appearence	Set Drop size	X	X	
X			Appearence	Set print upside down	X		
	X		Appearence	Set opposide side printing	X?		X
X			Appearence	Set Print margin	X	X	
		X	Appearence	Object lenght	X		
	X		Object	New Counter	X		X
	X		Object	Edit Counter	X		X
	X		Object	New shift	X		X
	X		Object	Edit shift	X		X
		X	Object	New user prompt	X		X
		X	Object	Edit user prompt	X		X
	X		Object	New Exp date	X		X
	X		Object	Edit Exp date	X		X
	X		Object	New Text object	X		X
	X		Object	Edit Text object	X		X
	X			New ext. variable	X?		X
	X			Edit ext. variable	X?		X
X			Setup	Edit time			X
X			Setup	Edit date			X
	X		Setup	Set date change time			X
X			Setup	Set Trig distance			X
X			Setup	Setup printhead			X
X			Appearence	Set tilt (skrivhöjd)			X
X			Setup	Set encoder pulse rate			X
X			Setup	Set print speed			X
X			Setup	Set print direction			X
	X		Setup	Set continuous print			X
X			Setup	Set trig mode			X
		X	Setup	Terminate			X
		X	Setup	Startup boost			X
X			Create message	Insert Text			
X			Create message	Insert Date			
X			Create message	Insert Time			
	X		Create message	Insert User defined field			
	X		Create message	Insert ext variable			
X			Create message	Insert Bitmap			
		X	Create message	Insert Barcode			

B Kravspecifikation

Se följande fyra sidor för kravspecifikationen.

Kravspecifikation.

* indikerar krav som ej accepterats

Systemet skall:

Effektmål 0 kunna välja **setupinställningar**
Användargränssnitt:
-skall ha ett menyval som heter inställningar
-skall kunna hämta och lagra inställningar i backend
IIP (XML schema)
-skall erbjuda <configuration>
Databas:
-skall erbjuda lagring av inställningar
Controller:
-skall kunna vidarebefodra inställningar mellan Användargränssnitt/Databas
E0.a kunna välja **trigdelay**
Användargränssnitt
-skall ha ett menyval under inställningar som heter trigavstånd
IIP(XML schema)
-skall erbjuda <trigdist>
Datapump:
-skall ingå i header

*E0.b kunna välja skrivhuvudens egenskaper
Användargränssnitt
-skall ha ett menyval under inställningar som heter skrivhuvuden
-skall visuellt beskriva huvudens ordning till varandra
*E0.b.1 kunna välja skrivhuvudets förhållande till referenspunkt
Användargränssnitt
-skall ha ett menyval under skrivhuvuden som heter orientering/referens eller liknande
-skall visuellt visa orienteringen i wysiwygåterkopplingen
IIP (XML schema)
-skall erbjuda <head> som beskriver vilket huvud som avses
*E0.b.2 kunna välja antal kanaler
Användargränssnitt
-skall ha ett menyval under skrivhuvuden som heter kanaler där antal kanaler för varje skrivhuvud kan väljas
-skall visuellt återspegla antal kanaler per huvud i wysiwygåterkopplingen
IIP(XML schema)
-skall erbjuda <channels> som egenskap för varje <head>
Rasteriserare
-skall använda <channels> som indata för bitmapens höjd
*E0.b.3 kunna välja om huvudet är stackat
Användargränssnitt
-skall ha ett menyval under skrivhuvuden som heter stackade
-skall visuellt hantera två stackade huvuden som ett enda
IIP (XML schema)
skall erbjuda <activestackedwith> och <passivestackedwith>
Rasteriserare
-skall använda <activestackedwith> för att beräkna höjden på bitmap

-skall använda <activestackedwith> och <passivestackedwith> för att formatera utdata

E0.b.4 kunna välja om huvudet är tiltat

Användargränssnitt

-skall ha ett menyval under skrivhuvuden som heter tiltat

-skall visuellt återspegla tiltat huvud genom att göra bilden kompaktare

IIP (XML schema)

-skall erbjuda <tilt> som egenskap för <head>

Datapump

-skall ingå i header

E0.b.5 kunna välja huvudets trigavstånd

Användargränssnitt

-skall ha ett menyval under skrivhuvuden som heter trigavstånd

-skall bortse från leading zeroes

Rasteriserare

-skall lägga in leading zeroes motsvarande trigavstånd-(trigdelay*skrivhastighet)

E0.b.6 kunna välja utskriftsriktning

Användargränssnitt

-skall ha en inställning under skrivhuvuden som heter utskriftsriktning

-skall återspegla utskriftsriktning i wysiwyg

IIP (XML schema)

-skall ha <oppositeside>

Rasteriserare

-skall vända på bitmap

E0.c kunna välja utskrifthastighet

Användargränssnitt

-skall ha ett menyval under inställningar som heter utskriftshastighet

-skall räkna om utskriftshastighet till tachodelning

IIP (XML schema)

-skall ha <tachorate>

Datapump

-skall ligga i header

E0.d kunna välja trigorsak

Användargränssnitt

-skall ha ett menyval under inställningar som heter trigorsak: manuell eller fotocell

-skall ha möjlighet att generera manuell trig

IIP(XML Schema)

-skall ha <trigmode> i <settings> och <sendtrig>

Datapump

-skall kunna skicka vidare manuell trig

E0.1 kunna välja **defaultinställningar** för meddelanden

Användargränssnitt

-skall ha ett menyval som heter defaultinställningar

IIP (XML schema)

-skall ha <defaultmessage>

E0.1.a kunna välja en font som är default

Användargränssnitt

-skall ha ett menyval under defaultinställningar som heter font
IIP (XML schema)
-skall ha <font>

E0.1.b kunna välja en fast kolumnbredd
Användargränssnitt
-skall ha ett menyval under defaultinställningar som heter kolumnbredd
IIP (XML Schema)
-skall ha <columnwidth>

E0.1.c kunna välja en droppstorlek
Användargränssnitt
-skall ha ett menyval under defaultinställningar som heter droppstorlek
IIP (XML schema)
-skall ha <dropsize>
Datapump
-skall ligga med i header

E0.1.d kunna välja en utskriftsmarginal
Användargränssnitt
-skall ha ett menyval under defaultinställningar som heter utskriftsmarginal
-skall bortse från leading zeroes i wysiwygåterkoppling
IIP (XML schema)
-skall ha <printmargin>
Rasteriserare
-skall addera <printmargin> till leading zeroes som kommer från trigavstånd

E1 kunna skapa ett **meddelande**
Användargränssnitt
-skall ha ett menyval från startbilden som heter skapa nytt meddelande
IIP (XML schema)
-skall ha <message>

E1.a kunna välja en font som är meddelandespecifik
-skall ha ett menyval från nytt meddelande som heter font
IIP (XML schema)
-skall ha <font>

E1.b kunna välja en fast kolumnbredd som är meddelandespecifik
-skall ha ett menyval från nytt meddelande som heter kolumnbredd
IIP (XML schema)
-skall ha <columnwidth>

E1.c kunna välja en droppstorlek som är meddelandespecifik
-skall ha ett menyval från nytt meddelande som heter droppstorlek
IIP (XML schema)
-skall ha <dropsize>

*E1.d kunna välja en utskriftsmarginal som är meddelandespecifik
-skall ha ett menyval från nytt meddelande som heter utsriftsmarginal
IIP (XML schema)
-skall ha <printmargin>

E1.e kunna välja en utskrift som är upp och ner
-skall ha ett menyval från nytt meddelande som heter orientering
IIP (XML schema)
-skall ha <orientation>

E1.1 kunna lägga in en **text** i meddelandet

Användargränssnitt
-skall kunna välja text under meddelande
IIP (XML schema)
-skall ha <text>

C Systemspecifikation

C.1 CPU-modul

- MCU: AT91SAM8G45, 400 MHz ARM926EJ-S^{TM}ARM®Thumb®processor
- 128 MiB DDR2 SDRAM, 16-bit
- 256 MiB NAND Flash
- 10/100 MBit/s Ethernet
- SO-DIMM 200 JEDEC MO-274 module (67x35.5mm)
- 32 KiB Data Cache
- 32 KiB Instruction Cache
- Memory Management Unit
- Dual External Bus Interface
- DMA Controller
- AC97 Interface
- Embedded ICE
- 128 KiB Internal ROM
- 64 KiB Internal SRAM

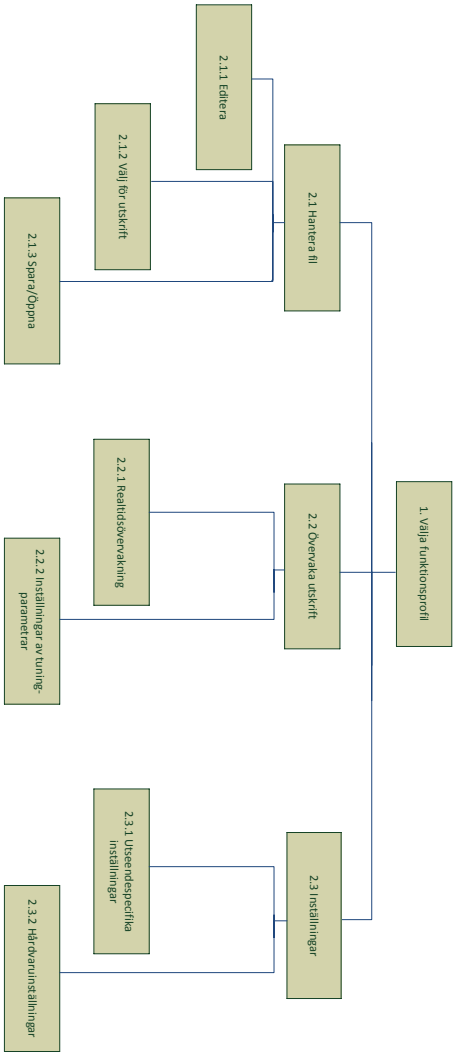
C.2 Utvecklingskort

Utvecklingskortet är ett kort där man har dragit ut en mängd anslutningar för lätt åtkomst. På kortet finns bland annat:

- MMC/SD card slot
- Compact Flash socket

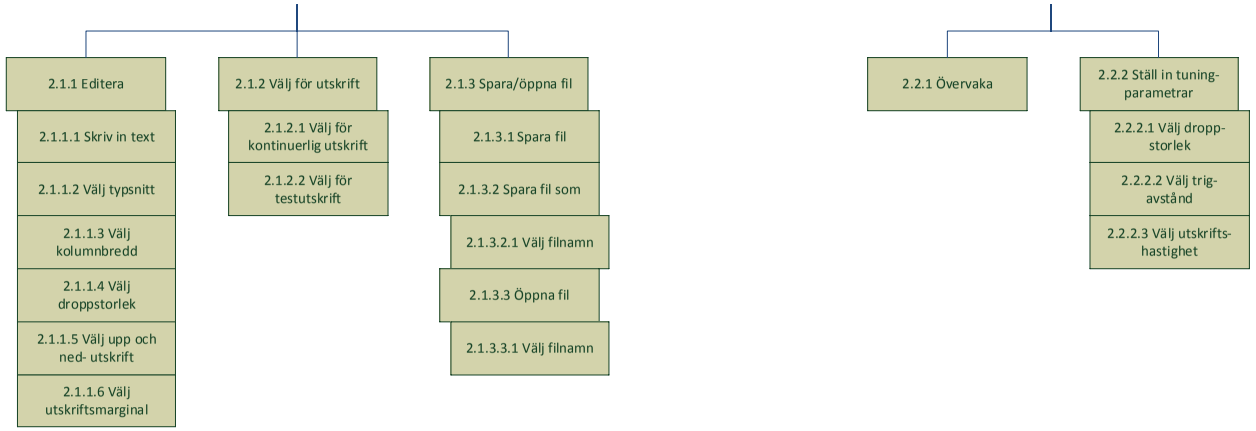
- LCD and Touchscreen interface
- 7.0” Touch screen LCD
- USB Host and slave
- UART RS232
- Debug RS232

D HTA

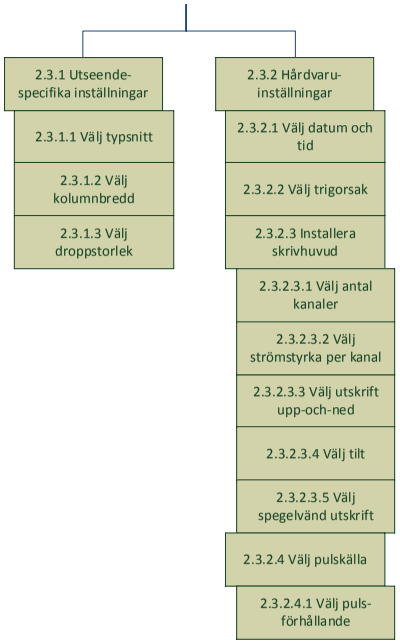


Hierarkisk uppgiftsanalys nivå 0-2

Hierarkisk uppgiftsanalys nivå 3



Hierarkisk uppgiftsanalys nivå 3



E Livstidsanalys av Flash-minne

Manufacturers 'Guaranteed' safe WRITES per erase-block

	SLC	**MLC**
	100000	5000

Size (MB) 256

Assume UBI/UBIFS[5] [3 p.27] uniform full disk wear leveling.

Block size (kb)	16	128	256	512
Blocks	16384	2048	1024	512
Total expected writes				
SLC	1638400000	204800000	102400000	51200000
MLC	81920000	10240000	5120000	2560000

Assume all writes fit block size (=< 16 KB)
Assume all writes are in different blocks.

(Assume 4 print heads * 4 counters + 1 statistics + 1 state)* 24 prints
+ 20 external (log files etc) / min == 452 IO/min ~= 8 IOPS)
Is this worst case?

Expected 'reliable' lifetime (days (24h))

Block size(kb)	16	128	256	512
SLC	2370,37037	296,296296	148,148148	74,0740741
MLC	118,518519	14,8148148	7,40740741	3,7037037

Put battery backup on SRAM or system and flush to flash every 1 minute
(Assume 4 print heads * 4 counters + 1 stat + 1 state)* 1 flush
+ 20 external (log files etc) / min == 38 IO/min ~= 0.66 IOPS

Block size(kb)	16	128	256	512
SLC	28731,7621	3591,47026	1795,73513	897,867565
MLC	1436,5881	179,573513	89,7867565	44,8933782

Summary:
Assume 8h workday and 230 workdays/year
Theoretically we could claim 30 years data retention with 256MB 16kb erase-blocks
SLC NAND flash. Flash manufacturers seem to claim 10Y (?)

[1] http://www.snia.org/sites/default/files/SSSI_NAND_Reliability_White_Paper_0.pdf
[2] http://sourceware.org/jffs2/jffs2.pdf
[3] http://elinux.org/images/9/9a/CELFJamboree29-FlashFS-Toshiba.pdf
[4] http://www.wdc.com/WDProducts/SSD/whitepapers/en/NAND_Evolution_0812.pdf
[5] http://www.linux-mtd.infradead.org/doc/ubifs.html
[1] Another limitation is that flash memory has a finite number of program-erase cycles (typically written as P/E cycles).Most commercially available flash products are guaranteed to withstand around 100,000 P/E cycles, before the wear begins to deteriorate the integrity of the storage.

F UML-diagram över backend