Simulink som modelleringsverktyg för pålitliga program

Andreas Engblom, 35697
Kandidatavhandling i datavetenskap
Handledare: Marina Waldén och Jonatan Wiik
Fakulteten för naturvetenskaper och teknik
Åbo Akademi
2018

Referat

Då systemen blir större och mer avancerade blir det också extra viktigt att kraven följs så att systemen fungerar på rätt sätt och är pålitliga. Det här är något som har speciellt stor betydelse på säkerhetskritiska system. Att garantera det här för komplexa system är inte en lätt sak, och därför finns det flera olika grafiska verktyg som kan användas under utvecklingen av systemet för att hjälpa till att göra resultatet pålitligt. Den här avhandlingen kommer att fokusera på verktyget Simulink, hur man kan modellera system och använda de olika verifieringsmöjligheterna som verktyget har för att undersöka om kraven följs.

Nyckelord: Simulink, Simulink Design Verifier, Grafiska användargränssnitt, Säkerhetskritiska system, Modellbaserad utveckling

Innehållsförteckning

1. Inledning

Det har blivit allt vanligare att utveckla säkerhetskritiska system i form av mjukvara. När man utvecklar sådana system är det mycket viktigt att alla krav följs noggrant så att systemen fungerar på rätt sätt. Om något går fel i säkerhetskritiska system kan följderna bli katastrofala. Att garantera att systemen fungerar som det ska, blir allt svårare i och med att systemens komplexitet ökar. För att hjälpa till med utvecklingen av komplexa system har det blivit allt vanligare att använda en metod som kallas modellbaserad utveckling. Modellbaserad utveckling är en metod som möjliggör snabbare och ett kostnadseffektivare sätt att utveckla system. Det finns flera olika verktyg som man kan använda sig av för att utveckla system med hjälp av modellbaserad utveckling. Simulink är verktyget som kommer att fokuseras på i den här avhandlingen.

Den här avhandlingen kommer att ta upp grafiska användargränssnitt i allmänhet och på vilket sätt man kan ha fördelar av dem bland annat inom programmering, jämfört med att alltid behöva använda sig av kommandoradsgränssnitt. Säkerhetskritiska system och vad man speciellt behöver tänka på när man utvecklar sådana kommer också tas upp. I kapitel 3 behandlas modellbaserad utveckling (eng. model-based design[12]). Modellbaserad utveckling behandlas som kärnan bakom utvecklingsverktyg som Simulink och vad som är bra med användningen av modellbaserad utveckling. I det kapitlet kommer det också att tas upp exempel på andra verktyg förutom Simulink som man kan använda sig av vid modellbaserad utveckling. Den här avhandlingen kommer att begränsas till att beskriva ett av de antal verktyg som kan användas inom modellbaserad utveckling. Verktyget som avhandlingen kommer att fokusera på är Simulink. I kapitel 4 beskrivs hurdant Simulink är samt teknisk information om hur det fungerar och vilka olika verifieringsmöjligheter som finns i verktyget. Till slut beskrivs hur Simulink kan användas för att modellera mer kritiska system.

2. Bakgrund

I det här kapitlet kommer avhandlingen att ta upp allmän information om grafiska användargränssnitt och information angående säkerhetskritiska system så att man får en förståelse om vad det innebär. Den här typen av bakgrundsinformation förklarar varför begreppet modellbaserad utveckling finns, något som kommer att beskrivas senare i avhandlingen.

2.1. Allmänt om grafiska användargränssnitt

Ett grafiskt användargränssnitt (eng. graphical user interface, GUI) är något som enligt Chris Adams [1] nuförtiden används av de flesta operativsystem och mjukvaruprogram. Det som kännetecknar system som använder grafiska användargränssnitt är att de använder sig av ikoner, fönster, olika menyer och andra visuella element som syns på skärmen. Med de här kan användaren sedan interagera med hjälp av att klicka med en mus eller till och med peka på skärmen med fingret ifall den typen av teknik stöds.

Grafiska användargränssnitt har inte alltid funnits utan enligt *Encyclopedia of Small Business* [7] var det på 1970-talet som företaget Xerox utvecklade det första grafiska användargränssnittet, men först år 1984 blev det populärt att använda genom datorn Apple Macintosh [10]. Före det fanns grafiska användargränssnitt var så kallade textbaserade gränssnitt (eng. command line interface ,CLI) vanliga, och de existerar fortfarande. System som använder textbaserade gränssnitt stöder endast tangentbord vilket betyder att allt som ska göras på sådana system kräver att användaren med tangentbordet skriver in rader med unika instruktioner.

Fördelar med att använda grafiska användargränssnitt jämfört med textbaserade är att de systemen är mycket lättare att lära sig att använda, även för användare som kanske inte har så stor erfarenhet. En orsak till varför det är lättare är att användare inte behöver komma ihåg alla olika datorinstruktioner, och vilken instruktion som gör vad. Fördelar för grafiska användargränssnitt kan också ses ur programmerarens perspektiv. Att programmera i en integrerad utvecklingsmiljö med grafiskt gränssnitt kan underlätta p.g.a. att programmeraren då har allt som behövs i samma program. Integrerad utvecklingsmiljö är ett datorprogram som innehåller olika

redskap för att underlätta vid programmering. Programmet har olika menyer för att man snabbt ska kunna navigera sig fram och hitta det man behöver använda sig av och kunna köra programmet man skapat med ett enda klick.

I artikeln *When Less is More: Meaningful Learning from Visual and Verbal Summaries of Science Textbook Lessons* [15] från 1996 beskriver Richard E. Mayer m.fl. tester som gjorts med deltagare som inte hade så stor erfarenhet av meteorologi. Idén med testerna var hur bra deltagarna kunde minnas viktiga saker från uppgiften de skulle göra. Deltagarna delades in i grupper och alla grupper skulle göra samma uppgift men fick olika material. Slutsatsen som Richard E. Mayer m.fl. kom fram till med de här testerna var att användning av bilder har stor inverkan på inlärningen, och att det effektivaste sättet visade sig vara då det fanns en kort text tillsammans med bilden. Enbart text, enbart bilder eller för mycket text tillsammans med bilder visade sig inte alls vara lika effektiva sätt att lära sig något och minnas det.

Ett enkelt exempel på hur de två gränssnitten fungerar är sättet man öppnar en mapp i datorn för att visa innehållet. Är det ett operativsystem som använder grafiskt användargränssnitt klickar man på ikonen som representerar mappen och ser därefter vilka filer som finns i mappen, men är det ett textbaserat gränssnitt krävs det att instruktionen ”ls” skrivs in och därefter sökvägen till mappen som ska öppnas. Till exempel ”ls /home/user/documents” ger en lista på alla filer som finns i mappen documents i alfabetisk ordning.

2.2. Säkerhetskritiska system

I artikeln *Safety Critical Systems: Challenges and Directions* [8] förklarar John C. Knight termen säkerhetskritiska system som att, ett system är säkerhetskritiskt om ett fel i systemet kunde leda till oacceptabla konsekvenser. Sådana konsekvenser kan vara att någon skadas eller till och med mister livet, allvarligare byggnadsskador eller också skador på omgivningen.

Att utveckla säkerhetskritiska system är inte en lätt uppgift att göra. Det finns många saker som kan gå fel och i värsta fall leda till katastrof, därför krävs det noggrann planering så allt går rätt till under utvecklingens gång.

Artikeln *The Ethics of Safety-Critical Systems (2000)* [4] av Jonathan Bowen tar upp bl.a. sju viktiga punkter i form av råd som kan hjälpa till att undvika att det sker fel i systemet som utvecklas.

(Råd 1) Före en viss typ av metod väljs för ett projekt ska det funderas på vilket sätt det kommer att hjälpa och om det verkligen är nödvändigt att välja den tekniken. En viss metod att gå till väga ska inte väljas utan bra orsak till varför. (Råd 2) En metod ska aldrig genast ses som den enda möjliga lösningen. Det lönar sig alltid att fundera noggrant ifall det finns andra möjligheter som t.o.m. kunde vara bättre att välja.

(Råd 3) Det ska aldrig tas för givet att svaret som fås av automatiserade verktyg är sant. Testning måste alltid göras även om bevis har gjorts antingen för hand eller med något verktyg p.g.a. det kan aldrig garanteras att inget fel har gjorts. (Råd 4) Om en ny metod ska sättas in utvecklingsprocessen så ska det försöka göras på ett kostnadseffektivt sätt. Det är viktigt att det finns möjlighet till någon form av utbildning av den nya metoden genast i början så att det blir klart hur den nya metoden används.

(Råd 5) Återanvändning av programkod i något annat system att är något som kräver extra försiktighet. Noggrann testning krävs för att undvika katastrof. Programkoden behöver nödvändigtvis inte fungera på rätt sätt i nya systemet fast det fungerat felfritt tidigare. (Råd 6) Dokumentation med passlig mängd information utan onödiga detaljer ska finnas med. Informationen som finns med ska vara skriven så att även otekniska personer förstår den. (Råd 7) Slutligen är det bra att så tidigt som möjligt försöka upptäcka och lösa problem. Ofta kan det hjälpa att göra en formell specifikation av systemet som ska utvecklas för att upptäcka problemen.

3. Modellbaserad utveckling

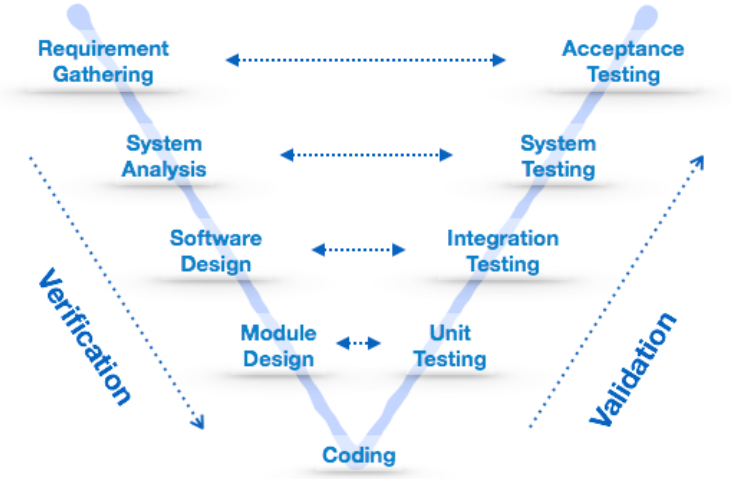
Modellbaserad utveckling (eng. model-based design [12]) är en utvecklingsmetod där man som utvecklare använder sig av så kallad grafisk programmering. Grafisk programmering betyder i det här fallet att en grafisk modell skapas som sedan är i fokus under hela processens gång. Modellen som skapas beskriver på ett grafiskt sätt det fysiska system som utvecklas. Modellen innehåller samma kravspecifikationer som krävs av det fysiska systemet. Om modellen är bra gjord betyder det här att det inte borde vara komplicerat för andra, som inte arbetat med modellen att förstå vad systemet som modellen motsvarar gör.

Orsak till att modellbaserad utveckling används är för att det möjliggör utveckling av system som t.ex. signalbehandlingssystem, kontrollsystem och kommunikationssystem på ett snabbare och kostnadseffektivare sätt [12]. En annan fördel är att modellen som skapas är exekverbar vilket betyder att modellen går att simulera. Genom att simulera modellen ofta så går det att upptäcka fel i ett tidigt skede och rätta till dem [12].

Då modellen är klar och alla implementationskrav har blivit inkluderade i modellen, är det tid för slutlig verifiering av systemet. Modellbaserad utveckling gör det nu möjligt att automatiskt generera programkod direkt från modellen, och skapa återanvändbara tester för att kunna verifiera att allt fungerar korrekt [12].

När ett system ska utvecklas med hjälp av modeller är det viktigt att rätt domänspecifikt modelleringsspråk väljs. Domänspecifikt språk är ett programmeringsspråk som finns för en specifik uppgift. Det finns många olika domänspecifika språk och var och en av dem är lämpliga för olika områden. Det är därför viktigt att ta reda på vilken typ av system som ska utvecklas och att modelleringsspråket som väljs åtminstone stöder de viktigaste verktygen. Ett specifikt område där flera olika grafiska modelleringsspråk har något gemensamt är området kontrollsystem utveckling. Några exempel på sådana grafiska modelleringsspråk är Simulink [12], SCADE [6] och Xcos [16]. Samtidigt som de heter domänspecifika språk är de också modelleringsverktyg med samma namn. Det som är speciellt med de här språken är att de är integrerade i utvecklingsmiljöer med stort stöd för olika verktyg. Simulink [12] har Matlab, SCADE [6] har SCADE Suite och Xcos [16] har Scilab som utvecklingsmiljö. Scilab med Xcos [16] är

öppen mjukvara variant av Matlab med Simulink [12]. Likheter finns mellan dem men innehållet fungerar ändå inte fullständigt med varandra. SCADE verktyget baserar sig på programmeringsspråket Lustre som ligger bakom utvecklingen av flera säkerhetskritiska realtidssystem [6]. Lustre är ett synkront språk som har en formell semantik och som stöds av flera verktyg, för simulering, testning och modellkontrollering [9].



Figur 1: V-modell processen som används vid modellbaserad utveckling. (Hämtat från [18])

I figur 1 visas en v-modell vilket är en typ av utvecklingsprocess som används vid modellbaserad utveckling. Pilarna visar i vilken ordning som man går till väga. I den här v-modell processen går verifieringsdelen och valideringsdelen parallellt med varandra. I varje stadie av processen görs testfall specifikt för stadiet för att verifiera att det stämmer överens med specifikationen. Processen börjar med att man samlar in kraven för systemet som ska utvecklas. Analysering av systemet sker sedan före modellering av systemet påbörjas. Efter att modelleringen är klar genereras programkod från modellen. Efter att systemet är utvecklad går man vidare till testning. Vid varje stadie på valideringsdelen använder man testfallen som skapades tidigare på verifieringsdelen. Testfallen används för att validera systemet faktiskt gör vad det ska .

4. Simulink som verktyg

Det här kapitlet kommer att delas in i tre delar där Simulink ligger i fokus. Det förklarar vad Matlab är och vad det har för betydelse för Simulink, och beskriver Simulink som ett modelleringsverktyg. Det tar även upp verktyget Stateflow och hur det kan kombineras med Simulink för att skapa modeller.

4.1. Matlab

Matlab som är en förkortning av Matrix Laboratory är en produkt utvecklad av MathWorks. Matlab är en teknisk databehandlingsmiljö för olika områden, men också ett högnivå programmeringsspråk. Matlab är en teknisk databehandlingsmiljö för matris manipulering, algoritm utveckling, dataanalys, signalbehandling och trådlös kommunikation, bild och videobehandling, system modellering och simulering, beräkningsfinansiering, robotteknik, bioinformatik och djup maskininlärning [11].

Matlab har stöd för ett stort antal verktygslådor (eng. toolboxes) som går att installera för att utvidga funktionerna i Matlab. Det går t.ex. inte att göra bildbehandling i Matlab före bildbehandling verktygslådan har installerats. Simulink och Stateflow är två tilläggsprodukter som går att lägga till i Matlab för att tillsammans göra modellbaserad utveckling. Både Simulink och Stateflow kräver att Matlab körs för att kunna användas. Eftersom de är tilläggsprodukter under Matlab så kan de använda sig av varandra när det behövs.

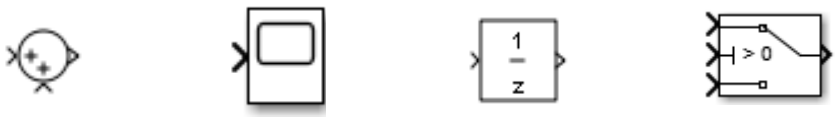
4.2. Simulink

Simulink [12] är ett populärt grafiskt modelleringsverktyg som används inom modellbaserad utveckling och är utvecklat av företaget MathWorks. Ett område inom vilket Simulink [12] är ett av de populäraste verktygen att använda, är utvecklingen av olika inbyggda kontrollsystem i form av mjukvara. Exempel på vad sådana kontrollsystem kan göra är t.ex. kontrollering av krockkuddar, låsningsfria bromsar, medicinsk utrustning och luftfarkosters flygstyrningssystem [8],[19]. Designen som används då man modellerar i Simulink baserar sig på

dataflödesdiagram. Dataflödesdiagram betyder att man visuellt med hjälp av olika symboler och pilar visar hur information rör sig mellan processer i ett system.

Modellerna som man bygger upp i Simulink består av många olika funktionella block. Det finns skilt block för varje funktion. Exempel på några block som finns är summablock, logiska operatorblock, signalruttningsblock, ingångsport- och utgångsport-block och minnesblock som kan lagra värden. Simulink har ett så kallat bibliotek fullt av olika block som man kan använda sig av. Blocken är ordnade enligt olika kategorier för att underlätta sökandet ifall man inte kan namnet på blocket. Biblioteket ger också tillgång till block som hör ihop med installerade verktygslådor. När man hittat blocket man sökt efter drar man och släpper ner blocket i fönstret där man börjat modellera. Om man vet namnet på blocket kan man direkt skriva ner namnet på en sök rad i modelleringsfönstret. Förslag på block namn ges därefter enligt det man skrivit.

De olika funktionella blocken har ingångsportar och utgångsportar till vilka de kopplas ihop med varandra med dataflödessignaler. På ingångsportarna kommer input som beräknas inne i blocket och ges sedan ut som en output på utgångsporten. Många av Simulink-blocken har block parametrar som går att justera på för att ändra hur blocket ska bete sig. Inställningar som kan finnas tillgängliga att ändra på i blocket är bl.a. parameter egenskaper, signal egenskaper, datatyper och tillstånd. Vad som finns tillgängligt beror på vilken typ av block det är.



a) b) c) d)

Figur 2: Exempel på olika Simulink-block. a) Sum-block, b) Scope-block, c) Unit delay-block, d) Switch-block. Modellering med de här blocken visas i kapitel 4.2.1.
(Bilderna från Simulink dokumentationen [12])

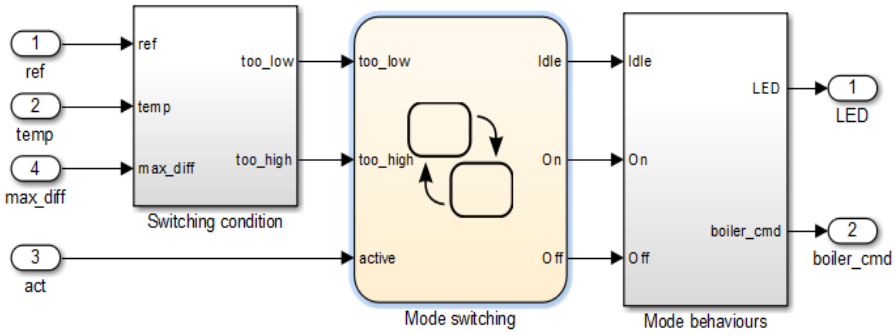
Figur 2 ger exempel på fyra olika funktionella Simulink-block som dessutom är från olika kategorier. Block a) Sum-block är ett block som adderar det som kommer som input. Block parametrarna som finns för blocket är olika signal- egenskaper

och möjligheten att välja mellan rund symbol som i a) eller en rektangulär form. Sum-blocket kan även användas för att subtrahera genom att i inställningarna ange tecken på ingångsportarna. Block b) Scope är ett block som visar signaler i form av grafer som genereras under simuleringen av en Simulink-modell. Block c) Unit delay- block hör till kategorin diskret-tids block och är ett minnesblock. Unit delay-blocket accepterar en input och fördröjer den med en samplingstid före en output ges ut. Samplingstiden anges i blockets parametrar där det också finns möjlighet att ändra på tillstånd egenskaper. Block d) Switch-block är ett signalruttningsblock där signalen passerar genom ingångsport 1 eller ingångsport 3, beroende på om input 2 stämmer överens med givet villkor. I blockets parameter inställningar anges villkoret och vid behov finns möjligheten att ändra på signal egenskaperna. Om input 2 stämmer överens med villkoret passerar signalen genom ingångsport 1 och annars genom ingångsport 3. Numreringen går uppifrån ner.

Simulink-modeller kan snabbt bli väldigt stora då avancerade system modelleras. Det här beror på att antalet Simulink-block som krävs kan bli väldigt stort så systemet som modelleras har många funktioner. Modellerna kan då bli mer invecklade att förstå. Simulink ger nu möjligheten att skapa hierarkiska modeller genom gruppering med så kallade subsystem-block. Subsystem-block läggs in i Simulink-modellen och skapar på det sättet nivåer i modellen. In i de här subsystem-blocken grupperas sedan en del av systemets block. Subsystem-blocken kopplas därefter på samma sätt ihop med dataflödessignaler. Användning av subsystem-block gör att Simulink-modellen får en hierarkisk struktur och gör den lättare att förstå. Det finns olika typs subsystem-block att välja mellan beroende på hur man vill att de ska exekveras. Virtuella subsystem-block används för att gruppera block utan annan större betydelse. Atomära subsystem-block betyder däremot att blocket exekveras som ett enda block.

Simulink har ett stort antal funktionella block att välja mellan, men om någon funktionalitet saknas kan egna block skapas. Simulink inbyggt i Matlab gör det möjligt att också definiera egna block med hjälp Matlab, C, C++ och Fortran programkod [12].

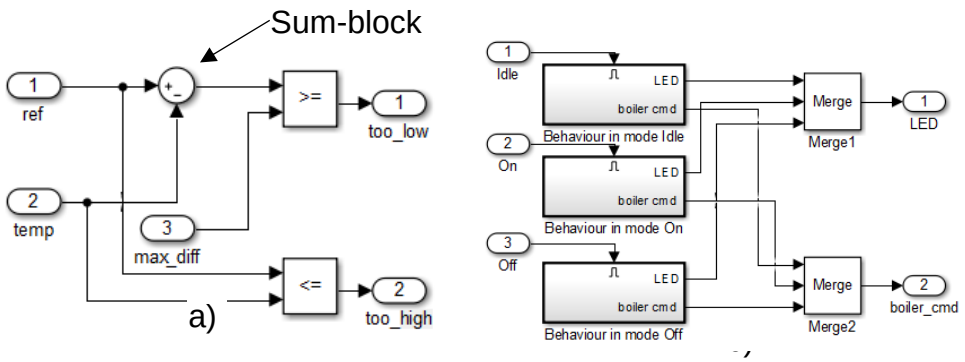
4.2.1. Exempel på Simulink-modeller



Figur 3: Exempel på Simulink-modell som kombinerar subsystem-block med ett Stateflow-block. Modellen visar ett kontrollsystem för en värmepanna. (Exemplet från [3])

Figur 3 visar en Simulink-modell på en värmepanna för att värma upp vatten. Värmaren kan antingen vara på eller av. Det finns tre olika lägen som systemet kan vara i. Två olika aktiva lägen där vattentemperaturen kontrolleras dvs. aktivt läge där värmaren är på och aktivt läge där värmaren är av, och så finns ett viloläge där ingen kontroll sker alls. Den del av systemet som sköter om kontrollen av vattentemperaturen dvs. reglerenheten får fyra stycken input. Variabeln *ref* ger referenstemperatur, *temp* ger nuvarande temperaturen, *max_diff* ger största möjliga avvikelse från referenstemperaturen och variabeln *act* aktiverar reglerenheten. Reglerenhetens uppgift är att sätta på värmaren vid för låg temperatur och stänga av då referenstemperaturen har nåtts. Ingen kontroll sker över värmaren ifall reglerenheten aldrig har blivit aktiverad.

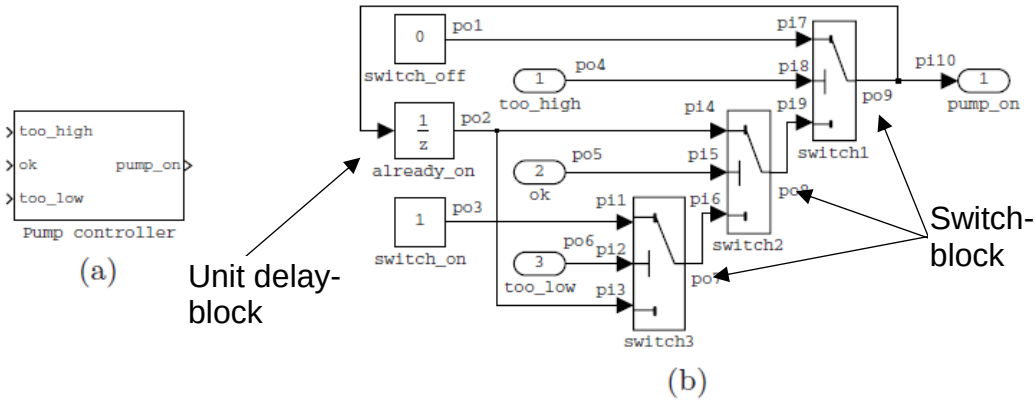
Systemet ger ut två stycken output, en *LED*-lampa på ett utgångsportblock och *boiler_cmd* på ett andra utgångsportblock. *LED*-lampan berättar i vilket av de tre lägena som systemet är i genom att ge olika värden dvs. 0, 1 eller 2. Värdet 0 för att värmaren är av, värdet 1 för viloläge och värdet 2 för att värmaren är på. Vid aktivering och avaktivering av systemet kan det ske byte av läge. Byte sker också då temperaturgränserna *ref* och *ref-max_diff* har nåtts. Den andra outputen *boiler_cmd* har värdet 1 då värmaren borde sättas på och värdet 0 då den ska stängas av.



Figur 4: Innehållet av subsystem-blocken i figur 3. a) Switching condition och b) Mode behaviours. (Exemplet från [3])

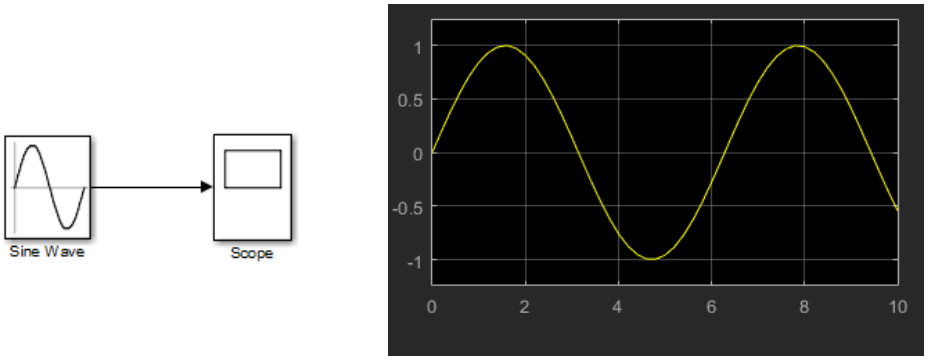
Både Switching condition-blocket och Mode behaviour-blocket I figur 3 är virtuella subsystem-block.Subsystem a) Switching condition beräknar de två tillstånden *too_low* och *too_high* som ger ut svaret sant eller falskt på utgångsportarna. Svaret används sedan som input av Mode switching-blocket i figur 3. För att visa användningen av Sum-blocket får utgångsport *too_low* värdet sant då $(ref\text{-}temp) \geq max\_diff$. Det betyder med ord att *too_low* får värdet sant då vatten temperaturen är lägre än vad den får vara. Utgångsport *too_high* ger sant då referenstemperaturen har nåtts dvs. då $ref \leq temp$.

Subsystem b) har tre så kallade enabled subsystem-block och två merge-block. De här subsystemen är atomära och har en aktiveringsport. Det finns ett subsystem för varje tillstånd som systemet kan vara i dvs. av, på och i vila. Systemet kan vara i ett tillstånd samtidigt vilket betyder att endast ett av subsystemen kan vara aktivt samtidigt. Det subsystem som är aktivt har sant som värde på aktiveringsporten. I vilket läge som systemet är i bestäms av inputen från Mode switching-blocket i figur 3. För att garantera att endast ett subsystem är aktivt så används merge-block. Merge-blocken sammanslår resultaten från subsystemen för att sedan ger ut ett svar.



Figur 5: Reglerenheten hos en ångpanna. a) subsystem-block och b) innehållet av subsystemet i a). (Hämtat från [3]).

Simulink-modellen i figur 5 används för att visa hur Switch-blocket och Unit delay-blocket används. Reglerenheten kontrollerar om en pump är på eller inte. I modellen finns två Constant-block som anger pumpens tillstånd. Det finns tre ingångsportblock som anger vattennivån dvs. för låg, för hög och ok nivå. Switch-blocken stänger av pumpen vid för hög vattennivå och aktiverar pumpen vid för låg vattennivå. Om vattennivån är ok sparas pumpens tillstånd i Unit delay-blockets minne.



Figur 6: Ett enkelt exempel för att visa hur Scope-blocket fungerar.

Figur 6 visar ett enkelt exempel där Sine Wave-blocket har kopplats till ett Scope-block. Scope-blocket visar sinusvågen som genereras av Sine Wave-blocket. Block parametrar som kan ändras för Sine Wave-blocket är bl.a. amplitud, frekvens och fas men i exemplet är parametrarna oförändrade.

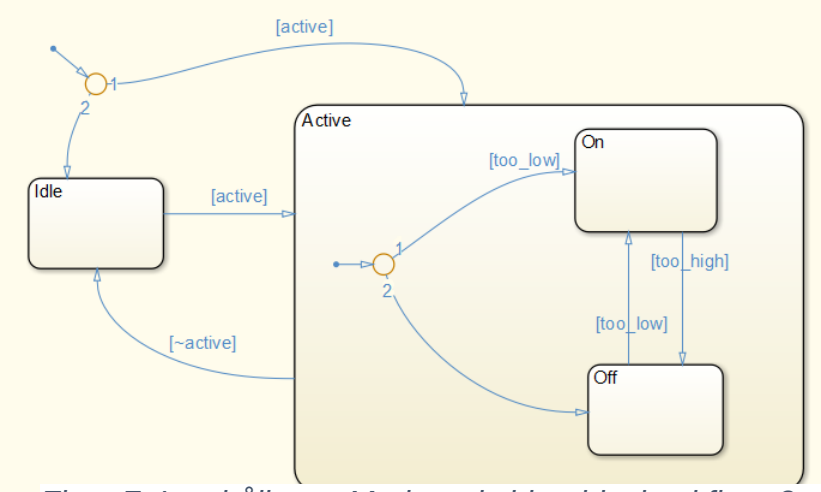
4.3. Stateflow

Stateflow [14] är en annan produkt som är utvecklad av MathWorks som fungerar som en del av Simulink. Stateflow har bl.a. använts inom flygindustrin och transportindustrin för att simulera inbyggda kontrollsystem [5]. Stateflow är ett grafiskt verktyg för modellering och simulering av komplex logik med en design som grundar sig på tillståndsmaskiner och flödesdiagram [14]. Stateflow kan användas för att modellera kontroll logik som tillståndsdiagram, t.ex. felhantering och schemaläggning av uppgifter.

För att skapa tillståndsdiagram i Simulink använder man sig av blocket Stateflow chart under kategorin Stateflow i Simulinks bibliotek. I samma kategori finns även olika tabell-block som t.ex. ger en sanningstabell och tabell som visar övergång av tillstånd. Bas delarna som används i ett Stateflow-diagram är tillstånd (state), övergångar (eng. transitions) och korsningar (eng. junctions)[5]. Tillstånd är olika lägen som ett system kan vara i. Övergång är t.ex. byte mellan två tillstånd och anges med pilar där övergångar är möjliga. Korsningar används då det finns alternativa tillstånd som systemet kan övergå till. Stateflow använder cirklar för att beteckna korsningar. Stateflow-blocken har på samma sätt som Simulink-blocken, ingångsportar och utgångsportar. Stateflow-block kan därför kopplas ihop med varandra men också kopplas med Simulink-block.

I Stateflow är det möjligt att skapa både hierarkiska och parallella tillståndsdiagram [3],[14]. Hierarkiska tillståndsdiagram i Stateflow betyder att det finns ett översta tillstånd dvs. rot tillstånd och sedan lövtillstånd som ligger under rottillståndet. Tillstånd och övergångar som ligger högre upp i hierarkin exekveras också före de övriga. Tillstånd och övergångar kan också ordnas på ett parallellt sätt men de kommer ändå aldrig att exekveras parallellt. Parallell exekvering stöds inte av Stateflow och därför kommer exekveringen alltid att ske enligt ordningsföljd. Ordningsföljden kan bestämmas på två olika sätt. Sätt ett är att Stateflow väljer att exekvera enligt hur tillstånden och övergångarna är placerade i diagrammet. Ordningsföljden går uppifrån neråt och från vänster till höger [3]. I sätt två väljer användaren ordningsföljden. När övergångar placeras ut mellan en korsning och tillstånd, sätts en siffra ut på övergången. Första övergången som placeras får siffran ett och den andra siffran två osv. och det bestämmer sedan ordningsföljden.

Övergångarna behandlas i den ordning som de placeras ut. På övergångar kan också så kallade vakter placeras. Vakterna säger när en viss övergång ska väljas och skrivs innanför en hakparentes ”[ ]”. Ifall det finns två eller flera övergångar väljs de i ordning, men om t.ex. vakten på övergång ett ger falskt så väljs övergång två istället.



Figur 7: Innehållet av Mode switching-blocket i figur 3. (Exemplet från [3])

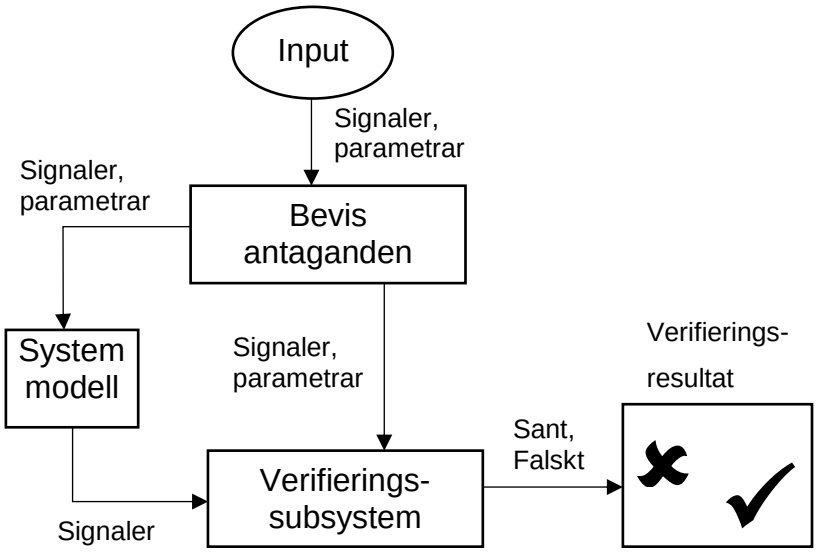
Figur 7 visar innehållet av Stateflow-blocket i figur 3. Stateflow-blocket används här för att hantera byte av tillstånd i systemet. I tillståndsdiagrammet finns två lägen, viloläge och aktivt läge. Det finns två aktiva lägen. Aktivt läge med värmaren på och aktivt läge med värmaren av. Korsning används innanför aktiva tillståndet med övergångar till tillstånden av och på. Om temperaturen är för låg används övergång ett till tillstånd på och annars övergång två till av. Övergångar med vakter finns också mellan tillstånden av och på för att stänga av och sätta på värmaren. Utanför aktiva läget finns ett viloläge och en korsning. Om reglerenheten har aktiverats används övergång ett till aktivt läge och annars övergång två till viloläge från vilket ett byte till aktivt läge kan ske senare.

5. Modellering av kritiska system med Simulink

Modellering av kritiska system är en flera stegs process som kräver noggrannhet. Människoliv kan vara beroende av att systemet fungerar på rätt sätt. Det är mycket viktigt att det finns bra verifieringsverktyg som går att lita för att underlätta med verifiering och testning av modeller. Inget verktyg ska litas på fullständigt men verktygen kan ändå hjälpa till att hitta fel i designen. Om fel upptäcks då systemet ännu håller på att utvecklas är de dessutom billigare att rätta till.

Simulink Design Verifier (SLDV) är ett verifieringsverktyg av Mathworks som fungerar i utvecklingsmiljön tillsammans med Matlab, Simulink och Stateflow. Simulink Design Verifier har på samma sätt som de övriga tilläggen, en verktygslåda med egna block. Simulink Design Verifier gör det möjligt att upptäcka olika typ av designfel i modellen, generera automatiska testfall, modellera krav som egenskaper med funktionella block, verifiera egenskaper i modellen och analysera och spåra beroenden i modellen. När modellen har verifierats markeras blocken med färgerna röd, grön och gul. Röd betyder att det finns fel i blocket, grön betyder att blocket har verifierats framgångsrikt och gul betyder att det ännu är oklart.

Tekniken bakom bevisen som görs av Simulink Design Verifier grundar sig formella metoder. Formella metoder är det som annars görs för hand på sidan om de automatiserade testerna, för att försäkra sig att allt blev rätt. Nu behövs det inte lika mycket testning för hand. Före någon form av verifiering kan ske görs ett kompabilitetstest av Simulink Design Verifier för att kontrollera att blocken som används stöds. Alla block stöds inte av Simulink Design Verifier. Olika typ av design fel som kan upptäckas med Simulink Design Verifier är bl.a. heltalsöversvämning (eng. integer overflow), division med noll, död logik och brott mot designkrav och påståenden [13]. Död logik är ett fel där delar av modellen förblir inaktiv under exekveringen, vilket kommer att leda till så kallad död kod då kodgenerering görs. För varje fel som hittas skapas ett testfall som används för att återskapa felen och åtgärda dem.

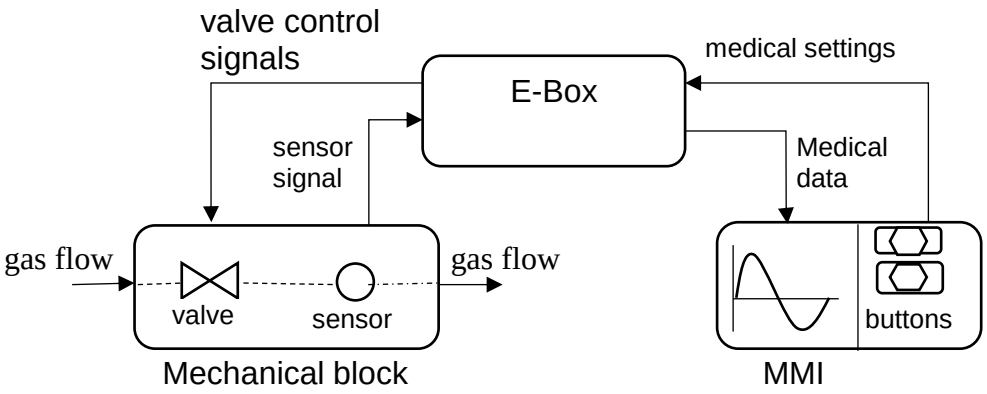


Figur 8: Diagrammet visar hur bevis av egenskaper går till med Simulink Design Verifier [17].

Figur 8 visar stegvis hur det går till då Simulink Design Verifier används för att bevisa egenskaper i modellen. Simulink Design Verifier kan bevisa egenskaper på hela modellen eller också bara på ett atomärt subsystem i modellen. Beviset börjar med att det bestäms vilken typ av input som ska gå till systemet. På de här inputvärdena görs sedan antaganden i ett antagandesubsystem. De här antagandena används sedan av både system modellen och verifieringssubsystemet. Verifieringssubsystemet använder dessutom den output som kommer från system modellen. Här i verifieringssubsystemet använder Simulink Design Verifier en teknik som grundar sig på formella metoder, för att upptäcka brott mot egenskaper. Ett resultat ges sedan utifrån det som säger vilka egenskaper som är godkända och vilka som är inte är det. Om något brott mot egenskaper hittas ger Simulink Design Verifier ett motexempel som svar.

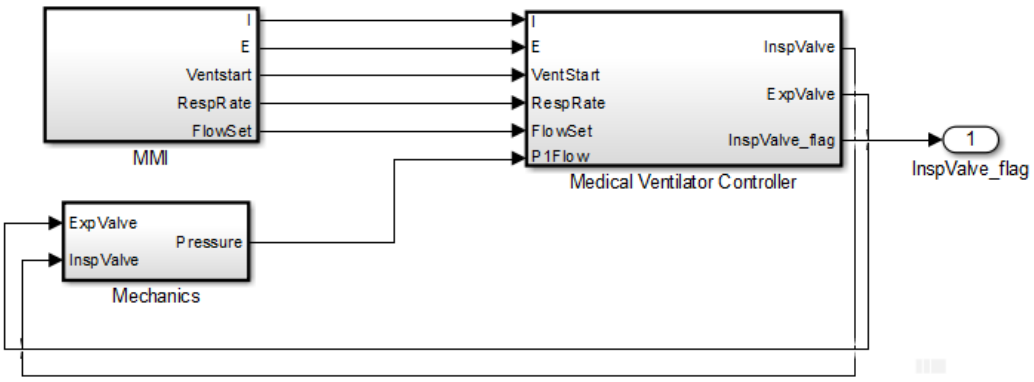
5.1. Exempel

Här kommer nu ett exempel för att visa hur man kan modellera ett kritiskt system med Simulink och sedan använda Simulink Design Verifier. Systemet som tas upp är en respirator, vilket är en medicinsk utrustning för att hjälpa människor som har svårt med andning.

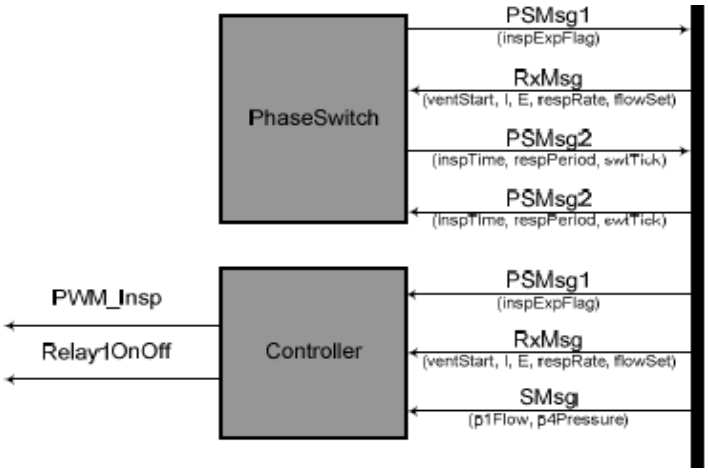


Figur 9: Diagram respirator systemet. Visar hur subsystemen påverkar varandra [20].

Första steget som man borde göra när ett system ska modelleras är att ta reda på allt om hur systemet ska fungera. Samtidigt ska alla krav samlas, eftersom det är ett kritiskt system är det mycket viktigt att kraven följs. Det kan vara bra att därefter göra en förenklad bild hur systemet fungerar enligt den givna informationen. Figur 9 visar de tre viktigast delarna i respirator systemet och hur de påverkar varandra dvs. subsystemen Mechanical block, MMI och E-box. E-Box är kontrollenheten där logiken sköts, Mechanical block hanterar ventilerna och sensorerna och MMI (Man Machine Interface) är gränssnittet mellan ventilationssystemet och personalen. Efter att man en bild av hur systemet fungerar så börjar man fördjupa sig på detaljer. Matematisk modell är mer detaljerad modell som visar systemets logik. Den här matematiska modellen används då kontroll logiken ska läggas in i Simulink-modellen av systemet.

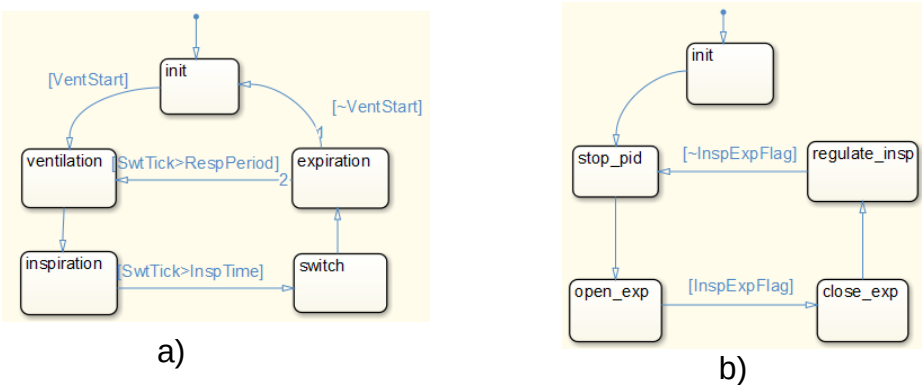


Figur 10: Simulink modell av respirator kontrollenheten tillsammans med subsystemen MMI och Mechanics block [2].



Figur 11: Underliggande subsystem till Medical Ventilator Controller [20].

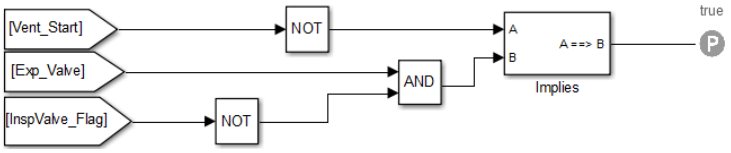
Figur 10 visar en Simulink-modell av respiratorns kontrollenhet och hur den interagerar med MMI-blocket och Mechanics-blocket. Figur 11 visar två stycken subsystem, PhaseSwitch och Controller som är subsystem till respirators kontrollenhet, Medical Ventilator Controller . De här subsystemen ger ut signaler på utgångsportarna. PhaseSwitch-blocket ger ut bl.a. *PSMsg1* och *PSMsg2* och Controller-blocket ger ut PWM_Insp och Relay1OnOff. En del värden är också samma för båda subsystemen. Nästa steg blir att skapa Stateflow-diagram som visar hur subsystemen fungerar. Figur 12 visar Stateflow-diagrammen för subsystemen. Vardera har fem tillstånd att byta mellan där *init* är starttillståndet. Varje tillstånd aktiverar något subsystem för att göra en specifik uppgift



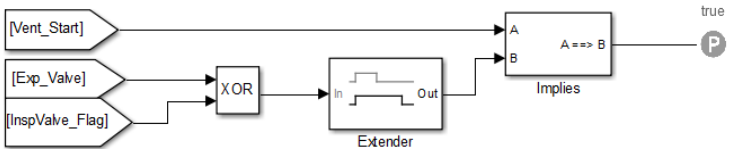
Figur 12: Stateflow-diagram på subsystemen a) PhaseSwitch och b) Controller [2].

När modellen är färdigt modellerad i Simulink är nästa steg att verifiera att alla krav stämmer. Simulink Design Verifier används nu för att verifiera kraven genom att

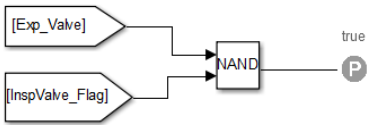
modellera kraven som egenskaper. Respirator kontrollenheten har flera krav som behöver verifieras men här kommer endast några av dem att visas. Första kravet är att systemet måste vara i säkert tillstånd då det startas [2]. Systemet är i säkert tillstånd då ventilationen är stängd, inandningsventilen är fast och utandningsventilen är öppen [2]. Beviset för första kravet visas i figur 13. Implies-blocket ger värdet sant då *A* medför *B* och annars falskt [13]. Andra kravet är att ventileringen ska starta då startsignalen är mottagen [2]. Figur 14 visar beviset för andra kravet. Blocket XOR ger värdet sant endast då ett av värdena är sant och Extender-blocket förlänger sedan sant-signalen med ändligt eller oändligt antal tidssteg [12],[13]. Tredje kravet är att inandningsventilen och utandningsventilen aldrig får vara öppna samtidigt [2]. I beviset i figur 15 används NAND-block för att bevisa det. NAND-blocket ger värdet falskt då båda värdena är sant och annars sant [12].



Figur 14: Bevis av krav 1 med Simulink Design Verifier [2].



Figur 13: Bevis av krav 2 med Simulink Design Verifier [2].



Figur 15: Bevis av krav 3 med Simulink Design Verifier [2].

När alla krav har blivit verifierade och fått godkänt är allting bra men för varje krav som blivit underkänt behövs ännu testning. För varje krav som inte blivit godkänt genereras ett så kallad harness-modell motexempel. Modellen visar värden på alla signaler mot antalet tidssteg. För ett underkänt krav visas signalvågor tills det tidssteg då det blev underkänt. Den här modellen kan återanvändas vid senare testning.

6. Avslutning

Den här avhandlingen har tagit upp säkerhetskritiska system, modellbaserad utveckling och hur modelleringsverktygen Simulink och Stateflow kan användas för att modellera kontrollsystem. Simulink visade sig ha ett användarvänligt gränssnitt där det mesta var lätt att hitta. Simulink har ett bibliotek där alla block är samlade och bra dokumentation som beskriver hur de enskilda blocken fungerar. Placera ut block och sedan dra pilar mellan blockens ingångsportar och utgångsportar, gör det till ett ganska lätt sätt att skapa modeller. Det som är svårare är att veta hur blockens parametrar ska modifieras för att få rätt funktionalitet.

Simulink Design Verifier är verifieringsverktyget som togs upp men det finns också andra verktyg som fungerar med Simulink och Matlab. VerSÅA är ett kontraktbaserat verktyg för att verifiera Simulink-modeller. Kontraktbaserad verifiering fungerar så att subsystemen skrivs om som kontrakt som verktyget sedan verifierar. Kontrakten förklarar subsystemen beteende i modellen och skrivs med en syntax som visas i figur 16. I kontrakten anges bl.a. förvillkor och eftervillkor, typen på parametrar, ingångsportar, utgångsportar.

$$
\begin{aligned}
&\textbf{parameters} : c : type \\
&\textbf{inports} : u : type \\
&\textbf{outports} : y : type \\
&\textbf{memory} : x : type \\
&\textbf{paramcond} : Q^{param} \\
&\textbf{precondition} : Q^{pre} \\
&\textbf{postcondition} : Q^{post} \\
&\textbf{initcondition} : Q^{init} \\
&\textbf{postconditionm} : Q^{postm} \\
&\textbf{refrel} : Q^{refrel}
\end{aligned}
$$

Figur 16: Syntaxen på kontrakt.
Bild från [19]

Modellbaserad utveckling har flera fördelar jämfört med den traditionella metoden. Det är ett snabbare och kostnadseffektivare sätt p.g.a. automatisk kodgenerering och testning kan göras under utvecklingsskedet för att snabbare hitta fel. En bra gjord modell gör det lättare att förstå funktionen i systemet som utvecklas Det är ändå inte alltid ett passligt alternativ. Modelleringsverktygen är ofta dyra om man inte väljer gratis varianter. Verifieringen av stora modeller kan ta lång tid.

Källförteckning

[1] Adams, C. ”Benefits of the Graphical User Interface.” ThoughtCo, thoughtco.com/benefits-of-graphical-user-interface-1206357. (hämtat 10.2.2018).

[2] Bijo, S., ”A comparison of Simulink Design Verifier and VerSÅA.” Master’s thesis, Åbo Akademi University, 2014

[3] Boström, P., ”Formal design and verification of system using domain-specific languages”. PhD thesis, Åbo Akademi University (TUCS), 2008.

[4] Bowen, J.,”The ethics of safety-critical systems”, Communications of the ACM, vol. 43, no. 4, pp. 91-97, 2000.

[5] Chen, C., Dong, J. S., Liu, Y., Sun, J. and Zheng, M. ,”Formal modeling and validation of Stateflow diagrams.” STTT, vol. 14, no. 6, pp. 653-671, 2012

[6] Esterel Technologies. SCADE. http://www.esterel-technologies.com/products/scade-suite/, 2018

[7]”Graphical User Interface.” Encyclopedia of Small Business. . Encyclopedia.com. 27 Feb. 2018 <http://www.encyclopedia.com>. (hämtat 27.2.2018)

[8] Knight, J. C., "Safety critical systems: challenges and directions," Proceedings of the 24th International Conference on Software Engineering. ICSE 2002, Orlando, FL, USA, 2002, pp. 547-550.

[9] Lustre. http://www-verimag.imag.fr/The-Lustre-Programming-Language-and.html?lang=en, 2018

[10] Macintosh – 1984 by Apple Computer (hämtat 20.2.2018) http://oldcomputers.net/macintosh.html

[11] MathWorks Inc. Matlab.
https://se.mathworks.com/products/matlab.html, 2018

[12] MathWorks Inc. Simulink.
https://se.mathworks.com/products/simulink.html, 2018.

[13] MathWorks Inc. Simulink Design Verifier
https://se.mathworks.com/products/sldesignverifier.html, 2018

[14] MathWorks Inc. Stateflow.
https://se.mathworks.com/products/stateflow.html, 2018

[15] Mayer, R. E., Bove, W., Bryman, A., Mars, R. and Tapangco, L., ”When less is more: Meaningful learning from visual and verbal summaries of science textbook lessons.”, Journal of Educational Psychology, vol. 88, no. 1, pp. 64-73, 1996.

[16] Scilab Consortium. Scilab/Xcos.
https://www.scilab.org/en, 2018

[17] Sulyman, M. and Ali, S., ”Applying model checking for verifying the functional requirements of a scania’s vehicle control system.”, Master’s thesis, Mälardalen University, Sweden.

[18] V-Model figur 1 (hämtat 27.2.2018)
https://www.utest.com/articles/theory-of-software-testing-part-2-software-development-cycles

[19] Wiik, J., ”Contract-based verification of multi-rate Simulink models.” Master’s thesis, Åbo Akademi University, 2012

[20] Zhou, F., Angelov, C., Guan, W. and Sierszecki, K., ”Component-based design of software for embedded control systems: The medical ventilator case study”, International Conference on Embedded Software and Systems, ICESS’09, 2009, pp. 157-163