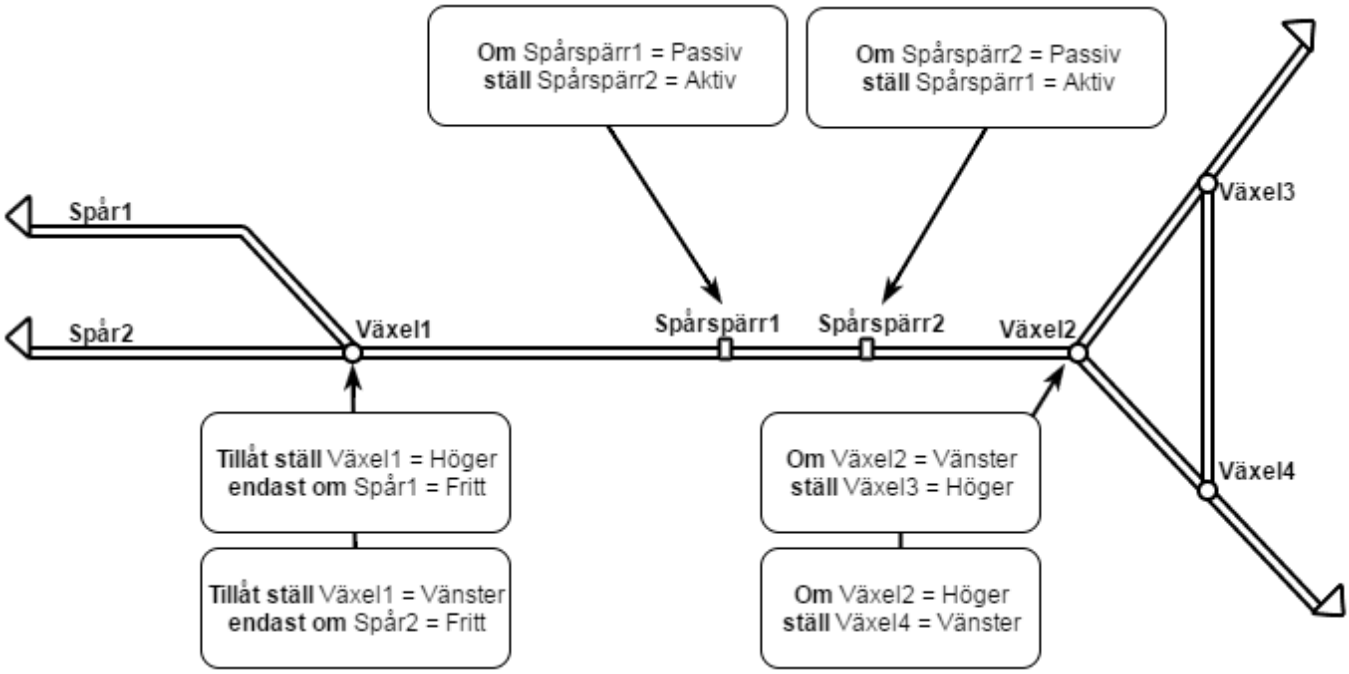
CHALMERS



Styrprogram med konfigurerbar logik för spårtrafik

Examensarbete inom Data- och Informationsteknik

LINN LEIULFSRUD
JOHAN SJÖBERG

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

EXAMENSARBETE

Styrprogram med konfigurerbar logik för spårtrafik

LINN LEIULFSRUD
JOHAN SJÖBERG

Institutionen för Data- och Informationsteknik
CHALMERS TEKNISKA HÖGSKOLA

Göteborg 2016

Styrprogram med konfigurerbar logik för spårtrafik
LINN LEIULFSRUD
JOHAN SJÖBERG

Examinator: Peter Lundin

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

Omslag:
Exempel på järnväg med logik.

Institutionen för Data- och Informationsteknik
Göteborg 2016

Styrprogram med konfigurerbar logik för spårtrafik

LINN LEIULFSRUD
JOHAN SJÖBERG
Institutionen för Data- och Informationsteknik, Chalmers Tekniska Högskola

Examensarbete

SAMMANFATTNING

Object Controller System (OCS) är ett datorsystem utvecklat av Bombardier Transportation. OCS styr och rapporterar status för exempelvis spårväxlar och andra objekt på järnväg. Systemet kommunicerar med ett externt styrsystem (Traffic Control Centre, TCC) som kontrollerar att banan alltid befinner sig i tillåtet tillstånd. TCC är mycket komplext och är inte alltid nödvändigt för enklare, geografiskt begränsade banområden. I detta arbete har en prototyp utvecklats där OCS utökats för att fristående kunna tillse att sådana begränsade banområden alltid befinner sig i tillåtet tillstånd. Funktionaliteten har uppnåtts genom att implementera möjligheten att definiera logiska regler för banområden i konfigurationsfilen för OCS. Vid exekvering validerar programmet både ordrar från externa system samt tillståndet på banan mot dessa säkerhetsregler. Dessa regler är fullt konfigurerbara och anpassas efter bangården. I denna prototyp har inte samtliga typer existerande bangårdsobjekt implementerats, utan endast ett antal grundläggande objekttyper. Arbetet har utförts på Bombardier RCS/EAPD i Göteborg.

Nyckelord: Spårtrafik, styrlogik

ABSTRACT

Object Controller System (OCS) is a computer system developed by Bombardier Transportation. OCS controls and reports the status of objects such as points and other objects on a railway. The system communicates with an external control system (Traffic Control Centre, TCC) which ensures that the track is always in an allowed state. TCC is very complex and not always necessary for simpler, geographically limited railway areas. In this project, a prototype has been developed in which OCS has been extended with functionality to be able to independently ensure that the track is always in an allowed state. The functionality has been achieved by implementing the possibility of defining logical rules for the railway area in the configuration file of the OCS. During runtime, the program validates both orders from external systems and the state of the track against these safety rules. These rules are fully configurable and are adapted to the railway area. In this prototype not all types of existing railway objects have been implemented, only a number of essential object types. The project has been carried out at Bombardier RCS/EAPD in Göteborg.

Keywords: Rail traffic, control logic

FÖRORD

Denna rapport är ett examensarbete som genomfördes våren 2016 av studenter vid Chalmers Tekniska högskola på institutionen för Data- och informationsteknik. Arbetet utfördes vid Bombardier Transportations kontor i Göteborg.

Vi vill tacka Christer Carlsson på Chalmers för handledning av rapportskrivandet samt Anders Palmér och Nicola Bottini vid Bombardier för all teknisk handledning under arbetet. Vi vill även tacka vår studiekamrat Quang Luong för det goda samarbetet vid Bombardier. För förmedlingen av kontakten med Bombardier vill vi tacka Susanne Andersson på Tritech. Slutligen vill vi tacka Torbjörn Hildesson vid Bombardier för att vi fick möjligheten att göra detta examensarbete.

Göteborg juni 2016
Linn Leiulfsrud, Johan Sjöberg

INNEHÅLLSFÖRTECKNING

FÖRKORTNINGAR OCH TERMINOLOGI

Ord	**Förklaring**
ACP	Assertion Check Package.
CCU, CCU6	Communication Controller Unit. Kontrollkort med datorsystem. Del av OCS.
CTC	Centralised Traffic Control. Fristående användargränssnitt som kommunicerar med TCC.
Derailer	Spårspärr på engelska.
INTERFLO	Samlingsnamn för en familj av integrerade tågkontrollsystem. Innefattar både trackside och ombordsystem.
OC	Object Controller. Kontrollkort som hanterar enskilt objekt på bangård. Del av OCS.
OCS, OCS950	Object Controller System. System som hanterar objekt på bangård. Systemet placeras tackside.
Set of points	Spårväxel på brittisk engelska.
Spårspärr	Säkerhetsanordning placerade på spåren. I aktivt läge hindrar spårspärren vagnar och lok att passera genom att spåra ur dem. I passivt läge påverkas inte trafiken. Under tiden som spårspärren byter läge är den i flyttande läge.
Spårväxel	Ställbar förgrening av spårväg där huvudspår förgrenas till två spår. I vänster- respektive högerläge leds trafiken till det vänstra respektive högra spåret sett ifrån huvudspåret. Under tiden som spårväxeln byter läge är den i flyttande läge. Spårväxel kallas ibland växel.
Switch	Spårväxel på amerikansk engelska.
TCC	Traffic Control Center. Central del i INTERFLO-systemen. Sköter traditionellt alla beräkningar och kommunikation med andra delar så som OCS och CTC.
Trackside	Utrustning placerad längs med rälsen.
Wayside	Utrustning som inte är placerad trackside eller ombord ett tåg.
XML	Extensible Markup Language.

1 INLEDNING

1.1 Bakgrund

Bombardier Transportation är världens största tillverkare av tåg och är en del av den kanadensiska koncernen Bombardier som har ca 39 400 anställda världen över [2]. I Sverige har Bombardier exempelvis levererat Öresundståg (X31), regionaltågen av typ Regina (X50-X54) och tunnelbanevagn C20 [7].

Arbetet utfördes på avdelningen Bombardier RCS/EAPD i Göteborg. Denna avdelning utvecklar säkerhetskritisk mjukvara för signalsystem för tågtrafik. I dagsläget används deras produkter främst till gruvtåg, till exempel i LKAB-gruvan i Kiruna. Dessa gruvtåg är förarlösa och styrs av ett högautomatiserat system samt en operatör som övervakar tågen [8].

Göteborgskontorets huvudprodukter heter INTERFLO 150 och INTERFLO 550. En av delarna i dess system är Traffic Control Centre (TCC). TCC kommunicerar via olika protokoll över TCP/IP med andra komponenter i signalsystemet, och ansvarar därigenom för all säkerhetslogik som säkerställer att inga spårväxlar står i en position som kan leda till olyckor. En operatör kan ge ordrar till TCC via det grafiska användargränssnittet Centralised Traffic Control (CTC).

Object Controller System (OCS) är systemet som direkt hanterar alla objekt i banområdet såsom spårväxlar, spårspärrar, etc. Varje sådant objekt är kopplat till en Object Controller (OC), och varje OC-kort är kopplat till en Communication Controller Unit (CCU6). OCS innehåller en CCU6 och upp till 16 OC-kort. En CCU6 kan alltså kontrollera flera OC-kort, medan ett OC-kort normalt sett kontrollerar ett objekt samt objektets in- och utgångar. CCU6 i sin tur kommunicerar med TCC. Via denna kommunikation får OCS ordrar från TCC (se Figur 1.1). En mottagen order skickas vidare till det objekt som ordern avser.

Figur 1.1 Hur det befintliga systemet fungerar.

På geografiskt begränsade banområden, som till exempel banområden för underhåll av lok, används i dagsläget ofta inga eller få automatiska säkerhetssystem. Personalen får i många fall slå om växlar och andra objekt manuellt, antingen via ett användargränssnitt eller fysiska knappar. Det finns då få eller inga automatiska säkerhetskontroller. Att installera TCC är inte optimalt, eftersom TCC är ett onödigt komplext och dyrt system för denna sorts banområden. I dessa fall skulle det vara användbart att kunna kontrollera OCS direkt från användargränssnittet CTC utan att behöva använda TCC. I nuläget innehåller inte OCS någon

säkerhetslogik. För att skapa ett system där CTC kontrollerar OCS behöver OCS utökas med säkerhetslogik. Denna säkerhetlogik skall inte vara lika omfattande som logiken i TCC.

1.2 Syfte

Syftet med detta arbete är att undersöka möjligheterna för att utöka OCS med nödvändig säkerhetsfunktionalitet genom att ta fram en prototyp av ett sådant system.

1.3 Avgränsningar

I detta arbete ingår endast mjukvaruutveckling. Hårdvaran för systemets olika delar är de som Bombardier tillhandahåller.

Hela systemet som skall utvecklas består av både CTC och OCS, medan detta projekt är begränsat till OCS-delen. CTC-delen kommer att utvecklas i ett parallellt projekt. Dessa båda delar kommer sedan att kopplas ihop för att tillhandahålla den kompletta funktionalitet som uppdragsgivaren efterfrågar.

1.4 Precisering av frågeställningen

Arbetet omfattar att skapa en prototyp för kommunikation direkt mellan CTC och OCS, samt säkerhetslogik i OCS för att kunna motverka olyckor på mindre banområden som inte kräver en fullskalig TCC. Prototyper av detta system skall utvecklas genom att modifiera den existerande mjukvaran för OCS, som sedan skall kommunicera med CTC för att utgöra det slutliga systemet.

Det kompletta systemet skall fungera som en mindre version av CTC-TCC-OCS-systemet. CTC och OCS konfigureras med en konfigurationsfil som beskriver banområdet samt de logiska regler som gäller för säkerheten. CTC tar sedan in order från operatören och skickar dessa vidare direkt till OCS. OCS avgör om ordern får utföras utifrån den säkerhetskonfiguration som är inladdad. Om det är tillåtet skickas lämpliga styrsignaler till det objekt som ordern avser. Det skall även vara möjligt att skapa säkerhetsregler som gör att CCU6 vid vissa tillstånd automatiskt skickar order till objekt att byta läge, utan en operatörs inverkan. OCS skall också kunna skicka information från objekten tillbaka till CTC där förändringar i banan presenteras grafiskt för operatören.
Den framtagna prototypen skall testas och exekveras mot en simulation av ett banområde.

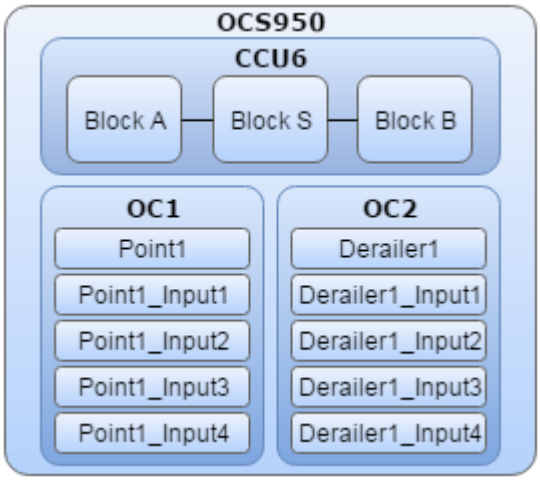
2 TEKNISK BAKGRUND

2.1 OCS

OCS används för att hantera styrning av utrustning placerad längs med rälsen. I sammanhanget kallas denna utrustning objekt. OCS950 är en OCS-enhet som består av ett CCU6-kort och upp till 16 OC-kort. Varje OC-kort kontrollerar ett objekt samt dess in- och utgångar (se Figur 2.1). CCU6-enheten innehåller tre stycken field-programmable gate arrays (FPGA) kallade block A, block B och block S på vilka Linux exekveras på soft microprocessors. Feltolerans är kritiskt i sammanhanget, vilket återspeglas i hela systemets design. Block A och block B kommer från två olika tillverkare. Detta för att minimera risken att fel i hårdvaran leder till likartade misstag på båda blocken. Koden i samtliga block är skrivna i C.

Block S kommunicerar med TCC över en nätverkslänk och tar därifrån emot ordrar över TCC-OCS, vilket är ett protokoll utvecklat av Bombardier. Block S skickar vidare dessa inkommande ordrar till block A och block B (som exekverar ekvivalenta men olika styrprogram för sina respektive plattformar) vilka tolkar ordern och skickar tillbaka ett svar till block S över protokollet OCLink. OCLink är utvecklat av Bombardier.

Block S kontrollerar om block A och block B har nått samma resultat. Överensstämmer inte resultaten stängs OCS av. Överensstämmer resultaten skickar block S vidare ordern till det OC-kort som kontrollerar det objekt som ordern avser. Ett OC-kort innehåller nödvändiga elektroniska komponenter för att kommunicera med objekt.



Figur 2.1 Exempel på OCS med spårväxel och spårspärr.

Den huvudsakliga mjukvaran som utvecklas i detta projekt exekveras i block A och block B i CCU6.

2.2 Spårvägar

De bangårdsobjekt som framför allt kommer att ingå i detta arbete är spårväxlar (points) och spårspärrar (derailers). En spårväxel är en förgrening av spår som kan ställas i olika lägen för att styra inkommande tåg till de olika grenarna av spåret [6]. En spårspärr är en säkerhetsanordning som sitter placerad på spåret. När spårspärren är i aktivt läge spårar den ur tåg som kör över den. Spårspärrar används för att spårvägsfordon inte skall kunna rulla ut på spår där de kan orsaka olyckor [6].

2.3 XML

Extensible Markup Language (XML) är ett textformat vars syfte är att utbyta data mellan informationssystem. XML är ett märkspråk som definierar regler för att skapa dokument som är läsbara för såväl datorer som människor [9]. Bombardiers konfigurationsfiler och datamodeller är XML-dokument.

En konfigurationsfil byggs upp av *objekt*. I detta sammanhang syftar inte uttrycket objekt endast till bangårdsobjekt, utan till delar av konfigurationen. Ett objekt kan t.ex. representera en OC, en logisk regel eller en spårväxel.

För att framställa en konfigurationsfil behövs en datamodell som definierar de olika typer av objekt som skall ingå i konfigurationen. Objekttyper definieras med hjälp av taggar som beskriver typens attribut (se Figur 2.2). När datamodellen är framställd kan en konfigurationsfil skapas som använder dessa objekttyper för att definiera specifika objekt (se Figur 2.3).

```
<type>
    <OBJECT_TYPE>Points</OBJECT_TYPE>
    <prop>
        <DESCRIPTION>OCS950 Points</DESCRIPTION>
        <PARENT>ControllableObject</PARENT>
        <VIRTUAL>0</VIRTUAL>
        <VISIBLE>1</VISIBLE>
        <PICTURE>NULL</PICTURE>
        <HELP>NULL</HELP>
        <MIN_NO>NULL</MIN_NO>
        <MAX_NO>NULL</MAX_NO>
        <NAME_FORMAT>[^/]{1,15}</NAME_FORMAT>
    </prop>
</type>
```

Figur 2.2 Exempel på en objekttyp i datamodellen.

```
<Object>
    <ObjectType>Points</ObjectType>
    <ObjectName>Point1</ObjectName>
    <Parameter>
        <Name>owner</Name>
        <Value>
            <ReferenceValue>
                <ObjectType>PointsOC</ObjectType>
                <ObjectName>OC1</ObjectName>
            </ReferenceValue>
        </Value>
    </Parameter>
    <CheckSum>0713784D</CheckSum>
</Object>
```

Figur 2.3 Exempel på ett objekt i konfigurationsfilen.

3 METOD

3.1 Verktyg och metodik

För att utveckla prototypen kommer vi främst att modifiera och utveckla kod för CCU6. Programmeringen kommer att ske i programmeringsspråket C då detta är språket som existerande programkod för CCU6 är skriven i. Bombardier kommer att tillhandahålla datorer under arbetets gång vilka har Windows installerade. Projektet kommer att byggas i en virtuell maskin med en Ubuntu-baserad miljö som har kompilatorer för målplattformen. Arbetet kommer alltså att utföras i en blandad miljö av Windows och Linux. Utveckling kommer att ske främst i texteditorn Visual Studio Code då editorn visat sig fungera väl och har Git-integration [3].

Arbetet kommer att genomföras i Bombardier RCS/EAPDs kontor i Göteborg, med stöd av handledare från företaget. I projektets inledande fas kommer arbetet att delas upp i förutsedda deluppgifter och en tidsuppskattning för dessa delar kommer att sammanställas.

Delvis kommer utvecklingen att ske i form av parprogrammering med nära samarbete inom gruppen. Till största delen kommer gruppmedlemmarna att arbeta från sina platser i kontoret för att lätt kunna nå varandra, diskutera och arbeta tillsammans. En allmän agile utvecklingsmetod kommer att användas.

Gerrit är ett gratis webbaserat verktyg som är integrerat med Git och som används för versionshantering och kodgranskning [4]. Git är det huvudsakliga verktyget för versionshantering som används på Bombardiers Göteborgskontor och kommer att användas under projektet.

Bombardiers verktyg DMP används för att från en datamodell i XML-format utvinna en fil med SQL-frågor. Denna fil laddas sedan in i Bombardiers verktyg DPT, vilket för in datamodellen i en databas. DPT används sedan för att skapa konfigurationsfiler i XML-format.

Under utvecklingen krävs en del simulering. Innan det parallella projektet som utökar CTC är färdigt behövs simulatorer för att skicka ordrar till OCS. Ett antal simulatorer behövs också för att simulera OC samt objekt. Bombardier har utvecklat simulatorer för dessa ändamål och tillhandahåller dessa under projektets gång. För simulering av sändning av ordrar används OCHostTool. För simulering av OC och objekt används OCSim och OCSimGUI.

Kunskapsinhämtning kommer att ske på ett antal olika sätt, främst via handledarna, dokumentation som Bombardier tillhandahåller, Internet, samt litteratur.

Bombardier använder en kodstandard som i största möjliga mån skall följas under arbetet. Denna standard innefattar bland annat att minnesallokering och rekursion inte tillåts när

styrprogrammet är klar med inläsning av konfigurationsdata och initiala allokeringar av resurser.

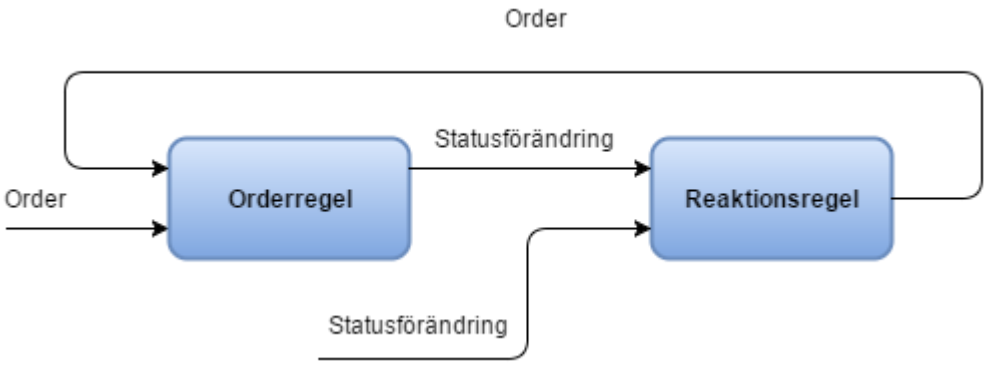
3.2 Förutsedda deluppgifter

3.2.1 Datamodellen

En datamodell för CCU6 för konfigurerbar säkerhetslogik skall skapas. Logiken skall definiera vilka styrsignaler som kan skickas från operatören till objekten baserat på objektens tillstånd. Denna logik kommer hädanefter att refereras till som *orderregler*. Orderregler används vid inkommande order från operatören. Funktionalitet skall även utvecklas för att CCU6 automatiskt skall kunna skicka vissa signaler till sina anslutna objekt om vissa tillstånd uppfylls. Denna logik kommer hädanefter att refereras till som *reaktionsregler*. Reaktionsregler evalueras automatiskt utan att läsa några ordrar från användaren. Syntax för datamodellen skall designas.

En regel innehåller en handling och ett villkor. Om regeln är en orderregel motsvarar handlingen den inkommande ordern och villkoret beskriver tillstånd som måste vara uppfyllda för att ordern skall accepteras. Om ingen orderregel finns för en inkommande order accepteras inte ordern.

Om regeln är en reaktionsregel motsvarar handlingen en reaktion som utförs när regelns villkor är uppfyllt. Reaktionen är en styrsignal som skickas till det objekt som definieras i handlingen.



Figur 3.1 Flöde av regelhantering.

3.2.2 Konfigurationer för test

Konfigurationer skall skapas i den nya datamodellen för att möjliggöra testning av logiken under utvecklingens gång. Den framställda datamodellen skall laddas in i DPT och en konfigurationsfil för en simulerad bangård skall framställas i DPT och sedan exporteras. Denna konfiguration skall innehålla all nödvändig data för att simulera de objekt som finns på en bangård. Konfigurationen skall även innehålla logik som berör dessa objekt. Den exporterade konfigurationsfilen importeras därefter till CCU6 och CTC.

3.2.3 Representation av logikmodellen i C

En representation av logikmodellen i C skall utvecklas. Detta kräver att de delar i konfigurationsfilen som behövs för att genomföra önskad logikkontroll läses in och sparas i en C-representation. De delar som behöver sparas innefattar logikreglerna som tagits fram under projektet.

3.2.4 Evaluering av villkor

Funktionalitet skall utvecklas för att jämföra inkommande ordrar mot konfigurationens orderregler. Likaledes skall status från bangårdens objekt jämföras mot konfigurationens reaktionsregler.

3.2.5 Skicka in- och utdata via logikmodellen

Kod skall skrivas för att in- och utdata skall kunna skickas till och från OCS. Detta omfattar flera delar. Ordrar som sänds från CTC till CCU6 skall skickas till de funktioner som jämför ordern mot orderreglerna. Funktionalitet skall skapas för att skicka ordrar till objekten i bangården, både när ordern kommer från CTC och när den kommer från en reaktionsregel. Status från objekten skall läsas in för att kunna kontrollera objektens status mot reaktionsreglerna. En inkommande order skall vidarebefordras till OC endast om logiken tillåter det, annars skall en varning skickas tillbaka till CTC.

3.2.6 Uppdatera ACP

Möjlig extrafunktionalitet som kan utvecklas är att uppdatera Assertion Check Package (ACP). ACP är designad för att verifiera att OCS inte kan skicka ut någon data som kan orsaka att systemet går in i ett osäkert läge. ACP är utformat som en separat modul bestående av testkod.

3.2.7 Virtuella in- och utgångar

Ytterligare extrafunktionalitet som kan utvecklas är virtuella in- och utgångar. Med dessa skulle tillståndsmaskiner för objektlägen kunna skapas och därigenom skulle konfigurationer med mer avancerad funktionalitet kunna defineras.

4 KONSTRUKTION

4.1 Uppstart

Arbetet inleddes med att handledarna och gruppen träffades, tillsammans med gruppen som genomförde det parallella projektet som gäller CTC. Arbetsuppgifter och kravspecifikationer gicks igenom och fastställdes. Arbetet bröts ned i delproblem och planering lades upp kring arbetsmetodik, samarbete och tidsplanering.

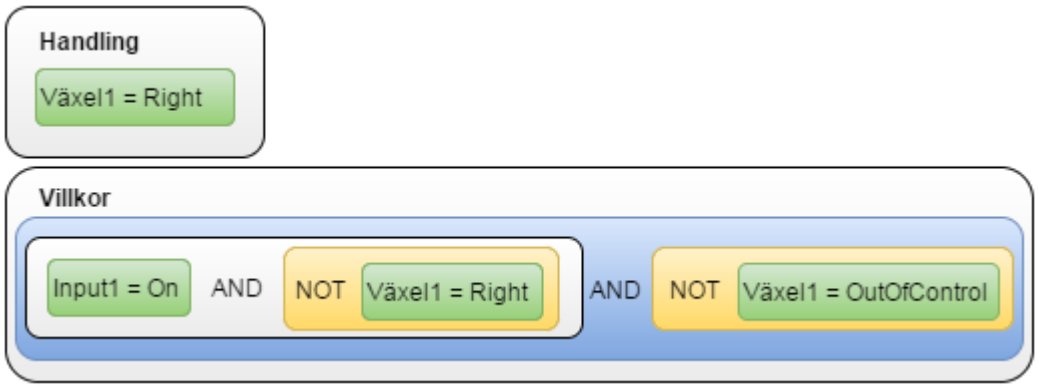
Gruppen gavs tillgång till källkoden för CCU6. Denna kod undersöktes och tillsammans med handledare diskuterades i grova drag hur CCU6 fungerar och hur CCU6 kommunicerar med bangårdsobjekt samt TCC.

4.2 Datamodellen

Företaget använder XML för konfigurationsfiler, och därför är det ett krav att datamodellen framställs i XML. Denna datamodell skall hålla konfigurerbar logik som beskriver förhållandet mellan tillstånd och tillåtna ordrar för bangårdsobjekten.

Handlingar och villkor implementerades i datamodellen. En handling är en signal som skickas från CCU6 till en OC för att en förändring skall utföras på tillhörande objekt, t.ex. att en spårväxel slår om till ett annat läge eller att ett alarm aktiveras. En handling måste alltså hålla en referens till ett objekt samt den status som objektet skall försättas i. Ett villkor är ett logiskt uttryck uppbyggt av logiska operatorer samt jämförelser mellan ett objekts nuvarande tillstånd och ett förväntat tillstånd. Villkoret måste uppfyllas för att dess tillhörande handling ska utföras.

I Figur 4.1 är handlingen att slå om Växel1 till läge Right, med villkoret att denna handling endast utförs om Input1 är On, och Växel1 inte är Right och inte är OutOfControl.



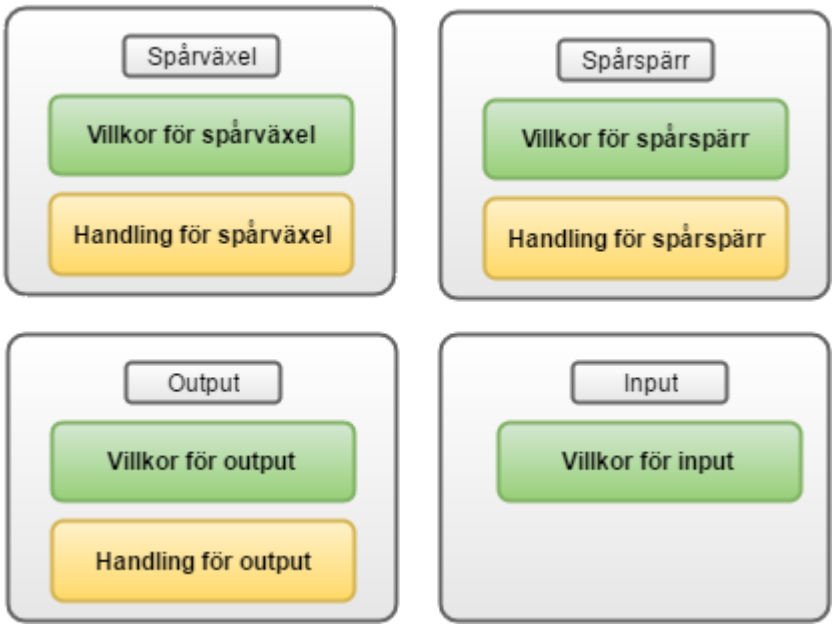
Figur 4.1 Exempel på en regel med tillhörande handling och ett villkor.

Ett villkor kan vara ett sammansatt uttryck med ett godtyckligt antal delvillkor. Designen av villkoren krävde därför en hel del eftertanke. En variant implementerades först där ett villkor

tar en godtyckligt lång lista av operander. Det ansågs senare mer lämpligt att ett villkor skall ha ett bestämt antal delvillkor beroende på typ. De logiska uttrycken AND, OR och XOR har två operander, NOT har en operand, och de villkor som representerar objektstatus har endast en referens till ett objekt samt önskad status för detta objekt.

Dessa villkor och handlingar används för att bygga upp orderreger och reaktionsregler. För att kunna utöka funktionaliteten i CCU6 med dessa regler gav Bombardier tillgång till en konfigurationsfil för CCU6. Denna konfigurationsfil modifierades sedan genom att en subklass till den ursprungliga CCU6 skapades. Denna subklass fick namnet LOCS_CCU6 (Local Object Control System CCU6) och innehöll förutom det ärvda innehållet även en lista för regler.

XML-syntax skapades för de olika objekttyper som behövs för LOCS_CCU6. Bombardier tillhandahöll information om alla sorters ordrar och status som kan skickas till och från objekten på bangården. Inte alla existerande typer av objekt skulle implementeras i LOCS_CCU6. Endast de som önskades av Bombardier markerades ut och implementerades. De slutgiltiga objekttyperna beskrivs av Figur 4.2.



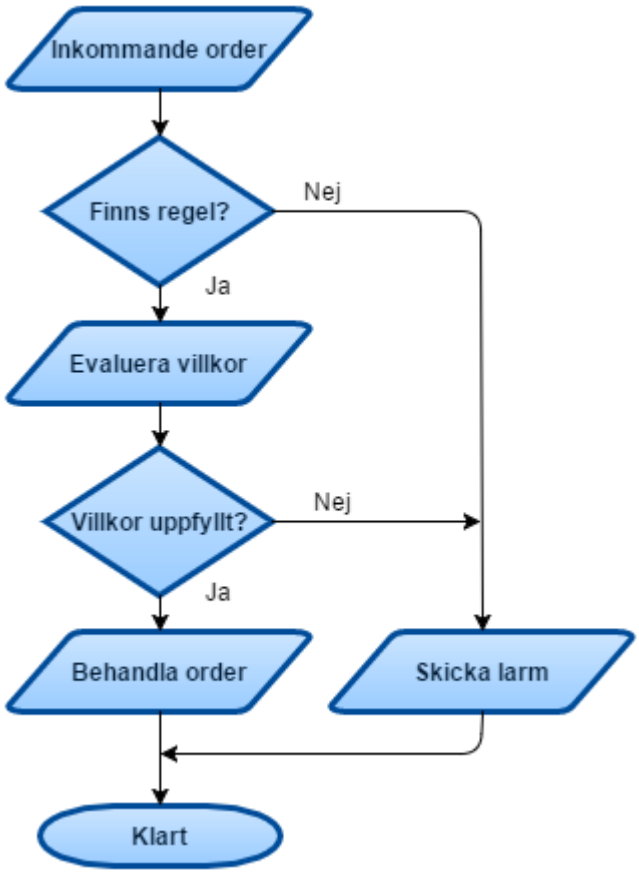
Figur 4.2 Samtliga objekttyper som implementerats.

Input har ingen objekttyp för handling. Detta beror på att en Input är en extern insignal till ett objekt. CCU6 kan därför inte skicka styrsignaler till en Input utan endast hämta status.

I datamodellen behövde en definition skapas för hur en status tolkas och representeras. I Bombardiers konfigurationsfiler representeras ett tillstånd med enum. Därför togs beslutet att representera status på detta sätt för att förenkla integration av den ursprungliga CCU6 med säkerhetslogiken.

Hur handlingar och villkor används och utvärderas i CCU6 beror på om det är en orderregel eller reaktionsregel.

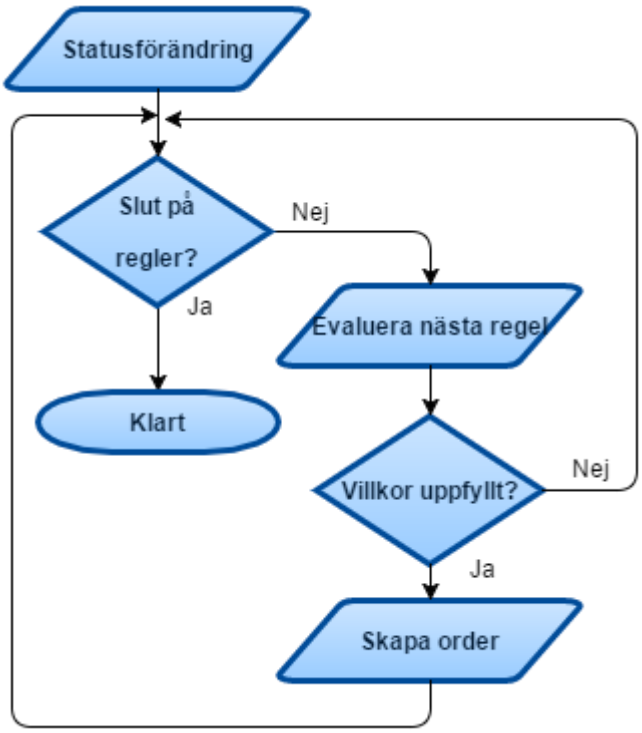
En orderregel hanterar inkommande ordrar från CTC eller från en reaktionsregel enligt Figur 4.3. Om konfigurationen innehåller en orderregel vars handling motsvarar ordern undersöks regelns villkor. Om villkoret är uppfyllt skickas ordern vidare till objektet. Annars skickas ett larm till CTC.



Figur 4.3 Flödesschema för hantering av en inkommande order.

Om exemplet i Figur 4.1 är en orderregel skall den hanteras på så sätt att när CCU6 tar emot en order om att ställa Växel1 i läge Right, undersöks om villkoret är uppfyllt. Om villkoret är uppfyllt skickas styrsignaler till Växel1 att ställa om till läge Right.

En reaktionsregel hanterar statusförändringar från bangårdens objekt enligt Figur 4.4. Vid statusförändring hos ett objekt jämförs objektets nya status mot villkoren i reaktionsreglerna. En statusförändring kan vara relevant för flera olika reaktionsregler. Därför behöver en uppslagning göras där alla berörda reaktionsregler hittas.



Figur 4.4 Flödesschema för hantering av en statusförändring.

Om exemplet i Figur 4.1 är en reaktionsregel skall CCU6 läsa in status från objekt på bangården vid statusförändring. Om Input1 då har läge On, och Växel1 inte är i läge Right eller läge OutOfControl, skickar CCU6 automatiskt styrsignaler till Växel1 att ställa om till läge Right.

En ny databas för LOCS_CCU6 skapades på en dator som Bombardier tillhandahöll. Verktyget DMP (se avsnitt 3.1 Verktyg och metodik) användes för att från den framställda konfigurationsfilen utvinna en fil med SQL-frågor. Via DPT (se avsnitt 3.1 Verktyg och metodik) lades SQL-filen för datamodellen in i databasen.

4.3 Konfigurationer för test

För att testa prototypen behövde konfigurationsfiler skapas med DPT. Till en början definierades några grundläggande konfigurationsfiler innehållande ett mindre antal objekt. Logikregler skapades med minst en av varje objekttyp. Arbete gällande inläsning och behandling av konfigurationen påbörjades med hjälp av dessa grundläggande filer. Senare i projektet behövde en mer komplett konfiguration framställas för testning. Handledarna gav exempel på vilka sorters scenarion som skulle behöva testas och vilka kombinationer av objekt som typiskt finns på en bangård. En skiss över en bangård ritades upp och objekt sattes in i ritningen. Säkerhetslogik för banområdet designades och översattes till orderregler och reaktionsregler. Till sist lades denna design in i DPT och en slutgiltig konfigurationsfil framställdes.

4.4 Representation av logikmodellen i C

En representation av logikmodellen behöver skapas i C för att LOCS_CCU6 skall kunna evaluera logiken. För att skapa denna representation behöver de nödvändiga delarna läsas ut ur konfigurationsfilen. För att konfigurationen skall kunna läsas in av LOCS_CCU6 krävs att konfigurationsfilen är framställd med DPT. I konfigurationsfilen representeras objekten och logikmodellen av ett syntaxträd. Syntaxträdet traverseras och läses in med en något modifierad version av libXML2 som Bombardier tillhandahåller [1].

Den ursprungliga CCU6 har redan funktionalitet för att läsa in en konfigurationsfil. Denna kod behövde utökas så att även listan av logikregler läses in. Logikreglerna representeras i konfigurationsfilen av noder i syntaxträdet. För att implementera funktionalitet för att leta upp och läsa in dessa specifika noder krävdes kunskapsinhämtning kring XML [9] och parsning med libXML2 [1][5]. Ett antal hjälpfunktioner skapades för att traversera och läsa in XML-trädet mer effektivt för programmets ändamål. Med hjälp av dessa stödfunktioner söks konfigurationsfilen igenom efter de relevanta delarna.

För att lagra logikmodellen i CCU6 behövs datastrukturer för logikregler. Eftersom både orderregler och reaktionsregler har samma innehåll (en handling och ett villkor) valdes en generell datastruktur för båda typerna av regel.

En handling innehåller i konfigurationsfilen en referens till ett bangårdsobjekt samt ett önskat läge för detta objekt. En handling behöver därför representeras i C som en datastruktur innehållande en objektreferens och en objektstatus. I konfigurationsfilen är dock referensen till bangårdsobjektet endast en sträng innehållande objektets namn. Hur detta strängnamn skulle kopplas till en objektreferens krävde en del designbeslut. Ett antal alternativ diskuterades. En första version implementerades där strängen tilldelas ett löpnummer, men denna version används inte i slutprodukten (se vidare kapitel 6 Slutsatser).

Vid undersökning av CCU6:s källkod framkom det att OCS redan har ett antal fördefinierade sätt att referera till bangårdsobjekt. CCU6 refererar till objekt som datastrukturer som skapas vid inläsning av konfigurationen. OC refererar till objekt med hjälp av två heltal, ett för objektets OC och ett för objektets index i dess OC. OC:s sätt att referera har vissa fördelar, eftersom den virtuella maskinen för evaluering av logiken använder sig av heltal (se vidare kapitel 4.5 Evaluering av logikregler). CCU6:s datastrukturer för objekt innehåller dock mer information än OC:s referenser, däribland information som behövs för logiken. Om OC:s referenser användes skulle denna information behöva hämtas på annat sätt. Därför togs beslutet att använda CCU6:s sätt att referera till objekt. En funktion implementerades också för att översätta mellan objektets referens och ett heltal för den virtuella stackmaskinens skull.

För att använda CCU6:s referenser till objekten krävs ett sätt att från ett objekts namn nå den datastruktur som motsvarar objektet i CCU6. En funktion som redan finns i den ursprungliga CCU6 ger denna funktionalitet, där ett objekts namn returnerar en pekare till den sökta datastrukturen.

Ett objekts status representeras i CCU6 av ett enumvärde. Varje objekttyp har en egen enumtyp som representerar status. När objektstatus skall lagras i LOCS_CCU6 behöver därför rätt enumtyp och enumvärde för objekttypen identifieras. Vid inläsningen av konfigurationsfilen kontrolleras därför objekttypen samt status så att rätt data lagras.

Ett villkor är mer komplicerat att läsa in ur XML-filen än en handling. Detta beror på att ett sammansatt villkor i konfigurationsfilen har flera nivåer i syntaxträdet. Inläsning av villkoret görs därför lättast med rekursion. Eftersom rekursion får användas vid startup (se kapitel 3 Metod) så är detta inte ett problem och rekursiv inläsning av villkor implementerades därför. Objektreferens samt objektstatus representeras på samma sätt som beskrivits för handlingar.

Representationen av ett villkor kräver en mer komplicerad datastruktur än en handling (se avsnitt 4.2 Datamodellen). Datastrukturen för villkor har olika utformning beroende på vilken typ villkoret har. Datastrukturen har ett enumvärde som definierar vilken typ villkoret har. Datastrukturen som representerar villkoret NOT innehåller endast en pekare till ett delvillkor. Datastrukturen som representerar ett sammansatt villkor innehåller två pekare till vart och ett av delvillkoren. Ett villkor som endast representerar en objektstatus innehåller två heltal, där den ena är ett id-nummer och den andra representerar önskat tillstånd. Att heltal används beror på implementationen av stackmaskinen som evaluerar villkor (se Kapitel 4.5 Evaluering av logikregler).

En datastruktur (kallad Environment) skapades för att hålla de inlästa reglerna. Datastrukturen behövde placeras på en lämplig plats. CCU6 innehåller redan en datastruktur som lagrar information om bl.a. sina objekt och sin konfiguration. Environment lades därför in i denna datastruktur.

Eftersom orderregler och reaktionsregler behandlas olika innehåller Environment en lista för respektive typ av regel. Vilken lista en logikregel placeras i avgör vilken typ den har och hur den skall behandlas av programmet. För att kunna använda listor skrevs en fältbaserad listdatatyp. Logikreglerna placeras i respektive lista under inläsningen av konfigurationsfilen.

Koden som framställdes i denna del av arbetet exekveras i både block A och block B (se avsnitt 2.1 OCS). De båda blocken refererar av säkerhetsskäl till objekt och status med olika addresser och nummer. Därför behöver olika versioner av logiksystemet kompileras för de respektive blocken.

All minnesallokering har gjorts med ett makro definierat i CCU6. Detta har gjorts för säkerhetsskäl i enlighet med den befintliga kodbasen. Felhantering har gjorts med hjälp av ett annat makro.

4.5 Evaluering av logikregler

För att evaluera logiken behöver sanningsvärdet av en regels villkor utvärderas. Villkoret representeras av en datastruktur som skapats efter inläsning av konfigurationsfilen (se Kapitel 4.4 Representation av logikmodellen i C). För orderregler evalueras villkoret när en order kommer in till CCU6. För reaktionsregler sker evalueringen när en statusförändring har inträffat hos ett objekt. Evalueringen sker på samma sätt för de två olika typerna av regler.

För att undersöka att logikmodellen fungerade som önskat utvecklades en första prototyp av evaluering som nyttjar rekursion (se vidare kapitel 6 Slutsatser). I den slutgiltiga versionen implementerades en stackbaserad virtuell maskin för att inte bryta mot Bombardiers kodningsstandard. Vid inläsning av villkor i konfigurationsfilen kompileras villkorens syntaxträd till ett program för en tillämpningsspecifik virtuell stackmaskin. Med statisk analys kontrolleras att det resulterande programmets maximala stackdjup inte överskrider den virtuella maskinens allokering. Kompilering av dessa träd är rekursiv, men eftersom denna kompilering endast exekveras vid uppstart är detta tillåtet enligt kodstandarden som används.

4.6 Skicka in- och utdata via logikmodellen

In- och utdata behöver skickas genom logikmodellen för att uppnå önskat resultat. Detta kräver separata metoder för orderregler och reaktionsregler.

4.6.1 Orderregler - Inkommande ordrar

Orderregler skall kontrolleras vid inkommande ordrar. För att kunna implementera detta krävs förståelse kring hur CCU6 hanterar ordrar.

En order består av ett paket innehållande önskad status samt en objektreferens. När ett sådant paket tas emot av CCU6 vidarebefordras paketet till de funktioner som skickar styrsignaler till objekt. Evaluering av logiken behöver alltså ske innan dessa styrfunktioner anropas. Anrop av evalueringsfunktionen utförs därför innan anrop av styrfunktionerna. Logiken skall undersöka om ordern får utföras och ett sanningsvärde skall returneras som anger om ordern får utföras eller inte. Om ordern får utföras anropas styrfunktionerna. Annars skickas ett varningsmeddelande tillbaka till CTC om att ordern blockerats av logiken.

När en order kommer in skall CCU6 kontrollera om det finns någon regel för ordern. En metod är att söka igenom alla orderregler i konfigurationen. Detta är dock ineffektivt om konfigurationen innehåller många regler. Därför gjordes designvalet att vid inläsning av konfigurationsfilen associera varje orderregel till den typ av order som regeln avser. På så sätt kan CCU6 ta ut och evaluera endast den orderregel som är relevant för en inkommande order, om en sådan orderregel finns. För att införa denna mappning behövs en datastruktur att lagra mappningen i.

Datastrukturen för mappningen implementerades i form av binära träd. Binära träd valdes framför en struktur av listor på grund av effektivitetsskäl. I detta binära träd associeras varje

objekt till sin eventuella orderregel. Om ingen sådan regel existerar för ett objekt läggs objektets referens inte in i det binära trädet. Trädet placerades i Environment (se Kapitel 4.4 Representation av Logikmodellen i C).

4.6.2 Reaktionsregler – Läsa av bangårdsobjektens status

Reaktionsregler skall skicka signaler till objekt när ett visst tillstånd infaller. En möjlig lösning på detta är att reaktionsreglerna evalueras kontinuerligt för att upptäcka att ett visst tillstånd har uppstått. Detta är dock ineffektivt, speciellt om det finns många reaktionsregler. Det är mer effektivt att reaktionsreglerna endast evalueras vid statusförändring av objekt som ingår i minst en sådan regel. Endast dessa regler bör då evalueras.

CCU6 har redan viss funktionalitet för att upptäcka statusförändringar. Varje gång CCU6 tar emot statusmeddelande från en OC, kontrolleras om denna status är annorlunda från senast lagrade kommando på objektet. Evalueringen av reaktionsregler lades därför in i koden efter denna kontroll, så att evalueringen endast utförs om status förändrats. Den återstående delen av problemet är att det är ineffektivt att evaluera samtliga regler. Endast de regler som rör det objekt vars status har förändrats behöver evalueras. Det skapades därför ett binärt träd likartat det för orderregler, med skillnaden att ett objekt kan vara associerat med flera reaktionsregler. I trädet associeras varje objekt till en lista av de reaktionsregler som objektet ingår i.

Objekttypen Output är ett specialfall. Dessa skickar kontinuerligt meddelande om status till CCU6, oavsett om status förändrats eller inte. Detta gör att eventuella reaktionsregler som innehåller Outputs kommer att evalueras kontinuerligt. Funktionalitet behövde utvecklas för att upptäcka när en Output byter status så att evalueringen kan anropas endast vid statusförändring. Lösningen blev att implementera flanktriggning för Output. Både positiva och negativa flanker ger flanktriggning, dvs. både när Output går från låg till hög och tvärtom. När flanktriggning uppstår evalueras reaktionsreglerna. Under utvecklingen av denna funktionalitet upptäcktes att Output inte alltid lagrar föregående status. Därför skapades en datastruktur i Environment som håller status för alla Outputs. När en styrsignal skickas till en Output att byta läge ändras även lagrad status i denna datastruktur.

Om ett tillstånd i en reaktionsregel evalueras till sant medför detta att handlingen i regeln skall utföras. Hur detta utförs skiljer sig från hur en order skickas vidare för orderregler. För orderregler har ett paket med en order kommit in från CTC och kan därför skickas vidare till objektet. För reaktionsregler har dock ingen order tagits emot och därför finns inte ett paket att skicka vidare. Ett nytt paket måste skapas innehållande ordern. Därför byggs ett paket upp med hjälp av funktioner i CCU6. För att detta skall fungera behöver ACP inaktiveras eftersom det ännu inte uppdaterats för att hantera denna funktionalitet.

5 RESULTAT

Målen som definierades för arbetet har uppnåtts. En prototyp har utvecklats efter specifikationerna. Den existerande mjukvaran för OCS har modifierats för att kunna ta emot ordrar från CTC och kontrollera dessa ordrar mot säkerhetslogik för banan som konfigurerats upp av en användare.

En datamodell har tagits fram för att representera säkerhetslogiken. Två olika typer av logikregler har tagits fram enligt specifikationen. Orderregler används för att kontrollera om inkommande ordrar får genomföras. Reaktionsregler används för att kontrollera om ett visst tillstånd har uppstått och i det fallet skicka handlingar till objekten. Datamodellen har infogats i den ursprungliga datamodellen för konfigurationsfiler. Konfigurationsfiler med regler kan nu framställas och laddas in i OCS samt CTC.

Mjukvaran i OCS har modifierats för att vid inkommande order från CTC undersöka logikmodellen för att se om någon orderregel finns för ordern. Om det finns en orderregel evalueras regelns villkor, och om villkoret är sant tillåts ordern passera vidare för att påverka objektet som ordern gäller. OCS har även modifierats för att vid statusförändring av objekt evaluera objektets eventuella reaktionsregler. Om villkoret i reaktionsregeln är uppfyllt utförs handlingen i regeln automatiskt genom att en order skapas och skickas till objektet som handlingen gäller.

Prototypen har under arbetets gång exekverats mot simulationer av ett antal olika banområden. Denna testning har fungerat väl och bekräftat att programmet fungerar på önskat sätt. I arbetets slutfas har även CTC inkluderats i systemet och ersatt den simulator som tidigare simulerade TCC. CTC (som modifierats enligt det parallella arbetet beskrivet i avsnitt 1.3 Avgränsningar) har samkörts med prototypen för att forma det kompletta system som definierades i projektets mål. Dessa samkörningar har bekräftat att det kompletta systemet fungerar enligt specifikationerna.

Sammanfattningsvis uppfyller den utvecklade produkten de specifikationer som definierades i början av arbetet. Inga deluppgifter behövde utelämnas på grund av tidsbrist och flera delar av programmet optimerades när grundfunktionaliteten var färdig.

5.1 Datamodellen

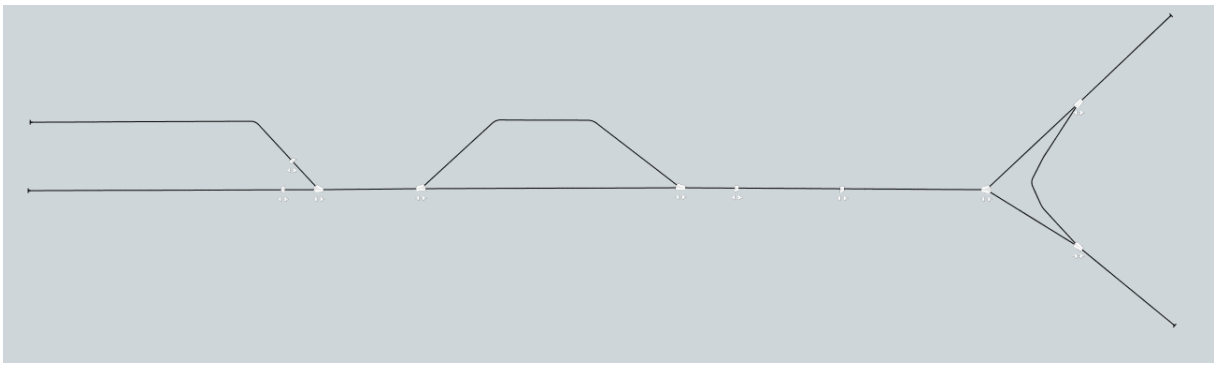
En utökad datamodell med data såsom reaktionsregler, orderregler, handlingar samt villkor har utvecklats. Den utökade datamodellen kan användas med de befintliga verktyg som uppdragsgivaren använder. Datamodellen laddas in i DMP tillsammans med en XML-fil som beskriver anpassade förutsättningar för en specifik järnväg. DMP ger sedan ett komprimerat tar-arkiv. Arkivfilen laddas i sin tur in i DPT från vilket en databas skapas.

I DPT skapar användaren konfigurationer för banområden. Objekt som spår, växlar, spårspärrar och så vidare skapas, och därefter skapas reaktionsregler och orderregler för dessa objekt.

5.2 Konfigurationer för test

En konfiguration för en mindre järnväg har tagits fram för testning. Scenariot innehåller flera olika regler för att olika sorters funktionalitet skall kunna testas. Denna konfiguration laddas in i OCSim som simulerar bangårdens objekt.

Konfigurationen innehåller exempelvis en sluss bestående av två spårspärrar (se Figur 5.1). Reaktionsregler har definierats vilka bevakar att när en av dessa två spårspärrar öppnas så stängs den andra.



Figur 5.1 Konfiguration för testning renderat i CTC.

5.3 Representation av logikmodellen i C

När LOCS_CCU6 laddar in konfigurationsfilen byggs ett associativt binärt träd upp för reaktionsreglerna respektive orderreglerna. I trädet för orderregler associeras en objektreferens samt en status till ett villkor.

En optimering har gjorts för att inte behöva utvärdera alla reaktionsregler på alla statusförändringar. När en statusförändring på ett objekt inkommer till CCU görs en uppslagning på det aktuella objektets id-nummer i trädet av reaktionsregler. En lista med enbart de reaktionsregler vars villkor innefattar det aktuella objektet erhålls.

Inläsning av villkor från konfigurationsfilen sker i två steg. I det första steget traverseras XML-trädet från huvudnoden i villkoret och ett rekursivt parsträd byggs upp. För att tillgodose kraven i kodstandarden att inte använda rekursion efter inladdning tillkom ett steg som kompilerar parsträdet till en bytekod som skapats för detta program.

5.4 Evaluering av villkor

Den virtuella maskinen har tagits fram specifikt för villkorsevaluering i detta program. Utformningen är en stackbaserad maskin med instruktioner för de nödvändiga logiska operationerna samt jämförande av ett objekts status mot ett referensvärde (instruktion CHECK) samt en instruktion för att avsluta programmet (instruktion RET) i den virtuella maskinen. På första position i programmet sparas ett tal som beskriver hur långt programmet är. Innan exekvering kontrolleras att slutet av programmet innehåller en avslutningsinstruktion. Vid kompilering av trädstrukturen som beskriver programmet traverseras trädet postorder och läggs ut som instruktioner.

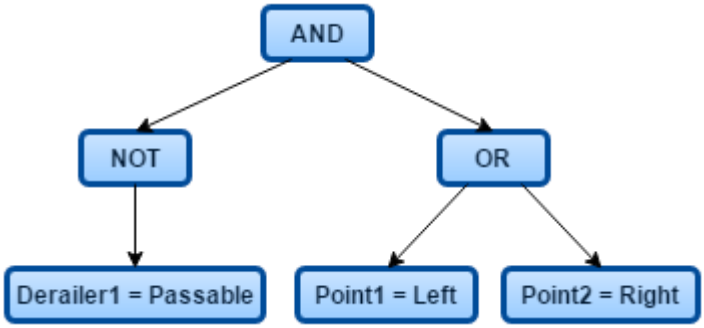
Exempelvis läses följande logiska uttryck in till villkorsträdet i Figur 5.2:

```
(NOT(Derailer1=Passable)) AND
((Point1=Left) OR (Point2=Right))
```

Villkorsträdet i Figur 5.2 kompileras till följande bytekod:

```
14, CHECK, Passable, Derailer1, NOT, CHECK, Left, Point1,
CHECK, Right, Point2, OR, AND, RET.
```

Alla instruktioner, objektreferenser och referensvärden är numeriska värden men skrivs här ut som namn för läsbarhetens skull.



Figur 5.2 Exempel på villkorsträd.

5.5 Skicka in- och utdata via logikmodellen

Orderreglerna kontrolleras vid inkommande order, och om en orderregel finns för ordern evalueras denna regels villkor. Om villkoret är sant vidarebefordras ordern, annars skickas ett varningsmeddelande till CTC.

Reaktionsreglerna kontrolleras vid statusförändring hos ett objekt, och om reaktionsregler finns för objektet evalueras dessa reglers villkor. Om villkoret är sant skapas en order som motsvarar handlingen i regeln, och denna order skickas vidare. Annars sker ingenting och nästa regel evalueras om sådan finns.

5.6 Extrauppgifter

Extrauppgifterna som gällde att uppdatera ACP och att implementera virtuella in- och utgångar har utelämnats till förmån för att optimera grundfunktionaliteten.

6 SLUTSATSER

6.1 Bedömning av resultatet

Prototypen som har utvecklats består av olika komponenter vars individuella utveckling har beskrivits i kapitel 4 Konstruktion. Den övergripande utvecklingen har följt ett mönster av att i början av projektet ta fram mycket grundläggande funktionalitet för varje komponent och sedan sammanfoga dessa komponenter till ett system. Fokus läggs på att de olika komponenterna samverkar på samma sätt som i det slutliga systemet, trots att delarna ännu är inkompletta. När komponenterna samverkar som önskat fortsätter arbetet på varje separat komponent. Komponenten förbättras tills den uppfyller specifikationerna. När detta uppnåtts ses eventuella brister över och ineffektiva lösningar optimeras om tid återstår. Detta arbetssätt har fungerat väl under projektet eftersom det redan tidigt i processen möjliggjort utveckling av ett system innehållande tidiga versioner av samtliga komponenter. Med ett sådant system att arbeta vidare på har få problem uppstått vad gäller samverkan mellan komponenterna.

6.1.1 Datamodellen

Datamodellen innehåller allt som specificerades i projektets inledning. En mindre brist är att vissa element som kan vara användbara inte implementerades eftersom deras användbarhet upptäcktes först i slutfasen av projektet. Datamodellen utökades inte med dessa element eftersom redigering av datamodellen skulle kräva att en helt ny konfigurationsfil skapades och tiden inte fanns för detta.

Det diskuterades huruvida det skall vara möjligt att skapa flera regler med samma handling i, dvs. att en order kan ha flera orderregler. Valet gjordes att en order endast skall ha en regel. Detta därför att logikmodellen blir mer lättnavigerad och lättare att modifiera om alla regler för en viss order finns på samma plats.

Datamodellens styrka är att den tillåter mycket komplexa logikuttryck med långa sammansatta uttryck och de logiska operatorerna AND, OR, XOR och NOT. Därför kan komplexa regler skapas i en konfiguration.

DMP som användes för att exportera datamodellen till SQL-kod fungerade väl och var inget hinder för utvecklingen.

6.1.2 Konfigurationer för test

Den slutgiltiga konfigurationsfilen är tillräcklig för sin uppgift. Den räcker för att demonstrera kommunikationen mellan CTC och OCS samt att verifiera att logikreglerna är korrekta. Den saknar dock vissa element som skulle göra demonstrationen mer visuellt tydlig. Detta beror på att dessa element inte har implementerats i vare sig datamodellen eller CTC.

En slutsats som har kunnat dras under arbetets gång är att konstruktionen av konfigurationsfiler via DPT är relativt tidskrävande. Det är dock det enda verktyg som i

nuläget finns för ändamålet. Framtida projekt bör ha detta i åtanke redan i planeringsfasen så att tillräcklig tid kan avsättas för konstruktion av konfigurationer.

6.1.3 Representation av logikmodellen i C

Representationen av logikmodellen fungerar väl och genomgick under arbetets gång flera förbättringar. En mindre svaghet är att de binära träden ej är balanserade, vilket kan åtgärdas vid vidareutveckling.

Stackmaskinen som kompilerar reglers villkor till bytekod är en värdefull del av programmet eftersom den uppfyller kraven i kodstandarden.

Vissa svårigheter har uppstått på grund av att dokumentationen för CCU6 i vissa fall varit bristfällig. Detta har lett till att vid ett antal tillfällen har tid lagts på att utveckla funktionalitet som redan existerar i CCU6. Källkoden har därför behövt undersökas innan ny funktionalitet utvecklats, för att säkerställa att funktionaliteten inte redan finns. Ett exempel på detta är att tid lades på att implementera ett sätt att komma åt ett objekts information och status med hjälp av dess unika strängnamn. På grund av bristfällig dokumentation upptäcktes det inte förrän senare i arbetet att liknande funktionalitet redan existerar i CCU6. I slutversionen av prototypen används CCU6:s implementation av detta.

6.1.4 Evaluering av regler

Den virtuella stackmaskinen ingår även i evalueringen eftersom den förutom att kompilera villkor till bytekod också exekverar denna bytekod och därigenom evaluerar villkoret. Stackmaskinen fungerar som önskat.

Innan stackmaskinen skapades togs en första prototyp för evaluering fram. Denna använder sig av rekursion för att söka igenom ett villkor. När en objekttyp hittas slås referensen till objektet upp och status för objektet kontrolleras mot önskad status i villkoret. Eftersom denna lösning bryter mot fastställd kodningsstandard byttes den senare i arbetet ut mot stackmaskinen. Den var dock en relativt användbar första version då den redan tidigt i arbetet kunde användas för att undersöka att logikmodellen fungerade som önskat.

6.1.5 Skicka in- och utdata via logikmodellen

En viktig erfarenhet från arbetet med att hantera in- och utdata är att statusuppdateringar från Outputs skiljer sig från andra objekttyper. Output skickar kontinuerligt statusuppdateringar och statusförändring upptäcks därför inte automatiskt av CCU6. För att upptäcka statusförändring för Outputs behövde därför flanktriggning implementeras. För övriga objekt har CCU6:s funktionalitet använts för att vid inkommande status upptäcka om status förändrats.

CCU6:s funktionalitet för att ta emot ordrar är densamma oavsett om ordern kommer från TCC eller CTC. Därför behövde denna funktionalitet inte modifieras för detta projekt. Sändning av styrsignaler till OC behövde likaså inte modifieras. För att handlingen i

reaktionsregler skall utföras behöver dock ett nytt paket med en order skapas för att skickas vidare till OC. CCU6 har funktionalitet för detta, och denna funktionalitet användes för att skapa orderpaket.

6.1.6 Simulering

Eftersom CTC först i slutfasen av projektet kunde sända ordrar till OCS var fungerande simulatorer en nödvändighet för utvecklingen. Dokumentation för simulatorerna var något bristfällig och simuleringarna blev därför problematiska. En betydande andel arbetstid gick åt till att få de olika simulatorerna att kommunicera med varandra och fungera korrekt.

6.1.7 Extrafunktionalitet

Uppdatering av ACP och implementation av virtuella in- och utgångar utelämnades ur konstruktionen på grund av tidsbrist. Beslutet togs att det var av större prioritet att förbättra grundfunktionalitet.

Under utvecklingen av funktionaliteten för att lagra status för Output (se avsnitt 4.6.2 Reaktionsregler – Läsa av bangårdsobjektens status) insågs dock att denna lösning även kan användas för att implementera virtuella in- och utgångar. För att ta fram en komplett implementation av detta krävs dock modifieringar i datamodellen och en ny konfiguration måste definieras. På grund av att detta är tidskrävande uppgifter togs beslutet att inte färdigställa denna funktionalitet.

6.1.8 Planering och arbetssätt

I stort har utvecklingen följt den planering som lades upp i början av arbetet. De verktyg som användes har inte avvikit från de som planerades. Handledarna på företaget har under hela arbetet varit den största källan till kunskapsinhämtning. Under första halvan av projektet har dokumentation som företaget tillhandahållit använts till stor del. Under senare delar av projektet har även Internet och litteratur nyttjats.

Den kodstandard som företaget använder har följts i största möjliga mån utan betydande avvikelser.

Utvecklingen har i stort sett endast skett i form av parprogrammering så att lösningar har kunnat diskuteras kontinuerligt och fel har kunnat upptäckas tidigt i processen. Arbete med dokumentation och övriga uppgifter har varierande utförts individuellt och i grupp.

Till största utsträckning har tidsplanen följts och arbetet kunde avslutas i enlighet med planeringen. I perioder har arbetet varit mer intensivt men ingen allvarlig tidsbrist har uppstått.

6.2 Hållbar utveckling

OCS950 som detta arbete är baserat på är en del i ett befintligt system som används för tåg i gruvverksamheter och persontransport. Avsikten med det framtagna systemet i detta arbete är

att ge automatiserad kontroll till sträckor av järnvägar som idag styrs manuellt. Detta skall öka säkerheten genom att förebygga misstag som kan begås vid manuell styrning. Genom att kunna reglera styrobjekt från en applikation ökar säkerheten för operatörer som kan undvika att beträda banområden.

Tågtrafik har lägre koldioxidutsläpp och högre säkerhet än bil, buss, och flyg och kan anses vara ett mer hållbarart transportmedel för person- och godstransport [11]. I takt med att hållbar utveckling blir en allt mer prioriterad aspekt vid samhällsutveckling, satsas mycket på att utöka spårtrafiken och minska annan trafik [10]. Automatiska säkerhetssystem är en mycket viktig del av att säkerställa att denna utökade spårtrafik fungerar på ett säkert sätt.

6.3 Vidareutveckling

Vid vidareutveckling av programmet skulle ACP kunna utökas med tester för koden som skrivits under projektet.

Vad gäller programkoden kan vissa optimeringar genomföras som vidareutveckling. Ett exempel är att implementera balansering av de binära träden. En annan förbättring är att frigöra minne som allokeras vid inläsning för tillfälliga datastrukturer. Ett prestandatest kan också genomföras för att åstadkomma optimering.

Fler typer av objekt skulle även kunna inkluderas i datamodellen för att kunna ingå i konfigurationer. En objekttyp som vore speciellt användbar är virtuella in- och utgångar. Dessa skulle kunna användas för att skapa tillståndsmaskiner för objektlägen och därigenom framställa konfigurationer med mer avancerad funktionalitet.

REFERENSER

[1] Hedberg, K., Elestedt F., *Safety-critical Communication Controllers for Railway Signalling in Public Networks.* Master of Science Thesis in Computer Science and Engineering, Chalmers Tekniska Högskola, 2008.

[2] *About us*. Bombardier Transportation. http://www.bombardier.com/en/about-us.html (Hämtad 2016-05-28).

[3] *Visual Studio Code*. Microsoft. https://code.visualstudio.com/ (Hämtad 2016-05-28).

[4] *Gerrit*. https://www.gerritcodereview.com/ (Hämtad 2016-05-28).

[5] *LibXML Home Page*. http://xmlsoft.org/ (Hämtad 2016-05-28).

[6] *Järnväg.net*. http://www.jarnvag.net/ (Hämtad 2016-05-28).

[7] *Projekt.* Bombardier i Sverige. http://se.bombardier.com/se/bt_projects.htm (Hämtad 2016-05-28).

[8] *Göteborg, Sverige.* Bombardier i Sverige. http://se.bombardier.com/se/site_details_gothenburg.htm (Hämtad 2016-05-28).

[9] *XML 1.0 Specification.* World Wide Web Consortium. https://www.w3.org/TR/REC-xml/ (Hämtad 2016-05-29).

[10] *IPCC Fourth Assessment Report: Mitigation of Climate Change, chapter 5, Transport and its Infrastructure.* Intergovernmental Panel on Climate Change. http://www.ipcc.ch/pdf/assessment-report/ar4/wg3/ar4-wg3-chapter5.pdf (Hämtad 2016-06-04).

[11] *Improving the sustainability of transport – The rail sector as a case study.* Henning Schwarz. https://sustainabledevelopment.un.org/content/dsd/csd/csd_pdfs/csd-19/learningcentre/presentations/May%2010%20pm/1-%20Henning%20Schwarz_CSD19%20Learning%20Centre_UIC_110510_update.pdf (Hämtad 2016-06-07).

BILAGOR

Bilaga A. Initial tidsplan

Tidsplan som sammanställdes tillsammans med handledarna under projektets uppstart.

v 12: Uppstart. Användarkonton, installera program, undersöka utvecklingsmiljöer, få tillgång till Gerrit etc.
v 13: Planering och fortsättning av uppstarten. Börja på XML-syntax till datamodellen.
v 14: Fortsätta på XML-syntax till datamodellen.
v 15: Börja skapa logikblock. Börja med grunden på rapporten.
v 16: Fortsätta arbeta på logikblocken. Börja skapa konfigurationer för test. Fortsätta med grunden på rapporten.
v 17: Fortsätta med konfigurationer för test.
v 18: Fortsätta med konfigurationer för test. Börja skapa representation av logikmodellen i C.
v 19: Fortsätta med representation av logikmodellen i C.
v 20: Fortsätta med funktionalitet för att skicka in- och utdata genom logikmodellen. Arbeta på återstående delar av rapporten.
v 21: Fortsätta på arbete som återstår. Eventuellt implementera extra funktionalitet. Fortsätta med rapportskrivningen.
v 22: Rapportskrivning. Avsluta eventuellt återstående arbete. Förbereda presentation och opposition.
v 23: Rapportskrivning. Förbereda presentation och opposition.