Anpassning av Banverkets program BARTRAD med moduler för kontaktledningsdynamik

Adjustment of Banverkets program BARTRAD with modules for overhead contact line dynamics

Kristofer Borg

EXAMENSARBETE
Datateknik
Nr: E3188D

2005

HÖGSKOLAN
Dalarna

EXAMENSARBETE, C-nivå
Datateknik

Program	Reg nr	Omfattning
Datateknik, 120 p	E 3188 D	10 p
Namn Kristofer Borg	Datum 2005-05-23	
Handledare Lars-Erik Cederlöf	Examinator Ernst Nordström	
Företag/Institution Banverket	Kontaktperson vid företaget/institutionen Sven Lundbäck	
Titel Anpassning av Banverkets program BARTRAD med moduler för kontaktledningsdynamik		
Nyckelord Kontaktledningssystem,bärtråd, dynamik		

Sammanfattning

Syftet med detta examensarbete är att utveckla applikationsmoduler till simuleringsdelen av Banverkets program BARTRAD för att kunna generera sektionsövergångar och kurvor samt beräkningskommandon för dynamiska simuleringar på kontaktledningssystemet. Dessa moduler skall generera utdata i form av noder och element som representerar kontaktledningssystemet i 3D med hjälp av Ansys/LS-Dyna. Ansys/LS-Dyna skall också utföra beräkningar och simuleringarna på kontaktledningssystemet. Simuleringsresultaten skall valideras mot EN 503 18.
Detta arbete presenterar en bra grund som kan konstruera sektionsövergångar och kurvor samt beräkningskommandon. Detta kan Banverket använda för att utöka sina kunskaper om kontaktledningsdynamik, men vidareutveckling av sektionsövergångar i kurvor samt beräkningskommandona behövs för att kunna utnyttja systemet till fullo.

HÖGSKOLAN
Dalarna

Degree Project
Computer Engineering

Programme	Reg number	Extent
Datateknik, 120 p	E 3188 D	10 ECTS
Name of student Kristofer Borg	Year-Month-Day 2005-05-23	
Supervisor Lars-Erik Cederlöf	Examiner Ernst Nordström	
Company / Department Banverket	Supervisor at the Company/Department Sven Lundbäck	
Title Adjustment of Banverkets program Bartrad with modules for overhead contact line dynamics		
Key words Droppers, dynamic, over head contact line		

Summary

The purpose with this thesis work is to develop application modules to the simulations parts of Banverkets (Swedish national railway administrations) program BARTRAD that shall generate sections overlaps, curves and commands for solving dynamic calculation on overhead contact line. These modules shall generate nodes and elements that represent the overhead contact line in 3D space with the help of Ansys/LS-Dyna. Ansys/LS-Dyna shall also perform the calculations and simulations on the over head contact line. The simulations results shall be validated against EN 503 18.
This work presents a solid base that can construct section overlapps, curves and commands for solving dynamic calculation on overhead contact line. This could be used by Banverket to expand their knowledge of overhead contact line dynamics, but further development of section overlaps in curves and the commands is needed for fully uses of this program.

Innehållsförteckning

1 Inledning

1.1 Bakgrund

Vid konstruktion av kontaktledningssystem är det viktigt att kontakttråden är så spänd som det bara är möjligt, detta för att tåget skall kunna ha så hög hastighet som möjligt. För att kontakttråden skall vara helt rak krävs det en oändligt stor inspänningskraft, vilket är omöjligt att uppnå. Idag konstruerar man kontaktledningssystemen så att det sitter vikter i stolparna som ger inspänningskraften, men för att tråden skall bli tillräckligt rak krävs en bärlina. Denna bärlina har ett varierande nedhäng, p.g.a. tyngdlagen, därför är det viktigt att bärtrådarna, som med hjälp av bärlinan, bär upp kontakttråden är av rätt längd.

Banverket har idag ett eget utvecklat program som beräknar bärtrådarnas längder och kan utföra simuleringar på de kontaktledningssystem som användaren har matat in. Banverket behöver dock nya applikationsmoduler som konstruerar sektionsövergångar och kurvor samt en modul som konstruerar parametrar för dynamiska beräkningar i kontaktledningssystemet.

1.2 Syfte

Syfte med detta examensarbete är att utveckla applikationsmoduler för beräkning av kontaktledningsdynamik för en modell som i 3D beskriver en kontaktledning. Programmet utvecklas i syfte att användas av Banverkets huvudkontor i Borlänge för att utveckla nya samt undersöka och förbättra nuvarande kontaktledningssystem. Banverket skall också ha möjligheter att utöka sina kunskaper om kontaktledningsdynamik med hjälp av detta program samt ge support till banregionerna för speciella kontaktledningskonfigurationer.

1.3 Mål

Målet med detta arbete är att utveckla applikationsmoduler som genererar en utdatafil som är förberedd för beräkning av kontaktledningsdynamik i 3D som uppfyller kraven ur EN503 18. Utdatafilen skall vara genererad på så sätt att Ansys/LS-Dyna klarar av att utföra beräkningar på denna. Programmet skall också vara kapabelt till att generera kurvor och sektionsövergångar.
Mer personliga mål är att uppnå ökade kunskaper inom matematik och programmering samt planering och design av applikationer.

1.4 Avgränsningar

Detta arbete avgränsar sig på följande punkter:

- En kurva har endast en radie
- Programmet kommer inte att ta hänsyn till rälsförhöjning eller korglutning i kurvor
- Det förekommer endast en strömavtagare vid simulering

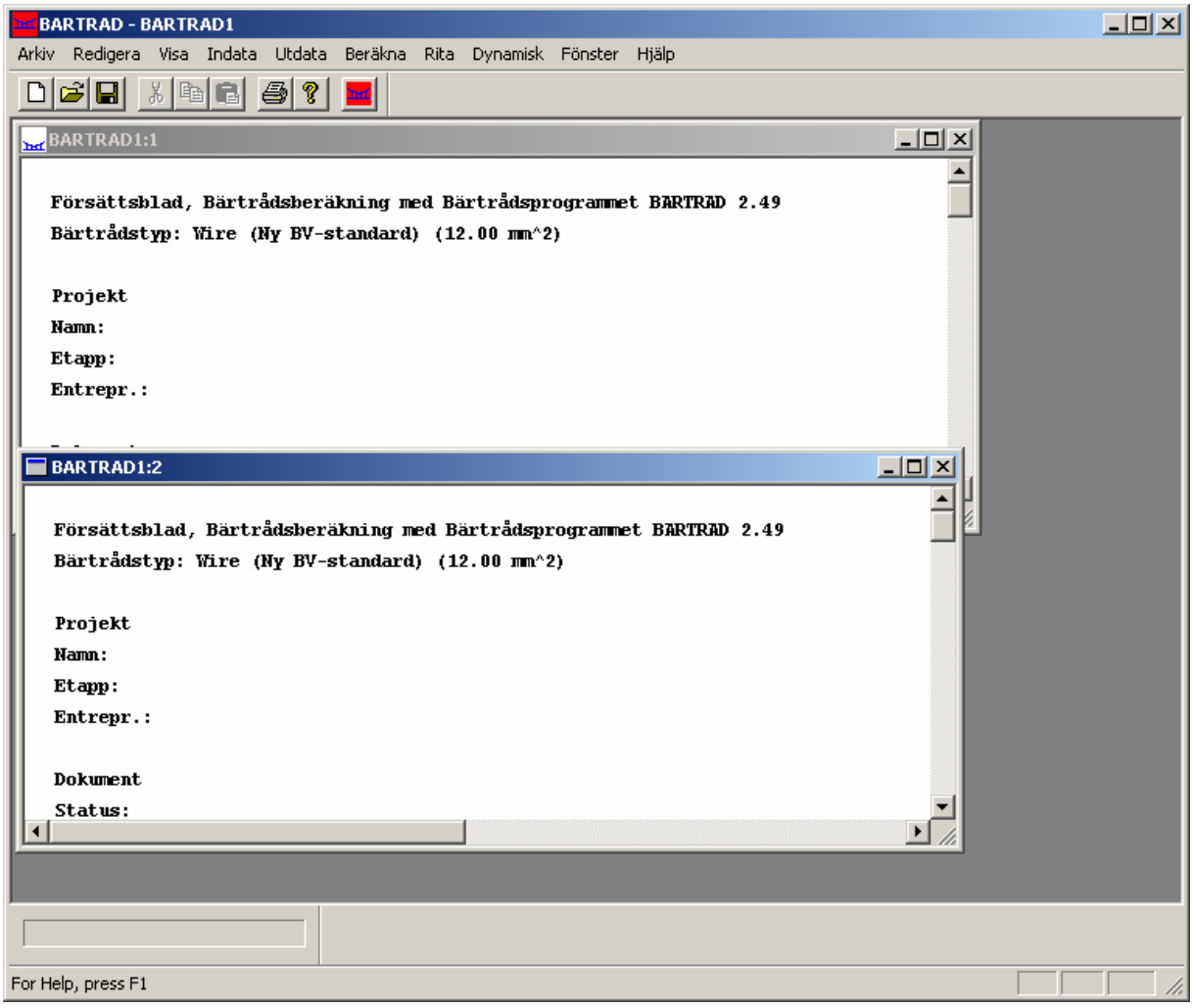
2 Tidigare arbete

2.1 Capasim

Capasim (Catenary pantograph simulation) är namnet på den samling program som används vid simulering av kontaktledningssystem och konstruktion av kontaktledningssystem i dag på Banverket.

Denna samling består bl.a. av ett egenutvecklat program som heter BARTRAD. Detta program klarar av att utföra beräkningar på bärtrådar och skapa filer som kan simuleras. Denna simulering utförs sedan av en ytterligare del i Capasim, nämligen Ansys. Denna applikation är ett kommersiellt utvecklat verktyg som klarar av att simulera olika modeller, med olika materialegenskaper. Nästa del i denna samling program är ett annat eget utvecklat program som heter main.exe och utför postprocessbehandling av datat som produceras av Ansys och denna behandling resulterar i ett antal Matlabfiler, som kan exekveras i just Matlab för att producera önskade grafer. Matlab är också ett kommersiellt utvecklat program som utför avancerade matematiska beräkningar och kan producera grafer.

2.2 BARTRAD



Figur 1. Visar en bild på bärtrådsprogrammet.

BARTRAD, eller bärtrådsprogrammet som det också kallas, består av två delar. Den första delen gör beräkningar på kontaktledningssystemet, framförallt beräknar programmet längder på de bärtrådar som hör till de spann som användaren har matat in i passande fält.

Den andra delen utför förberedelser till simuleringar. Denna del skapar en utdatafil som skall vara färdig att importera i Ansys/LS-Dyna för att sedan utföra beräkningar och simuleringar på detta utdata. Till denna fil skrivs information om uppbyggnaden av kontaktledningssystemet med hjälp av noder och element, vidare så skriver programmet också ner materialegenskaper för dessa element och noder så att LS-Dyna kan ta hänsyn till detta vid simuleringar. Man bifogar också en strömavtagarmodell, som i dagsläget är en

ganska simpel modell. I Figur 1 kan man se den första ruta som dyker upp då man startar programmet.

2.3 Papasim

Papasim utvecklades för att underlätta simuleringsdelen i Capasim programmen. Programmet fungerar som ett skalprogram, där användaren enkelt kan skapa och exekvera flera olika simuleringar på en och samma gång. Användaren matar in en fil i Papasim som konstrueras i Bartrad för att beskriva kontaktledningens uppbyggnad, sedan kan användaren via Papasim ange de parametrar som ska variera i de olika beräkningarna.
När användaren sedan startar programmet skapas det olika datafiler och mappar för varje beräkning, dessa datafiler och mappar organiseras utifrån de parametrar man valt att variera. Programmet startar sedan själva beräkningen, och de utdatafiler som skapas när programmet är klart konstrueras till en ny fil som kan exekveras i Matlab för att generera de grafer som användaren valt i Papasim.
Dessa simuleringar kan ta lång tid därför är det viktigt att all data blir korrekt. Innan detta program utvecklades så utfördes en hel del klipp och klistra-arbete för hand, vilket kunde generera fel och långa beräkningar var bortkastad tid.

3 Teori

Denna del handlar om teori bakom kontaktledningssystemet samt teori hur programmet hanterar konstruktion av rakspår, kurva eller sektionsövergång.

3.1 Kontaktledningssystemets uppbyggnad

3.1.1 Kontakttråd

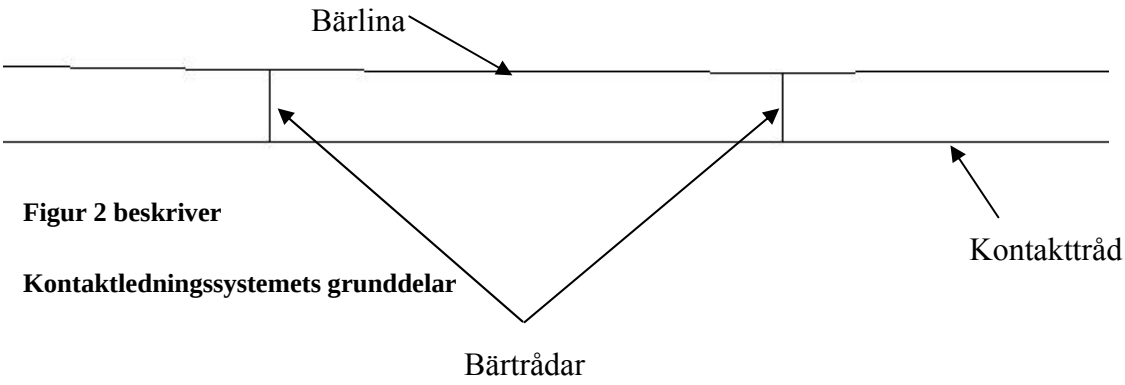
Kontakttråden är den ledning som strömavtagaren åker längs och det är även från denna ledning som strömmen kommer ifrån. Kontakttråden är kopplad till stolparna via s.k. tillsatsrör. Denna del av kontaktledningssystemet kan ses i Figur 2.

3.1.2 Bärlina

Bärlinan är den lina som med hjälp av bärtrådarna håller upp kontaktledningen. Bärlinan är kopplad direkt till de olika stolparna. Denna del av kontaktledningssystemet kan ses i Figur 2.

3.1.3 Bärtrådar

Bärtrådarna kan ses som supporttrådar då kontaktledningen skall hängas upp. Bärtrådarna är designade så att de kan leda ström, vilket kan ses i figur 4.22b i (Kiessling, Puschmann, Schmider, 2001, s.154). Denna del av kontaktledningssystemet kan ses i Figur 2.



Figur 2 beskriver

Kontaktledningssystemets grunddelar

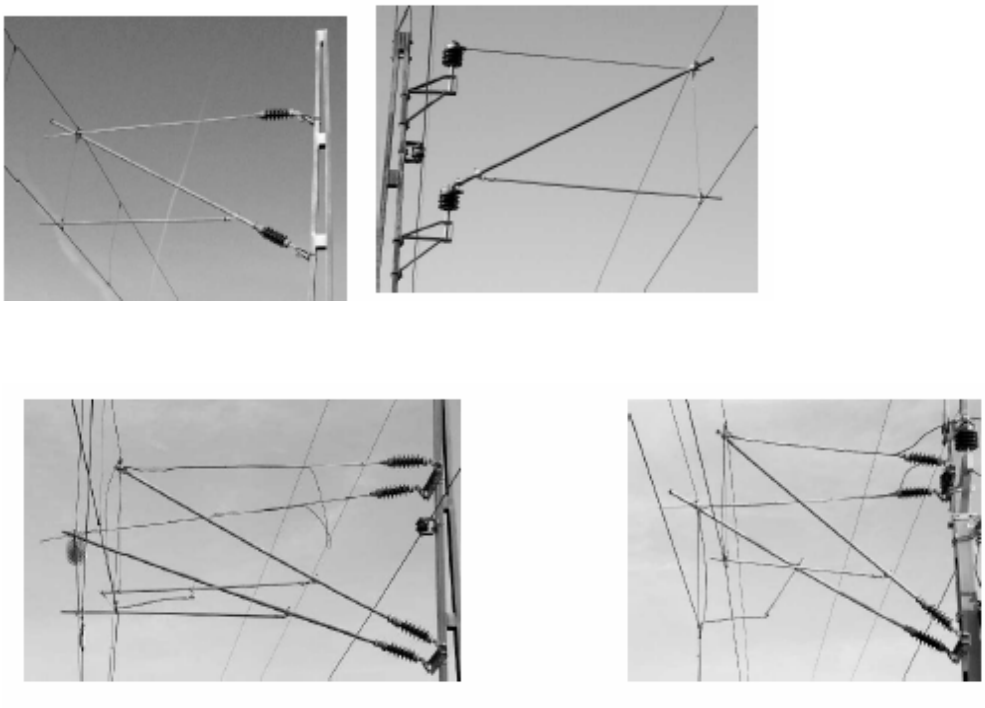
3.1.4 Strömavtagare

Strömavtagaren är den del av tåget som är placerad på taket och har kontakt med kontaktledningen och hämtar strömmen från denna. Antalet strömavtagare kan variera från tåg till tåg och det kan också förekomma flera strömavtagare. Moderna strömavtagare består oftast av en head och en frame (Drugge, 2000, s.2). I Figur 4 kan man se en bild på strömavtagarmodellen så som den var beskriven i Ansys/LS-Dyna.

3.1.5 Utliggare

En utliggare, eller upphängningspunkt som de också kallas, monteras på de stolpar som skiljer spannen från varandra. En del av dessa utliggare monteras korta medan andra monteras långa, på detta vis uppstår en s.k. zick-zack i förhållande till spårets mittlinje.

I Figur 3 kan man se olika utliggare.



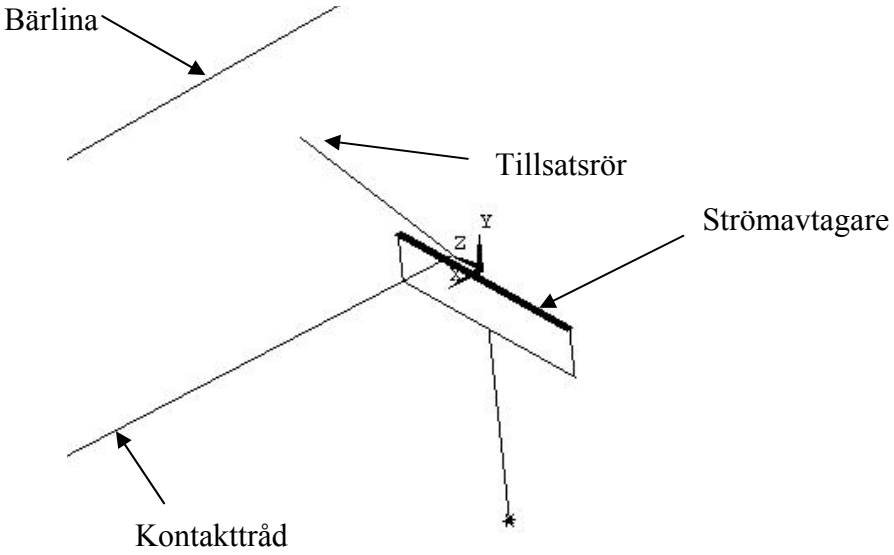
Figur 3. Bilder på olika utliggare där de undre bilderna tillhör en sektionsövergång.

3.1.6 Tillsatsrör

Tillsatsrören länkar kontaktledningen till stolparna, jämfört med bärlinan som är direkt kopplad till stolparna. Rören sitter monterade på utliggarna. I Figur 3 kan man se på de undre bilderna att utliggarna har tillsatsrör monterade på sig.

3.1.7 Zick-zack

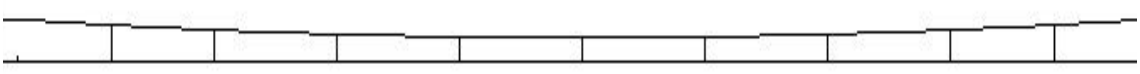
Om man tittar på kontaktledningssystemet uppifrån kan man se att det går i zick-zack i sidled. Detta för att strömavtagarens kolslitsskena inte skall slitas på ett enda ställe och få bytas ut på en gång. Eftersom ytan är liten där kontakt sker så skulle detta ske på en väldigt kort tid om man inte varierade kontaktytan mellan kontaktledningssystemet och strömavtagaren med hjälp av zick-zack på kontaktledningen.



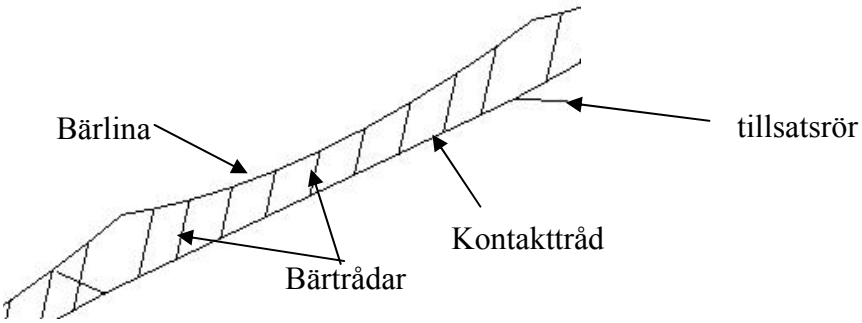
Figur 4. Visar en bild på strömavtagaren så som den såg ut representerad i Ansys / LS-Dyna.

3.1.8 Spann

Kontaktledningssystemet är uppbyggt med ett visst antal spann, där varje spann består av ett visst antal bärtrådar och en bärlina samt kontaktledning. Ett spann börjar med en stolpe och avslutas med en stolpe. Spannets längd kan variera men oftas brukar de vara 60 meter långa. Bild på ett spann kan ses i Figur 5 och i en mer 3D orienterad vy i Figur 6.



Figur 5. Visar ett 60 meter långt spann med tillhörande bärtrådar, kontaktltråd samt bärlina.



Figur 6. Visar också ett 60 meter långt spann, fast denna gång i en mer vinklad 3D vy.

3.1.9 Sektionsövergång

De olika spannen bildar en sektion som brukar vara 1200-1500 meter lång. Där två sektioner möts bildas en sektionsövergång. Vikter sitter i varje ände av sektionen för att få en inspänningskraft i kontaktledningen och låsning av bärlinan sker på mitten. Sektionsövergången består av ett byte mellan två trådar som möts och strömavtagaren har vid sektionsövergången kontakt med båda ledningarna. Växlingen mellan trådarna sker på en sträcka av 10 meter, men hela sektionsövergången kan vara mellan 150 – 300 meter.

3.2 Uppbyggnad av modell för rakspår

Vid konstruktion av rakspår matar användaren in ett visst antal spann under fliken ”Spann” i ”Indata” → ”Ändra”. Programmet startar nodgenerering vid origo och fortsätter i positiv riktning längs x-axeln.

Konstruktion av rakspår är ingen ny funktion i programmet i och med detta arbete, men det ligger som grund vid konstruktion av sektionsövergång och kurva.

3.2.1 Kontakttrådens uppbyggnad

Programmet producerar noder för kontakttråden i positivriktning längs x-axeln. X-värdet på dessa noder ökas kontinuerligt med ett värde som beräknas utifrån det nuvarande spannets position samt nuvarande bätrådens position och det nuvarande elementet mellan den nuvarande bärtråden och nästa bärtråd. Y-värdet för samtliga noder i kontaktledning är av ett konstant värde som beräknas utifrån kontanktledningens höjd samt nedhängsvärdet. Dessa värden anges av användaren i indata dialogen.

3.2.2 Bärlinans uppbyggnad

Konstruktionen av bärlina för rakspår sker också i positiv riktning längs x-axeln. Y-värdet på dessa noder varierar utifrån längden av bärtrådarna samt kontaktledningshöjden och systemhöjden.

3.2.3 Zick-zackens uppbyggnad

Z-värdet på samtliga noder motsvarar zick-zackvärdet, och detta värde beräknas utifrån de trådlägevärden som användaren har matat in i indatadialogen för varje upphängningspunkt.

3.3 Uppbyggnad av modell för kurva

Vid konstruktion av kurva utgår programmet från ett rakspår, d.v.s programmet genererar först ett rakspår. När rakspåret sedan är klart används denna data samt de data som användaren har matat in i indatarutan för att manipulera rakspåret så att det uppstår en kurva. Programmet genererar alltid en högerkurva, men har användaren angett att denne vill konstruera en vänsterkurva speglas de noder som skapades så att en vänsterkurva uppstår.

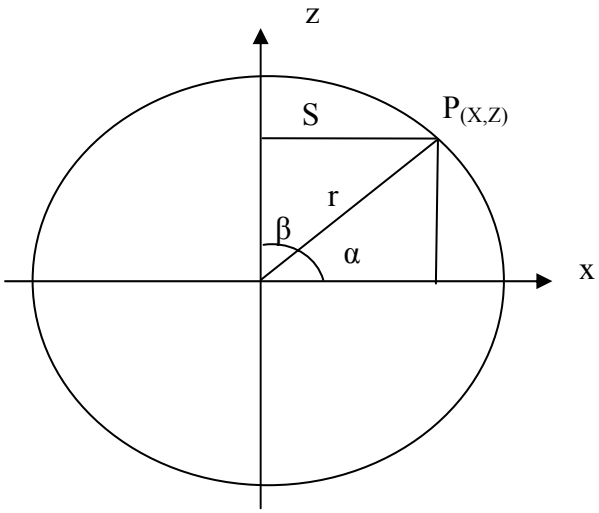
Kurvan som skall konstrueras kommer alltid att ha en konstant radie, det kommer inte i detta arbete att förekomma kurvor med varierande radie. Programmet kommer inte heller att ta hänsyn till att kurvan kommer att luta, d.v.s. noderna som genereras till kurvan kommer inte att förändras i y led. I verkligheten lutar egentligen vagnen när den körs i en kurva på grund av rälsförhöjning av den yttre rälen, detta för att kunna dra nytta av en högre hastighet, om man vill ha en mer realistisk simulering så borde man ta hänsyn till denna faktor, men då detta är en komplicerad process anses det därför inte vara en del av detta arbete.

3.3.1 Ny position för stolparna

Varje stolpe måste få ett nytt värde då man skall skapa en kurva. En stolpe indikerar starten på ett spann.

Eftersom kurvan i fråga endast har en radie och att denna kommer alltid att vara konstant kan vi använda enhetscirkeln för att beräkna stolpens nya position. I enhetscirkeln vet vi att en x-koordinat är cos α och att en y koordinat är sin α. I det här fallet kan vi betrakta sin α som z-koordinaten då vi vet att y-koordinaten kommer att vara konstant. Vidare så har programmet redan beräknat y-koordinaten då den beräknade rakspåret som vi manipulerar på, så y-koordinaten låter vi anta sitt nuvarande värde.

Med hjälp av enhetscirkeln kan man härleda följande:



$$P_x = Cos\alpha$$

$$P_z = Sin\alpha$$

$$Sin\beta = \frac{S}{r}$$

$$\beta = \arcsin(\frac{S}{r})$$

$$\alpha = 90 - \beta$$

$$\alpha = 90 - \arcsin(\frac{S}{r})$$

$$\sin\alpha = \sin(90 - \frac{S}{r})$$

$$\sin\alpha = \sin(\frac{\pi}{2} - \frac{S}{r})$$

$$Stolpe_z = \sin(\frac{\pi}{2} - \frac{S}{r})$$

r är radien som användaren har angett vid inmatning av kontaktledningssystemsdata och S talar om hur långt in på spåret vi är i denna punkt. S är x-koordinaten för motsvarande punkt i det rakspår vi tidigare har beräknat.

På samma sätt kan vi beräkna den nya x-koordinaten i kurvan.

$$Stolpe_x = \cos(\frac{\pi}{2} - \frac{S}{r})$$

Y-koordinaten antar det redan beräknade värdet:

$$Stolpe_Y = Koordinat_Y$$

Med denna formel kan vi placera stolparna längs spårmitt i kurvan, men det är inte det vi vill göra. Kontaktledningssystemet skall byggas upp så att det uppstår en zick-zack, och detta kan vi komplettera med i ovan formel genom att låta

$$R = r + zz$$

där r är den radie som användaren har angett och zz är det z värden som har beräknats ut i tidigare rakspår i motsvarande nod. Detta stoppas in i ovan formel och ger då

$$Stolpe_x = \cos(\frac{\pi}{2} - \frac{S}{R})$$

$$Stolpe_z = \sin(\frac{\pi}{2} - \frac{S}{R})$$

3.3.2 Ny position på kurvan för bärtrådarna

När man nu har beräknat de nya positionerna för stolparna måste man beräkna de nya positionerna för de bärtrådar som är placerade mellan stolparna. Detta kan man göra genom att approximera deras positioner på en rät linje mellan två stolpar. Då antar vi att denna approximation är lika lång som om vi skulle ha placerat bärtrådarna på kurvan.

Detta kan man göra med hjälp av räta linjens ekvation.

$$y = k * x + m$$

Med hjälp av Pytagoras sats (Hamrén, 2001) så kan vi visa att

$$X'bti = \sqrt{\Delta zbti + \Delta x'bti}$$

$$X'bti^2 - \Delta x'bti = \Delta zbti$$

där X'bti är bärtråd i's position på rakspåret och $\Delta x'bti$ är sträckan mellan den nya positionen och den gamla positionen för bärtråden, samma sak gäller för $\Delta zbti$.

$$\Delta x'bti = xbt_2 - xbti_1$$

och

$$\Delta zbt = zbt_1 - zbti_2$$

där xbt_1 är känd som x positionen för den stolpe som bärtråden tillhör. Och $xbti_2$ är den nya positionen för bärtråden.

Kombinera detta med linjens ekvation ger det

$$X'bti^2 - \Delta x'bti = (k * xbti_2 + m) - (k * xbti_1 + m) \rightarrow$$

$$X'bti^2 - (xbt_2 - xbt_1) = (k * xbti_2 + m) - (k * xbti_1 + m) \rightarrow$$

$$X'bti^2 = (k * xbti_2 + m) - (k * xbti_1 + m) + (xbt_2 - xbt_1) \rightarrow$$

$$X'bti^2 = (1+k) * xbti_2 - (1+k) * xbti_1 \rightarrow$$

$$xbti_2 = \frac{X'bti^2 + (1+k) * xbti_1}{(1+k)}$$

Där $xbti_2$ är x koordinaten för bärtrådens nya position i kurvan och k är den approximerade linjens lutning. $xbti_1$ är den nuvarande stolpens position och $X'bti$ är bärtrådens position på rakspåret.

Z-koordinaten får vi fram genom att stoppa in x koordinaten i linjens ekvation:

$$zbti_2 = k * xbti_2 + m$$

där $k = \frac{\Delta y}{\Delta x} = \frac{P2_y - P1_y}{P2_x - P1_x}$

Och

$$m = y_1 - k * x_1 \rightarrow$$

$$m = y_1 - \frac{\Delta y}{\Delta x} * x_1 \rightarrow$$

$$m = P1_z - \frac{P2_z - P1_z}{P2_x - P1_x} * P1_x$$

Där P1 och P2 är koordinaterna för stolparna som bygger upp det nuvarande spannet.

3.3.3 Nya positioner för noder mellan bärtrådarna

Med övriga noder menas de noder som sitter mellan bärtrådarna.

Eftersom vi nu vet positionerna för varje bärtråd kan man interpolera de noder som ska sitta mellan elementen med hjälp av formeln

$$P = P1 + (P2 - P1) * t$$

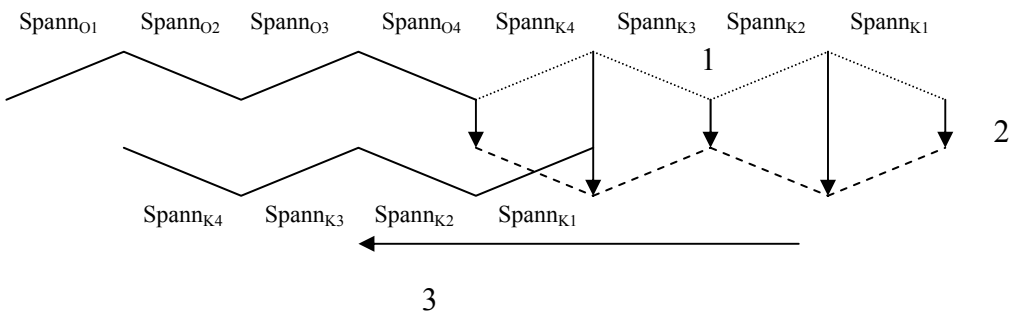
där t är mellan 0 och 1 och P1 och P2 är start och slutpunkt på en line, d.v.s. bärtråd x och bärtråd x+1.

Om t är 0 så är man i början av linjen och om t är 1 så befinner man sig i slutet av linjen.

Med hjälp av detta kan man nu beräkna positionerna för noderna mellan bärtrådarna.

3.4 Uppbyggnad av modell för sektionsövergång

Algoritmen för sektionsövergången går ut på att man först kopierar nuvarande spann som användaren har angett sedan speglar man dessa och sist kontrollerar programmet vilket system som användaren har angett. Utifrån detta system avgör sedan programmet hur många spann som de nya spannen skall dras tillbaka för att bilda själva sektionsövergången. Figur 7 visar en sådan process.



Figur 7 visar konstruktion av en sektionsövergång. Vid 1 sker en kopiering av de spann som användaren har matat in. Vid 2 sker en spegling längs x axeln på kopiorna, och vid 3 sker en tillbaka förflyttning av de kopierade spannen.

3.4.1 Kopiering av de inmatade spannen

Kopieringen av spannen sker i omvänd ordning i förhållande till den ordning som användaren matade in spannen. T.ex. om användaren matade in spannen i ordningen 1,2,3 och sedan väljer att skapa en sektionsövergång kopieras spannen så att ordningen blir 1,2,3,3,2,1 där den sista delen, 3,2,1, är en kopia av de första spannen. Figur 7 visar detta.

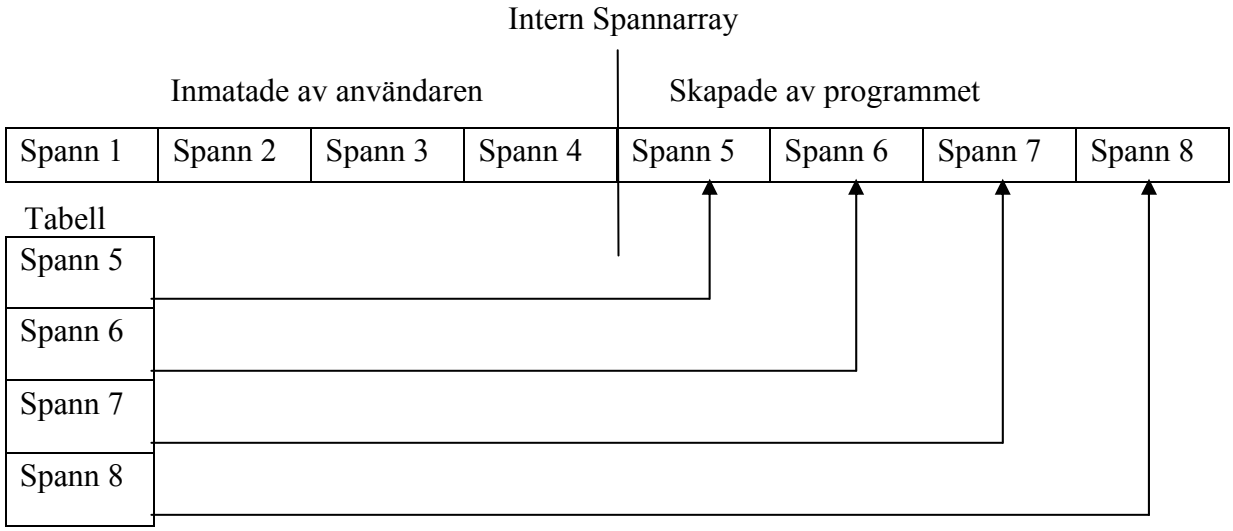
En annan lösning på detta problem hade varit att kopiera kontaktledningssystemet precis som det är och sedan placera kopian i slutet av originalet, nackdelen med denna lösning är att programmet måste då klippa bort onödiga delar av spåret, i början och slut. Detta område är inte intressant vid simulering av sektionsövergången, och då vill man klippa bort det för att spara simuleringstid. Detta kan bli en rätt mödosam process som innebär att man måste manipulera med programmets spannarray då man vill plocka bort delar av spåret.

Efter undersökning av den nuvarande programkoden valdes implementationsalternativ nummer ett från ovan, då speglingsoperation varken är svår att implementera och inte är beräkningsintensiv.

Vid allokering av de nya spannen fanns två alternativ att tillgå. Det första alternativet var att allokera minne till de nya spannen direkt i den redan existerande spannarray som hanterade spannen i programmet. Det andra alternativet var att skapa en tabell som hade pekare till de spann som redan existerade, de spann som fanns i programmets spannarray. Tabellen innehöll också pekare till de nya spannen som allokerats av sektionsövergångsmodulen. Detta kan man se i Figur 8. Vidare så innehåller tabellen ytterligare information som hämtas ur spannarrayen.

Fördelen med den första lösningen är att minnet som allokeras placeras på ett och samma ställe jämfört med i alternativ två där minnet sprids ut lite. Vidare så konstrueras extra pekare i alternativ två, vilket i sig kan vara en källa till att buggar uppstår. Men detta kan man gå runt genom att kapsla in modulen så mycket att yttre faktorer inte kan operera mot dessa pekare och sedan sköta den interna hanteringen av minnet på ett bra sätt.

Fördelen med alternativ två är att vi har en bättre översikt över det nya minnet som vi allokerar. Det visade sig också att när vi väl har allokerat de nya spannen, lagt till dem på lämpligt sätt i systemet och sedan genererat sektionsövergången så var man tvungen att deallokera de nya spann man har allokerat. Misslyckas denna operation leder detta till en fördubbling av spannen. Om användaren har valt att generera en sektionsövergång och direkt efter att utdatafilen har producerats väljer användaren att producera en till utdatafil , kommer spannen att fördubblas varje gång användaren väljer att producera om utdatafilen. Denna fördubbling kan endast stoppas om användaren väljer att avsluta programmet, och sedan öppnar programmet och öppnar filen igen. Detta är ingen bekväm lösningen på problemet. En bättre lösning på detta problem är att deallokera de nya spann som programmet har allokerat. För mer information om hur detta går till se avsnitt 3.4.4.



Figur 8. Visar fördelen med att använda tabellen, nu kan vi peka ut de spann som har allokerats av programmet för att förenkla rensningen av minnet.

Vid ett försök att implementera alternativ ett så visade det sig också att det inte bara var att placera de nyallokerade spannen direkt i spannarrayen, då andra arrayer var direkt beroende av dess storlek. Om spannarrayerns storlek ökade så fattades det element i andra arrayer då deras storlek förblev oförändrad. Då programmet baserar sina bärtrådsberäkningar på spannarrayerns storlek uppstår det en bugg här. Programmet försöker läsa element som inte finns ur en array. Detta kan man tro gynnar alternativ två, men det visade sig vid ett försök att implementera detta alternativ att samma problem uppstod även här. Programmet måste binda de nya spannen till spannarrayern, annars beräknas inte bärtrådarnas längder, vilka är viktiga delar i modellens uppbyggnad.

Detta problem löstes genom att välja implementationsalternativ två ovan och generera tillfälliga element i de arrayer som inte har tillräckligt antal element. Anledningen att alternativ två valdes var för att då får man en bättre översikt av det minne man allokerar vilket är väldigt viktigt när man ska deallokera minnet

3.4.2 Spegling av de kopierade spannen

Speglingen av de nya spannen utfördes i xz-planet mot x-axeln. Detta genomfördes genom invertera det beräknade z värdet för varje koordinat i de noder som definierar spannet.

Detta är en väldigt enkel ur implementation och beräkningssynpunkt.

Figur 7 visar operationen i detalj.

3.4.3 Tillbakadragning av de kopierade spannen

Tillbakadragningen av de nya spannen sker efter att kopiering och spegling har utförts. Detta gör man för att bilda själva sektionsövergången. Framtill nu har sektionsövergången börjat precis där sista spannet av originaldelen slutar, och det är inte detta resultat man vill uppnå, därför skjuter man de nya spannen tillbaka ett visst antal spann, så att den andra kontaktledningen börjar lite innan den första slutar. Figur 7 visar hur detta skall se ut.

Längden som de nya spannen skjuts tillbaka med är summan av ett visst antal spanns spannlängder. Detta antal bestäms utifrån det kontaktledningssystem som användaren matar in via inmatningsrutan. Antingen är det frågan om ett femspannsystem eller ett trespannsystem. Om det är frågan om ett femspannsystem dras de nya spannen tillbaks summan av de tre sista spannlängder. Om det är frågan om ett trespannssystem dras de nya spannen tillbaks med längden av det sista spannet.

3.4.4 Rensning av de kopierade spannen

När man har allokerat de nya spannen och applicerat dessa på ett bra sätt i systemet och genererat utdatafilen som innehåller sektionsövergången, så måste programmet rensa upp bland spannen efter sig. Detta för att antalet spann inte skall fördubblas vid varje generering av utdatafilen, se avsnitt 3.4.1.

Vi har tidigare sett att implementationsvalet blev i form av en tabell. Då kan vi dra nytta av det faktum att vi kan identifiera de spann vi har lagt till i systemet via deras minnesadresser som ligger lagrade i pekarna. Samma sak gäller för de element som vi skapade för att fylla ut arrayer som var beroende av spannarrayerns storlek. Nu blir det väldigt enkelt för programmet att lokalisera och deallokera rätt minne, vilket förhindrar komplexa buggar i programmet.

4 Utförande

Utvecklingen av dessa moduler skedde i tre steg. I det första steget togs UML diagram och flödesscheman fram och i det andra steget implementerades modulerna utifrån dessa diagram. I det tredje steget genomfördes en verifikation att de nya modulerna kommunicerade på ett korrekt sätt, både internt och med den redan existerande applikationen.

4.1 De olika modulerna

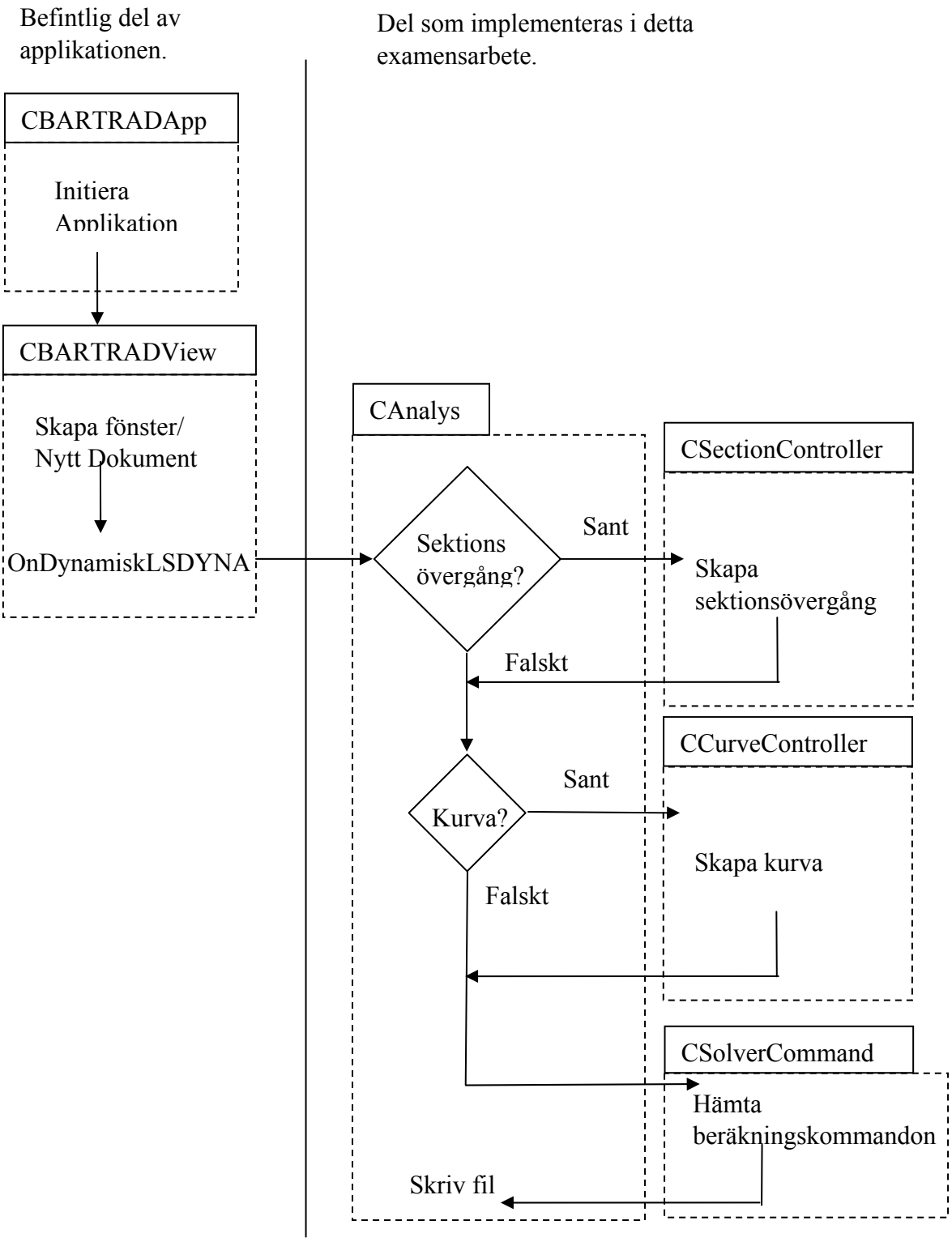
Programmet skulle implementeras objektorienterat, därför valdes modulbegreppet, detta för att det ger många fördelar i underhållsaspekt samt förenklar vidareutvecklingen av programmet. Dessa moduler är självständiga klasser som inte är direkt beroende av de övriga systemen. Kommunikation sker via indata och modulerna producerar någon form av utdata. Designen har gjorts så att så få funktioner som möjligt kan anropas utifrån, detta kapslar in data i modulerna och gör dem mer självständiga. (Prata, 1998, s. 4)

Dessa moduler kommer också att vara ganska globala, d.v.s det kommer att ske en hel del anrop från olika ställen i den redan befintliga applikationen, därför valdes designpattern Singleton vid implementation (Robin, 2002).

4.1.1 Flödesschema

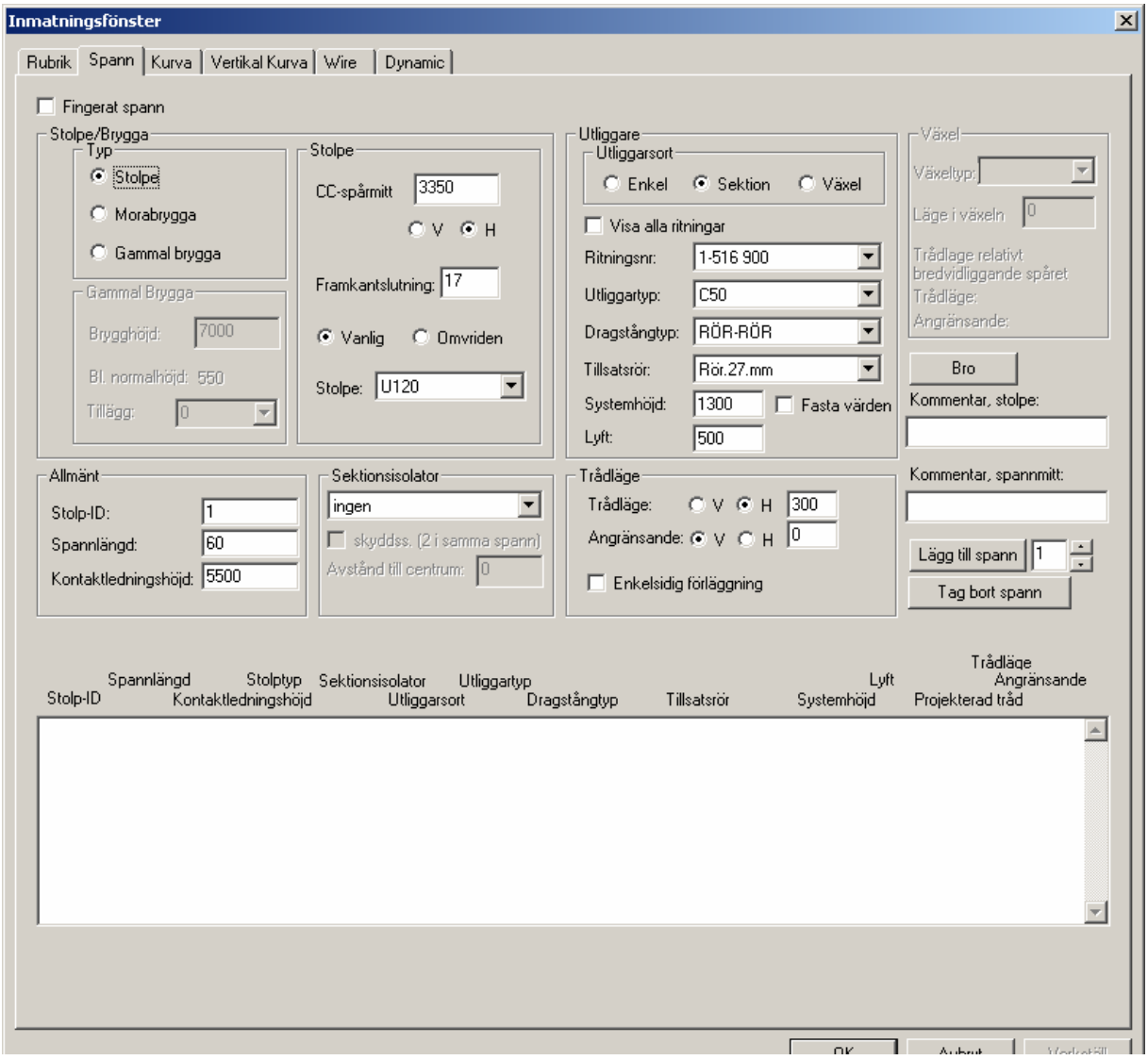
Figur 9 visar hur de olika modulerna interagerar med redan befintligt program. Detta schema visar upp en förenklad bild av nedan UML-diagram för att öka förståelsen för hur de olika delarna kommunicerar. En del delar av processen har lämnats ut då dessa är ganska omfattande och är inte helt av relevans för detta examensarbete.

Allt till vänster om *CAnalys* är delar av den befintliga applikationen. Här kan man se tydlig hur de nya modulerna har adderats in i det vanliga flödet vid konstruktion av modellen.



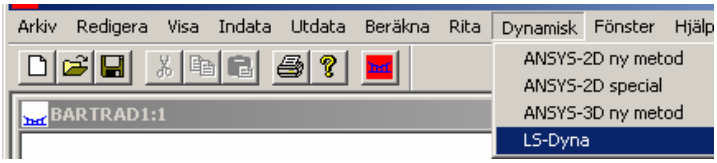
Figur 9. Visar ett flödesdiagram över hur samtliga nya applikationsmoduler kommunicerar med befintligt program

Hela händelseförloppet börjar med att man laddar in en befintlig BARTRAD-fil (.bap) som beskriver modellens uppbyggnad, eller så väljer man att skapa en egen modell från början, genom att mata in spann. En dialog för inmatning kan ses i Figur 10.



Figur 10. Visar dialogen för inmatning av spann, i det vita området presenteras alla skapade spann.

När man har gjort detta kan man välja att skapa en modell för simulering av dynamiska analyser, men för att göra en sådan analys, måste man aktivera det dynamiska läget i programmet. Detta gör man via CTRL+SHIFT+D. När detta är klart adderar programmet ytterligare en rad i menyn som heter Dynamisk, nu kan man välja det ”Dynamiska” menyalternativ och välja ”LS-Dyna”. Detta kan man se i Figur 11.



Figur 11. När man har aktiverat det dynamiska läget kommer ett nytt alternativ upp i menyn. Härifrån kan man producera utdatafiler som sedan används vid simulering.

Nu körs funktionen *OnDynamiskLSDYNA* i Figur 9. Det första som händer i denna funktion är att applikationen kontrollerar ifall användaren har valt att skapa en sektionsövergång. Ifall användaren har valt att göra detta måste programmet skapa de extra spann som krävs samt utföra spegling och tillbakadragning av de spann som berörs, se avsnitt 3.4 för ytterligare information om detta. Detta avspeglas i Figur 9 i den första romben, som också markerar att ett val i flödet sker här. Om det skulle uppstå fel i skapandet meddelar programmet detta för användaren, och användaren kan då välja ifall programmet skall fortsätta generera utdatafilen eller avsluta.

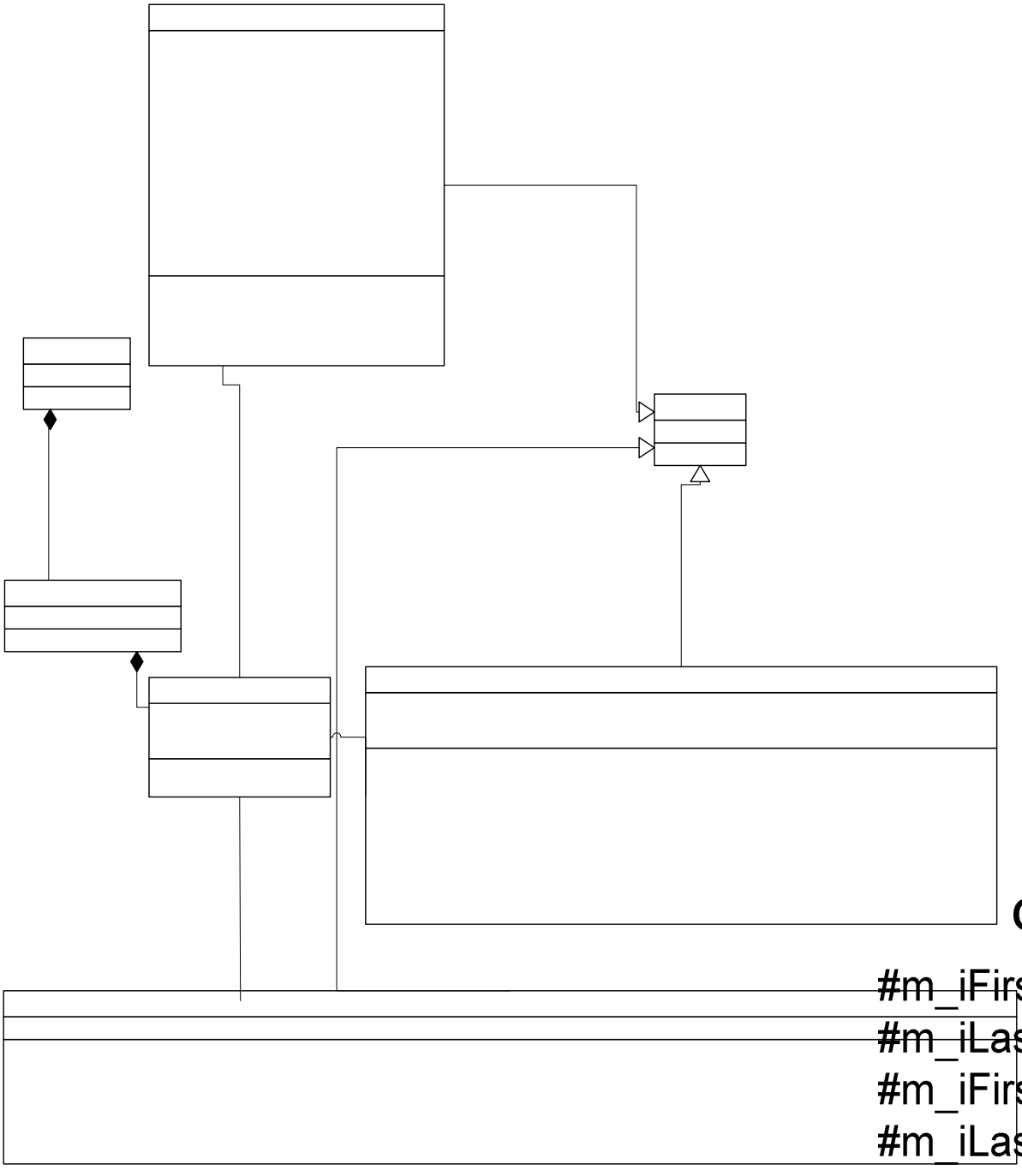
Nästa steg i processen är att bärtrådarnas längder i modellen beräknas, denna del har uteblivit i Figur 9 på grund av att det inte är en del av detta arbete.

Nu är det dags för programmet att generera noderna som beskriver modellen i en 3D-rymd. Det är med hjälp av dessa noder LS-Dyna kan bygga upp en 3D-modell av de spann användaren har angett. Men det krävs lite mer än bara noder för att få en visuellt bra modell, detta sköter programmet genom att skapa element som fogar samman dessa noder och ger en mer visuellt anpassad bild av modellen. Varje element har också sina egna materialegenskaper, så som t.ex koppar eller stål, och detta blir sedan ett hjälpmedel för LS-Dyna vid analys och simulering av modellen. Nodgenereringsfunktionen har också uteslutits i Figur 9, detta för att stora delar av *CAnalys*-delen är en del av nodgenereringen. Modulerna för sektionsövergång och kurvhantering kommer sedan in i detta flöde och manipulerar de noder som genereras. På detta vis slipper modulerna implementera egna funktioner för att generera noder.

Nästa romb i flödesschemat indikerar att det är dags att avgöra om användaren har valt att göra en kurva. Om fallet är så måste programmet manipulera de noder som har genererats för spåret så att de bildar en kurva, för ytterligare information om detta se avsnitt 3.3.

4.1.2 Anpassning av moduler i befintlig applikation

I Figur 12 kan man se ett UML diagram över hur de nya modulerna passar in i det befintliga programmet BARTRAD, samt hur de kommunicerar med BARTRAD. Stora delar av den befintliga applikationen har uteslutits i nedan diagram, en anledning är den stora yta detta skulle ta och en annan är för enkelhetens skull.



Figur 12. Visar ett UML diagram över hur de nya modulerna passar in i den befintliga applikationen

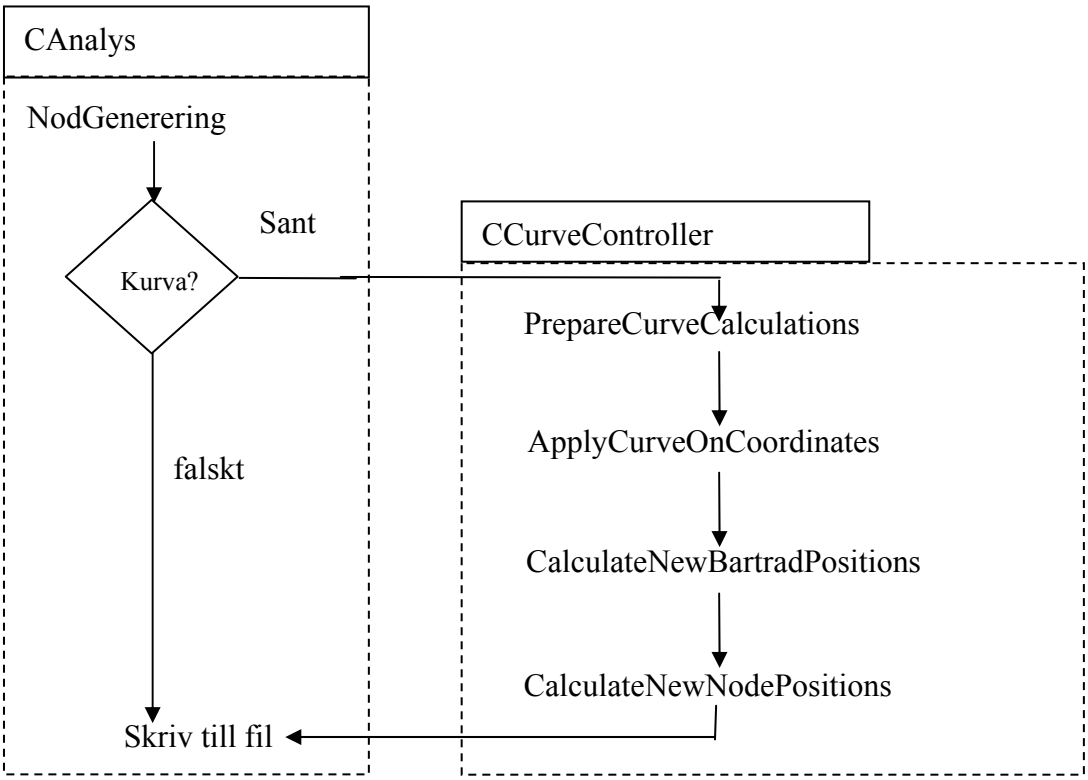
CBARTRADApp är den klass som startar upp programmet, det är här programkoden börjar sin exekvering och själva ramen för programmet skapas här. *CBARTRADView* är den klass som representerar det dokument som man ser i programmet för närvarande. En analys startas då man trycker på ”Dynamisk” → ”Ls-Dyna” i programmet. Detta kommando orsakar ett funktionsanrop till *OnDynamiskLSDYNA* i *CBARTRADView*. I denna funktion skapas en instans av objektet *CAnalys* och utifrån denna instans anropas funktionen *CLSDynaAnalys*, som producerar en utdatafil utifrån valda kommandon och parametrar. Denna utdatafil kan man sedan importera i Ansys/LS-Dyna och exekvera en simulering.

4.2 Kurvmodulen

4.2.1 Beskrivning av kurvmodulen

Denna modul har i uppgift att definiera en kurva eller ett rakspår samt manipulera de noder som har genererats för rakspåret så att de bildar en kurva. Man kan se ett rakspår som en kurva med en oändlig radie.

4.2.2 Flödesschema



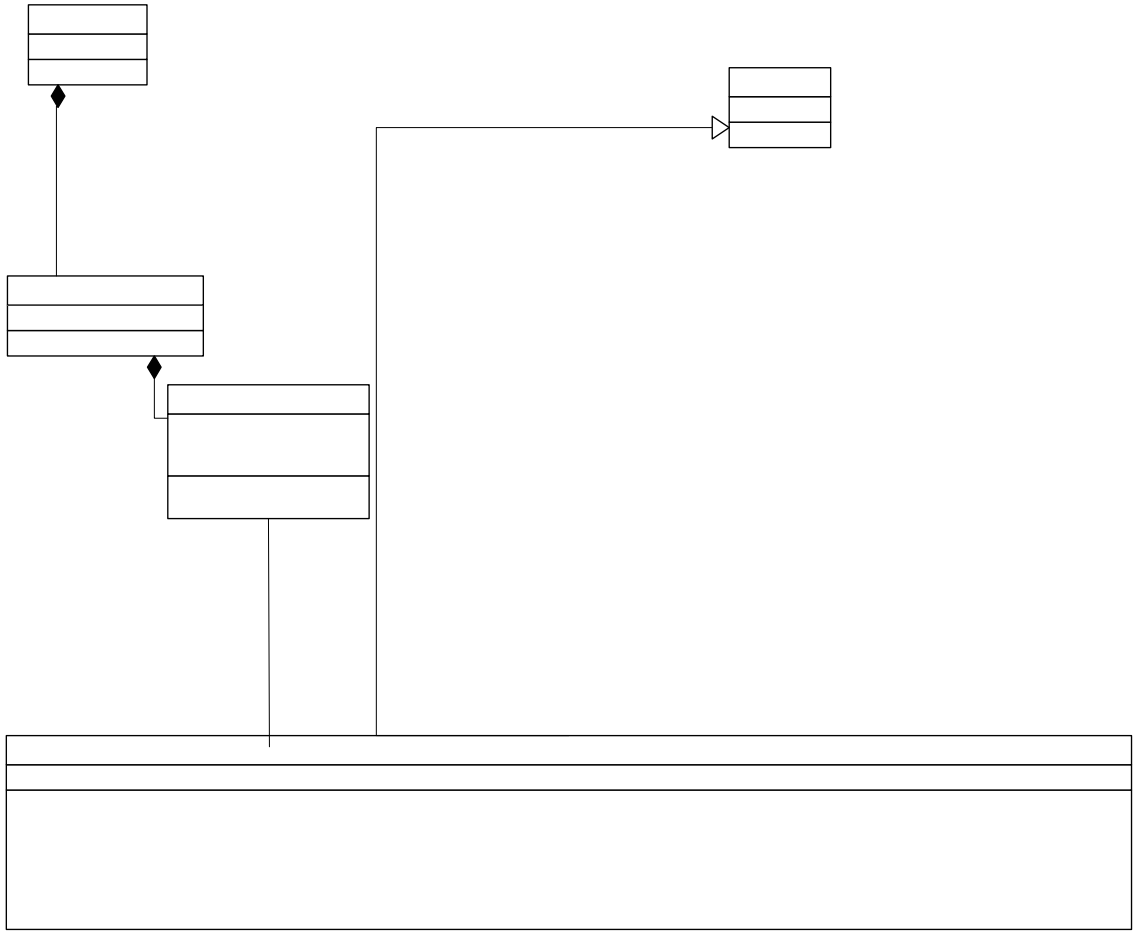
Figur 13. Visar ett flödesdiagram över kurvmodulen och hur denna modul kommunicerar med befintligt program

I Figur 13 kan man se att efter det att noderna har genererats tar programmet ställning till huruvida det skall konstruera en kurva av dessa noder eller ej. Detta är ett beslut den gör utifrån användarens tidigare inmatningar. Om användaren har angett att det skall produceras en kurva anropas *CCurveController* modulen och konstruktion av kurva påbörjas. Den första funktionen, *PrepareCurveCalculations,* har i uppgift att beräkna de nya positionerna för de stolpar som sitter i början och slutet på varje spann, att applicera dessa på kurvan. *ApplyCurveOnCoordinates* funktion används i detta fall för att producera önskat resultat.

Funktionen *CalculateNewBartradPositions* har sedan i uppgift att utifrån de nya positionerna för stolparna kalkylera nya positioner för de olika bärtrådarna och slutligen har *CalculateNewNodePosition* i uppgift att beräkna de nya postionerna för de noder som sitter mellan bärtrådarna. Den sista funktionen är en del av den näst sista funktionen, d.v.s *CalculateNewNodePositions* är ingen egen funktion utan en del av *CalculateNewBartradsPositions* men eftersom detta är en ganska stor del har denna del separerats från dess funktion för att förenkla flödet.

4.2.3 UML-diagram över kurvmodulen

I Figur 14 kan man se ett UML-diagram över kurvmodulen. Diagrammet i Figur 14 visar vilka funktioner som implementeras i modulen samt vilka datamedlemmar modulen implementerar. Nedan följer en mer detaljerad beskrivning av funktionerna samt deras in- och ut parametrar. Det finns också en kortare beskrivning av modulens datamedlemmar.



Figur 14. Visar ett UML-diagram över kurvmodulen.

ApplyCurveOnCoordinates: **CBARTRADApp**

Denna funktion accepterar en koordinat, som motsvarar en punkt i en 3D-rymd (x,y,z) och det nuvarande dokument som programmet arbetar mot. Via detta dokument kan funktionen sedan hämta data som är intressant för denna funktion och dess beräkningar. Funktionen beräknar sedan ut en ny position för denna koordinat, fast då på en kurva. För mer information om hur detta går till se avsnitt 3.3.1. För att göra beräkningen krävs att funktionen vet hur långt in på rakspåret koordinaten befinner sig, hur många meter längs rakspåret den befinner sig. Denna information hämtar funktionen från X variabeln bland inparametrarna. Koordinatens x värde på rakspåret beräknas tidigare vid nodgenerering för rakspåret och skickas med till funktionen som parameter X.

Parameter X, Y och Z är även referensparametrar vilket medför att ändringar i dessa variabler speglar direkt av sig på de variabler som skickades med till funktionen, därför genererar denna funktion inget direkt utdata.

ControllCurveProperties:

Denna funktion kontrollerar om en kurva eller ett rakspår har rätt egenskaper, dessa kontroller utförs enligt (Lindberg, 2004). Det man främst är intresserad av att kontrollera är att zick-zacken har konstruerats korrekt utifrån det system man valt samt radien på kurvan. Avståndet mellan upphängningspunkterna för två ledningar i samma spann är också en intressant faktor som man kontrollerar så att dessa blivit större än eller lika med 400 mm.

Indata för denna funktion är systemet kontaktledningen tillhör samt en datastruktur som beskriver kontaktledningssystemet. Utdata är sant eller falsk beroende på resultatet av alla kontroller.

CalculateNewBartradPositions:

Denna funktion beräknar den nya positionerna för bärtrådarna då man skapar en kuva. För mer information om hur detta går till se avsnitt 3.3.2.

Indata till denna funktion är pekare till de arrayer där alla noder lagras, en array med nodid som pekar ut vilka noder som är stolpar, antalet element mellan varje bärtråd samt det nuvarande dokumentet. Med hjälp av denna information kan funktionen beräkna vilka noder som är bärtrådar och sedan beräkna deras nya positioner.

Ingen utdata genereras av denna funktion på grund av att x, y och z är pekare till redan befintliga arrayer och detta resulterar i att om man manipulerar dessa så kommer också de ordinarier arrayer, de som skickades med till funktionen, att bli manipulerade.

Denna funktion är egentligen tvådelad, den första delen är beskriven ovan, där man beräknar de nya positionerna för bärtrådarna, och den andra delen beräknar nya positioner för de noder som befinner sig mellan bärtrådarna. För mer information se avsnitt 3.3.3.

PrepareCurveCalculations:

Denna funktion beräknar de nya positionerna för upphängningspunkterna (stolpar) på kurvan. För mer information om hur detta går till se avsnitt 3.3.1.

Indata till denna funktion är pekare till de arrayer där alla noder lagras, en array med nodid som pekar ut vilka noder som är stolpar samt det nuvarande dokumentet. Med hjälp av denna information kan funktionen plocka ut de noder som motsvarar stolparna, beräkna nya positioner för dessa på kurvan och lagra resultatet i de arrayer som skickas med till

funktionen. Eftersom de parametrar som motsvarar arrayer för x, y och z koordinaterna är pekare, så pekare dessa på de arrayer som skickas med till funktionen. Detta medför att all manipulation med dessa arrayer kommer att speglas i de arrayer som skickas med till funktionen.

DefineCurve:

Denna funktion har i uppgift att definiera en kurvtyp utifrån vald radie. Detta görs enligt (Lindberg, 2004) och indata till denna funktion är radien och utdatat utgörs iform av en enumrerad datatyp som talar om kurva är korrekt.

Denna funktion används bl.a. vid kontroll av kurva eller rakspår.

m_bMirrorThisCurve:

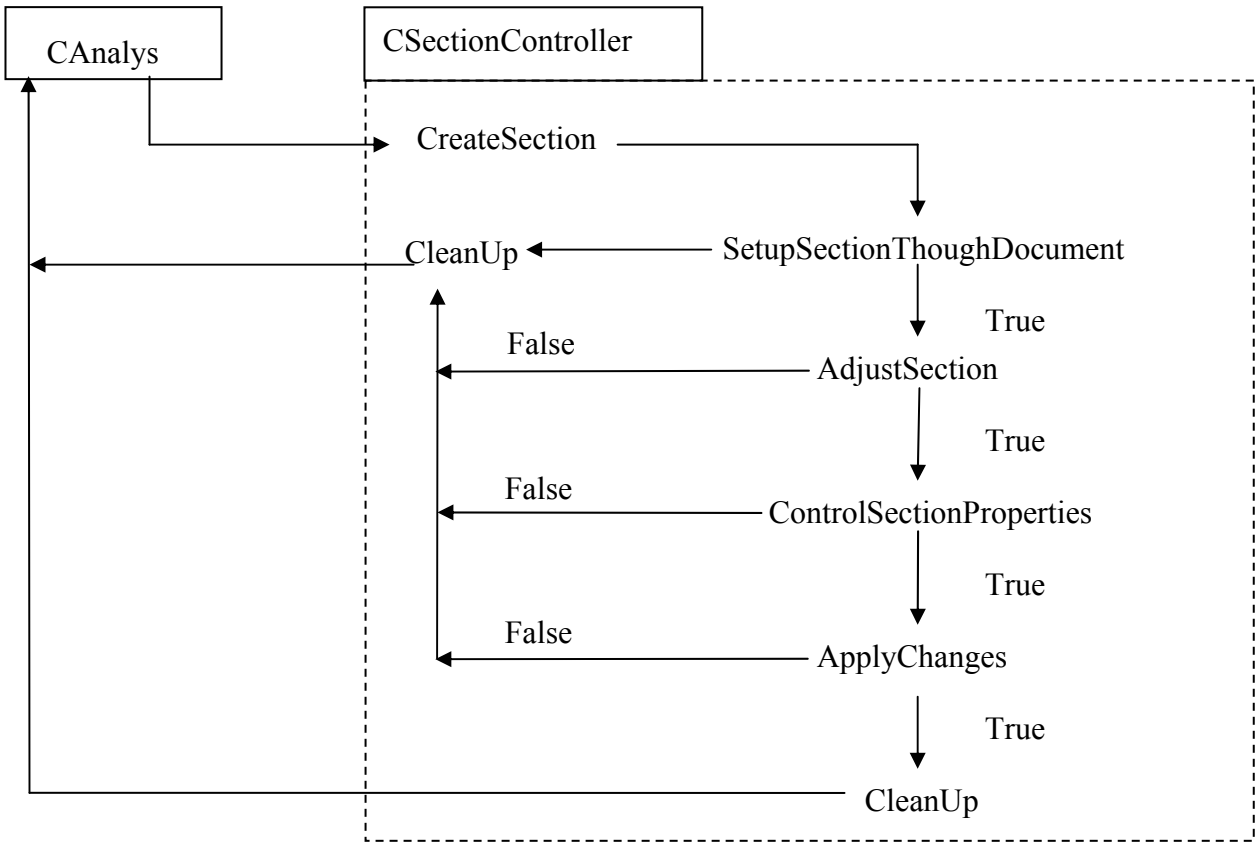
Denna modul skall ha den egenskapen att den skall kunna producera en vänster eller en höger kurva utifrån användarens val. Denna datamedlem talar om ifall det är en vänster eller höger kurva. Modulen skapar alltid en högerkurva, men om användaren har valt att skapa en vänster kurva speglas resultatet på slutet och på så viss skapas en vänster kurva.

4.3 Sektionsövergångsmodulen

4.3.1 Beskrivning av sektionsövergångsmodulen

Denna modul har i uppgift att konstruera en kopia av nuvarande spann, spegelvända dessa samt skjuta tillbaks spannen beroende på längden av sektionensövergång.
Sektionsövergångenslängd är i sin tur beroende av vilket typ av system kontaktledning är av.
Kopian och speglingen bildar sedan tillsammans med original spannen en sektionsövergång.

4.3.2 Flödesschema över sektionsövergångsmodulen



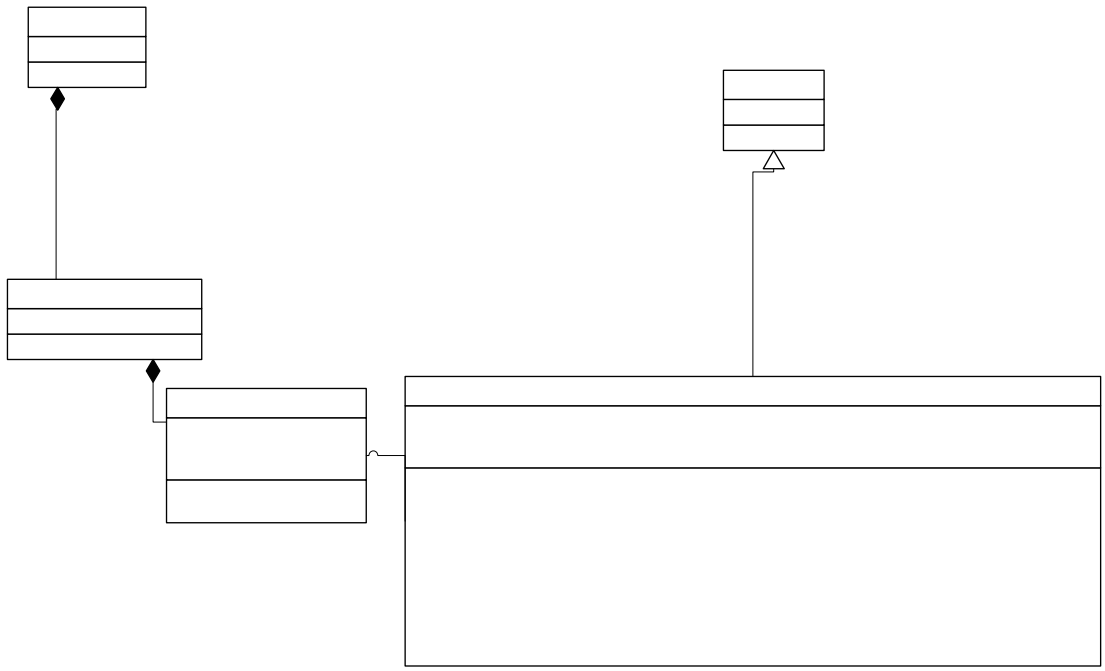
Figur 15. Visar ett flödesschema över sektionsövergångsmodulen.

CAnalys anropar *CreateSection* för att skapa en sektionsövergång. Det första som händer är att *SetupSectionThroughDocument* anropas, denna funktion tar de befintliga spannen och kopierar dessa, sedan spegelvänder funktioner dessa spann. Funktionen *AdjustSection* tar de skapade spannen och drar tillbaks de ett antal spann. Detta antal är beroende av vilken typ av system man bygger upp kontaktledningssystemet med. Om systemet är ett 5 spannssystem dras sektionsövergången tillbaks 3 spann och om det är ett 3 spannssystem dras sektionsövergången tillbaks 1 spann. I nästa steg, *ControlSectionProperties*, kontrollerar programmet om resultatet vid konstruktionen av sektionsövergången var korrekt. Om resultatet var godkänt adderas de nya spannen till det befintliga programmets spannarray. Det är utifrån denna array som programmet sedan konstruerar de noder som beskriver kontaktledningssystemet i en 3D-rymd.

Allra sist anropas en *CleanUp* funktion som låter applikationen rensa de data som genererats men som inte används, frigöra utrymme som senare kan användas av operativ systemet.

4.3.3 UML-diagram över sektionsövergångsmodulen

I Figur 16 kan man se ett UML-diagram över denna modul. Detta diagram visar vilka funktioner och datamedlemmar som implementeras i modulen.



Figur 16. Visar ett UML diagram över sektionsövergångshanteraren.

CreateSection:

Denna funktion anropas vid konstruktion av en sektionsövergång, och det är denna funktion som konstruerar sektionsövergången. Som indata accepterar den det aktiva dokument som används av programmet just nu. Det är i detta dokument som programmet lagrar all information om de olika spannen och därför kommer detta dokument att vara väldigt viktigt vid konstruktion av sektionsövergången.

Denna funktion genererar inget utdata i sig men däremot utför den en rad operationer som kommer att resultera i en sektionsövergång.

SetupSectionThroughDocument:

Denna funktion skall kopiera det nuvarande spannet. Kopieringen av spannet sker i omvänd ordning, d.v.s. 1,2,3 → 1,2,3,3,2,1. På detta vis slipper applikation klippa bort delar av start och slut av kontaktledningssystemet. För mer information om hur detta går till se avsnitt 3.4.

Som indata accepterar funktionen det aktiva dokument som programmet för tillfället använder. Detta dokument innehåller det data som krävs för att utföra kopieringen.

Utdata är sant eller falskt, beroende på hur operationen gick.

AdjustSection:

Denna funktion utför tillbakadragningen av spannet samt spegling efter att det har kopierats. Denna tillbakadragning sker i x-led och längden består av summan av längden på ett visst antal spann. Antalet spann är beroende på vilket system det handlar om, t.ex. om det är ett 5-spanns system dras den nya sektionen tillbaks 3 spannlängder, om det är ett 3 spann system dras den nya sektionen tillbaks 1 spannlängd. Speglingen sker sedan mot x-axeln i xz-planet.

Som indata accepterar funktionen det aktiva dokument som programmet arbetar med just nu. Inga utdata produceras direkt av denna funktion, utan opererar på data som påverkas av en hel process.

ControlSectionProperties:

Denna funktion har i uppgift att kontrollera att skapandet av sektionsövergången gick rätt till och att de data som producerades är redo att tas i bruk.

Funktionen består av två delfunktioner, *ControllspanLength* och C*ontrollCurveProperties,* där den senare funktionen kan ses i avsnitt 4.2.3. *ControllCurveProperties* används även på rakspår då man kan se rakspår som en kurva med oändlig radie.

Funktionen accepterar inga indata, och ger utdata i form av sant eller falskt beroende på om sektionsövergångens egenskaper uppfylls enligt (Lindberg,2004).

ControllSpannLength:

Denna funktion har i uppgift att kontrollera längder på vissa spann. För att sektionsövergången skall bli korrekt måste vissa spann ha samma längder. Funktionen plockar ut, beroende på om det är ett 3 eller 5 spanns sektionsövergång, de spann som skall kontrolleras och undersöker sedan deras längder. Visar det sig att spannlängden skiljer sig från varandra i något av fallen ger funktionen falskt utdata. Om spannlängderna stämde överens så genererar funktion sant utdata. Funktionen accepterar inga indata.

ApplyChanges:

När alla spann och deras egenskaper har verifierats och programmet kommit fram till att de är korrekta applicerar denna funktion spannen i systemet, så att programmet tar hänsyn till dessa när det blir dags att generera noderna till kontaktledningsystemet.

Som indata accepterar funktionen det aktiva dokument som programmet jobbar mot just nu. Som utdata producerar funktionen antigen sant eller falskt hurvida appliceringen av de nya spannen lyckades eller inte.

CleanUp:

Denna funktion har i uppgift att rensa upp i minnet bland de data som inte längre används efter skapandet av sektionsövergången. Om denna funktion inte anropas i slutet av sektionsövergångskonstruktionen skulle det resultera i att spannantalet skulle öka varje gång använder utfärdade kommandot ”Dynamisk” → ”LS-Dyna”, vilket i sin tur skulle leda till oönskade resultat. För mer information om hur allokering och deallokering av minnet till dessa spann, se avsnitt 3.4.

Funktionen accepterar det aktiva dokumentet som indata och ger ingen direkt utdata, men manipulerar mot en intern array i dokumentet och modulen.

CalculateStartPositionForSection:

Denna funktion används då noder skall genereras till spannen. Eftersom sektionsövergången syftar till att två trådar i kontaktledningssystem skall mötas och sedan slutligen övergå till en tråd. Det är därför viktigt att man vid något tillfälle slutar att generera noder för den första tråden och sedan börjar generera noder för den andra tråden. Den andra tråden kommer då att vara tillbaka dragen 3 eller 1 spann och det är det som denna funktion gör, beräknar startpositionen för tråd nummer 2.

Som indata vill funktionen ha den nuvarande positionen samt det nuvarande aktiva dokumentet. Ur det nuvarande dokumentet kan funktionen få viktig data angående den nya positionen. Den nuvarande positionsparametern anges som en referensparameter, vilket kommer att leda till att resultatet lagras i den variabel som skickades med denna funktion. Därför genereras ingen utdata av denna funktion.

m_CurrentTrack:

Denna datamedlem är en datastruktur som beskriver den nuvarande kontaktledningen och kan ses som den tabell som nämns om i avsnitt 3.4.

m_SystemType:

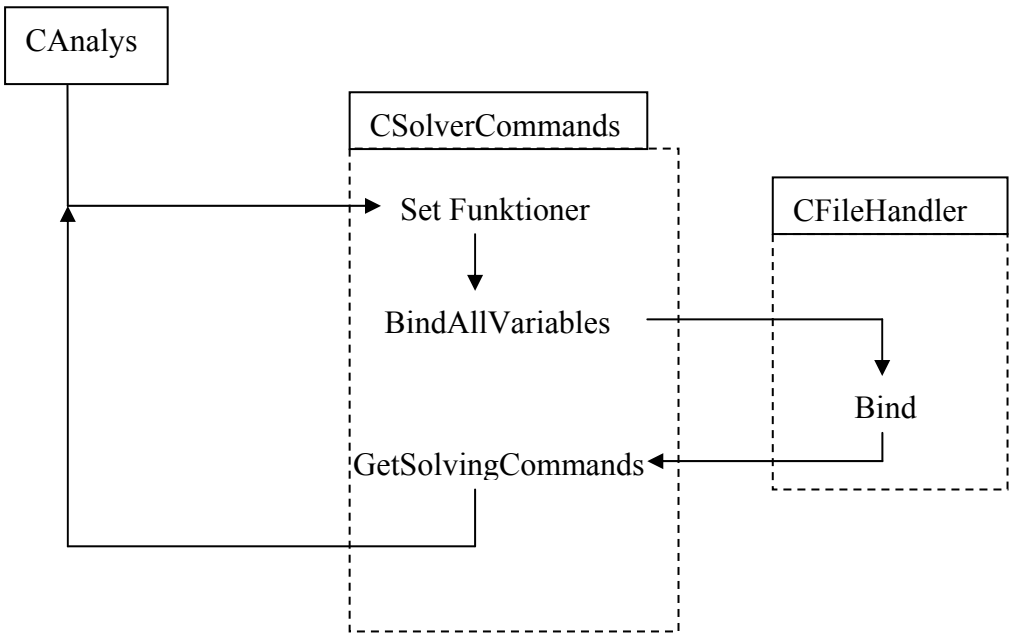
Denna datamedlem identifierar nuvarande kontaktledningssystem huruvida det är ett 3 spannssystem eller ett 5 spannsystem.

4.4 Beräkningskommandomodulen

4.4.1 Beskrivning av beräkningskommandomodulen

Denna modul har i uppgift att leverera de beräkningskommandon som ska användas vid simuleringen av modellen. Modulen är utvecklad så att det ska vara väldigt enkelt att addera en variabel som skall vara dynamisk, variera från körning till körning.

4.4.2 Flödesschema över beräkningskommandomodulen



Figur 17. Visar ett flödesschema över beräkningskommandomodulen

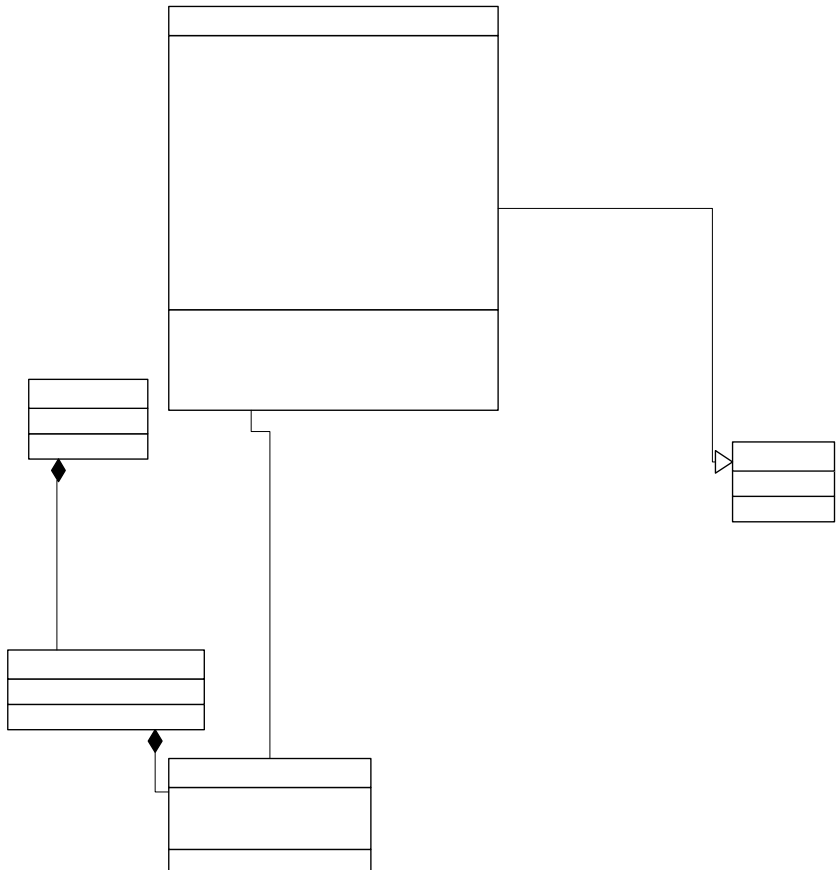
CAnalys anropar *Set* funktioner för alla variabler som är dynamiska. Detta innebär att man tilldelar de variabler som skall vara dynamiska sina värden via dessa funktioner. Vill man ha en ny variabel som varierar mellan olika körningar måste man skapa den i klassen och sedan ge den en set funktion.

När Samtliga variabler har fått sina värden anropas funktionen *BindAllVariables* som binder samman en variabels text namn och dess värde. Denna bindning går sedan via en

filhanterare, som skriver ner värdet i filen på den plats som den tillhör. För mer information om hur detta går till se avsnitt 4.4.3.

Sist i flödet anropas *GetSolvingCommands* och på detta sätt får programmet tillgång till de kommandon som modulen genererar. Dessa kommandon kan nu skrivas ner till filen som genereras.

4.4.3 UML diagram över beräkningskommandomodulen



Figur 18. Visar ett UML-diagram över beräkningskommandomodulen.

*Set**:

Varje variabel som skall varieras mellan olika körningar har en egen Set funktion, dessa har utelämnats i ovan diagram för enkelhetens skull. Dessa funktioner fungerar på så vis att de accepterar en variabel av samma typ som den variabeln som skall varieras är av. Sedan tilldelar man den varierande variabeln värdet av dessa indata.

BindAllVariables:

Denna funktion binder samtliga variabler som skall varieras mellan körningar med ett textbaserat namn mot ett värde. När man vill skapa en variabel som skall kunna varieras under olika körningar måste man definiera ett textnamn för denna variabel. Sedan om detta namn förekommer i en textsträng byts det namnet ut mot variabelvärdet. För att detta skall fungera måste man göra en bindning mellan textnamnet och värdet via en filhanterare. Denna funktion har i uppgift att göra detta.

Funktionen accepterar som indata en filhanterare. Filhanteraren har öppnat en fil som är redo att ta emot text. När filhanteraren tar emot text läser den igenom denna textsträng efter ord som är bundna till ett värde, om det förekommer ett ord som filhanteraren känner igen skriver den ner det värde som detta ord pekar på innan den skickar texten vidare till filen.

När *BindAllVariables* anropas binder funktionen alla fördefinierade variabelnamn till ett värde via filhanteraren, sedan när programmet skriver samtliga kommandon till en fil via filhanteraren så byts de reserverade orden ut mot värden.

GetSolvingCommands:

Denna funktion ger som utdata en pekare till en sträng som motsvarar alla de kommandon som skall skrivas till utdatafilen. Det är i denna flerraders sträng som de reserverade orden förekommer och när skrivning till utdatafilen sker ersätts dessa med variabelvärdena.

Funktionen accepterar inga indata.

CreatePredefinedVariables:

Denna funktion skapar de textnamn som alla variabler får innan de ska ersättas med värden. Om man vill skapa en variabel som skall varieras mellan olika körningar måste man i denna funktion ge denna variabel ett namn, på så vis kan man senare binda detta namn till ett variabelvärde.

4.5 Singleton

Singleton är ett av de mest använda designpattern. Detta designpattern instansierar och hanterar en klass på så vis att det endast finns en enda instans av denna. Klassen har också i uppgift att låta användaren få access till denna enda instans via en statisk pekare. Vidare lägger man ansvaret på klassen att hantera denna enda instans, vilket ur en objektorienterad synvinkel känns mest korrekt.

Med detta design pattern globaliserar man också modulen som implementeras med Singleton.

(Kalev, Danny) visar en bra implementation av ett Singleton

4.5.1 Fördelar med Singleton

Största fördelen med ett Singleton är att man får en accesspunkt till ett objekt, som dessutom är global. Man ska dock tänka sig noga för när man implementerar Singelton. Man kan se en klass som implementeras med Singleton som en form av en hanterare, t.ex en filhanterare.

Ett bra exempel på ett Singleton är t.ex. en applikation som använder musen. Musen instansieras från en klass, men man behöver bara en instans av denna mus. Därför skulle man då kunna implementera denna mus med hjälp av Singleton design pattern.

4.5.2 Nackdelar med Singleton

Singleton är inte alltid så simpla som de kan verka, (Tan, Robin 2002) visar att det kan uppstå situationer som kan bli komplicerade att implementera på ett korrekt vis om man inte noggrant tänker igenom designen. Ett andra par exempel på problem som kan uppstå presenteras i (Nakamura, Jun 2005)

Ett dåligt exempel på ett Singleton är om man har en klass som implementerar en vektor av någon datatyp och sedan implementerar man denna vektor med en Singleton basklass. Då begränsar man applikationen genom att låta den endast skapa en vektor av den datatypen. Sannolikheten att man vill ha flera arrayer av denna datatyp är väldigt stor, men om ovan implementation sker så kan inte detta hända.

4.5.3 Varför Singleton?

De olika modulerna valdes att implementeras med ett Singleton designpattern för att dessa moduler kunde betraktas som hanterare. Sektionsövergångsmodulen accepterar som indata ett kontaktledningssystem och manipulerar sedan detta så att det uppstår en sektionsövergång. Kurvmodulen accepterar också ett kontaktledningssystem och manipulerar detta så att det bildas en kurva. Kommandomodulen accepterar variabel värden som indata och producerar en sträng som motsvarar de kommandon som skall användas vid beräkningar. Om man tittar på detta ur en designmässig synvinkel märker man att dessa moduler inte borde instansieras flera gånger. Dessa moduler är också ganska globala, d.v.s. programmet behöver access till deras instanser på flera ställen, vilket också gynnar Singleton konceptet. Detta skulle man kunna lösa med t.ex en vanlig pekare som är global. Men denna lösning hindrar inte programmeraren att instansiera en lokal variabel av samma klass, vilket medför att klassen inte kommer att användas på rätt sätt.

5 Systemverifikation

Systemverifikationen sker i två steg. I det första steget, som pågick ända sedan starten av detta arbete, testas modulerna. Detta sker kontinuerligt efterhand som dessa implementeras och modifieras. Det är viktigt att man hela tiden kontrollerar modulerna vid större händelser så att dessa händelser inte genererar nya buggar som kan ledda till större och svårupptäckta fel.

I det andra steget är det ett jämförande test som utförs mellan två olika modeller. Detta test skall utföras enligt (SEK 2002) men det finns vissa undantag. De två olika modellerna är en 2D-modell samt en 3D-modell. 2D-modellen görs av det befintliga programmet och 3D-modellen produceras av programmet i och med detta arbete.

5.1 Modulerna

Varje modul testades kontinuerligt under utvecklingen. Detta skedde med jämna mellan rum då någon ny funktion adderats till de olika modulerna.

Testerna utfördes på så vis att indata matades in på ingångarna och sedan kontrollerades mot utdata. Om utdata inte stämde överens med det förväntade resultatet, stegades inmatningen igenom steg för steg för att kontrollera var det uppstod fel.

Under utvecklingen valdes också att skriva de viktigaste data som producerades till en fil för att kontinuerligt kontrollera att dessa data var korrekt.

När varje modul sedan var testad och gav korrekt utdata på given indata, kopplades dessa i sin tur en och en in i systemet för att kontrolleras ifall de kunde samarbeta med den redan befintliga applikationen. När modulerna visade ett godkänt sådant beteende ansågs dessa vara godkända.

5.2 Benchmarktest

Benchmarktestet utfördes enligt (SEK 2002) avsnitt 10 och 11. Avsnitt 10 hänvisar till verkliga simuleringar, men enligt avgränsningar skall inte detta arbete gå in på verkliga simuleringar, så dessa mätdata har hämtat ur simuleringar från den redan existerande 2D-varianten av programmet.

Figur 19 visar den de resultat som måste erhållas för att testet skall klassas som godkänt.

	Range of results	
speed [km/h]	250	300
F_M [N]	110 to 120	110 to 120
σ [N]	26 to 31	32 to 40
Statistical maximum of contact force [N]	190 to 210	210 to 230
Statistical minimum of contact force [N]	20 to 40	- 5 to 20
Actual maximum of contact force [N]	175 to 210	190 to 225
Actual minimum of contact force [N]	50 to 75	30 to 55
Maximum uplift at support [mm]	48 to 55	55 to 65
Percentage of loss of contact [%]	0	0
NOTE The values in the table are based on results from five independent simulation methods. These methods have been checked with results from line tests.		

Figur 19. Visar noggrannheten som måste erhållas för att simuleringen skall bli godkänd

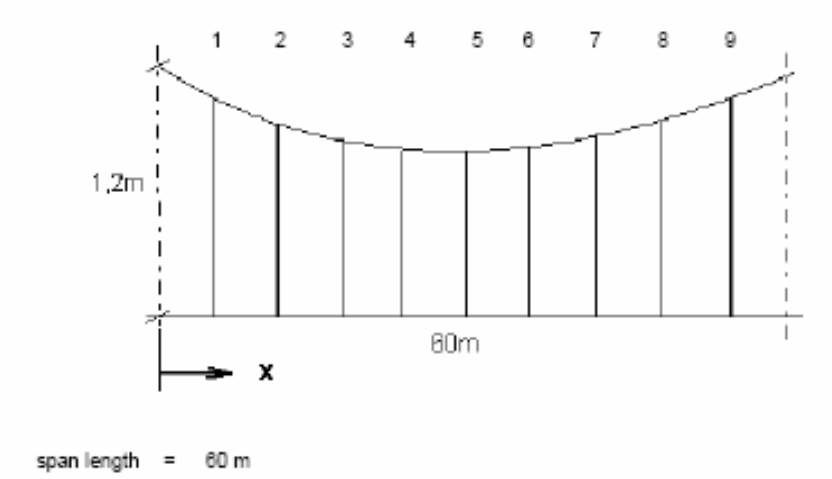
Man kan se i (SEK 2002), avsnitt 10.2.1 att det är tillåtet att modifiera parametrar som fjäderegenskaper, friktionsegenskaper och materialegenskaper i modellen för att uppnå korrekt data, medan det inte är tillåtet att ändra direkta data på strömavtagaren.

Man kan också se att i avsnitt 10.2.2 i (SEK 2002) är det också tillåtet att modifiera en del data över kontaktledningen, medan det inte är tillåtet att ändra data så som antalet kontaktledningar bärlinor och bärtrådar.

Övriga parametrar där variation godtas kan man läsa om i avsnitt 10.2.3 i (SEK 2002).

5.3 Uppbyggnad av modellerna

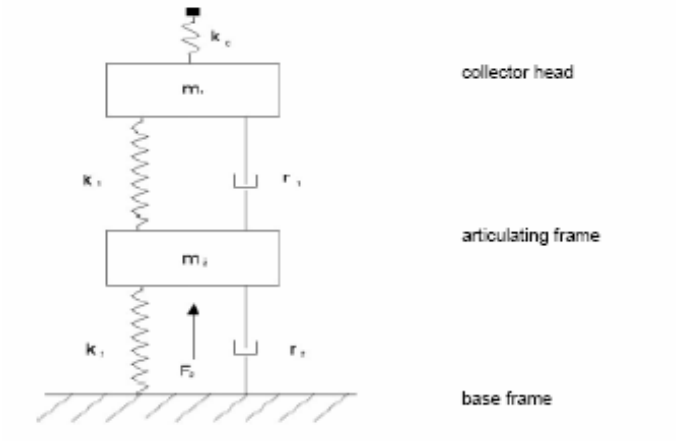
De båda modellerna utgår ifrån en gemensam samling data. Dessa data härstammar från en referensmodell där kontaktledningssystemet byggs upp utifrån bestämd data som kan ses i Figur 20. Detta kan man läsa mer om i avsnitt 11.3 i (SEK 2002). Det är också viktigt att följa en del bestämmelser när man bygger upp strömavtagare modellen, dessa värden kan ses i Figur 21 och Figur 22 visar principen för hur strömavtagaren är uppbyggd.



Figur 20. Visar hur spannen skall byggas upp.

	Effective dynamic mass kg	Stiffness N/m	Damping Ns/m
Contact spring	-	k_c = 50 000	-
Collector head	m_1 = 7,2	k_1 = 4 200	r_1 = 10
Articulation frame	m_2 = 15	k_2 = 50	r_2 = 90

Figur 21. Visar en tabell över strömavtagardata.



Figur 22. Visar den principiella uppbyggnaden av strömavtagaren

Simuleringen gjordes på båda modellerna med avseende på:

- hastigheten, som i båda fallen simulerades på både 250 km/h och 300 km/h

- endast en strömavtagare användes
- en filterfrekvens på 20 Hz användes vid simuleringen

Kontaktledningssystemet byggdes upp med 10 stycken identiska spann, där spann 5 och spann 6 var de spann som man ville analysera krafterna på, där samtliga spann är 60 meter långa. Detta medför att mellan 240m och 360m hamnar de punkter som är intressanta att titta på.

För mer information om referensmodellen se avsnitt 11 i (SEK 2002).

5.3.1 2D-modellen

Denna modell byggdes upp med 10 spann enligt avsnitt 5.3.Utdatafilen, den fil som sedan används som indata till main.exe, genererades genom att aktivera det dynamiska läget och trycka ”Dynamisk” → ”Ansys 2D ny metod”.

Filen som producerades användes sedan som indata till ansys som i sin tur producerade en utdatafil som formaterades sedan med hjälp av ett program som heter main.exe.

5.3.2 3D-modellen

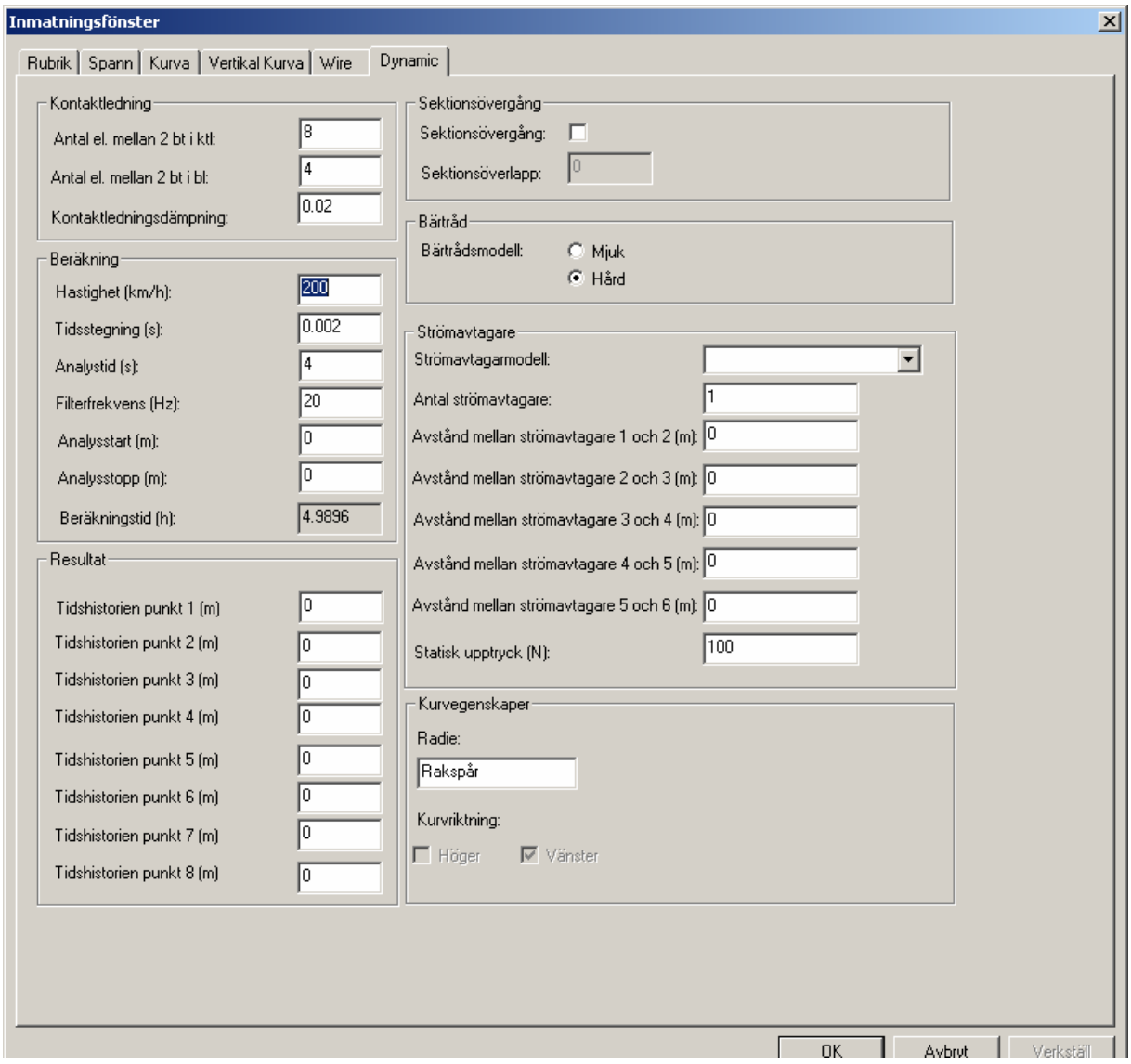
Denna modell byggdes upp med 10 spann enligt avsnitt 5.3. Utdatafilen, den fil som sedan används som indata till LS-dyna, genererades genom att aktivera det dynamiska läget och trycka ”Dynamisk” → ”LS-dyna”.

Filen som producerades användes sedan i LS-Dyna för att importera och bygga upp den modell i 3D som avsnitt 5.3 beskrev. Slutligen utfördes simuleringen i LS-Dyna.

6 Resultat

6.1 Programmet

De delar som skulle implementeras i och med detta arbete var möjligheter att skapa sektionsövergångar och kurvor. Detta avspeglas i programmet under den dynamiska fliken i det dynamiska läget av programmet, se Figur 23.



Figur 23 visar den dynamiska fliken, här kan man ändra de parametrar som skall variera mellan olika körningar i de dynamiska beräkningarna.

Från denna flik kan man mata in om man vill ha en sektionsövergång eller om man vill ha en kurva i sin modell.

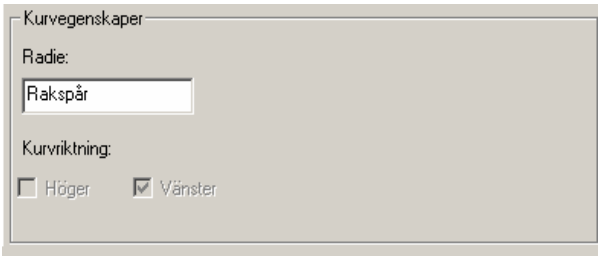
6.1.1 Inmatning av spann

Inmatning av spann sker under menyalternativet ”Indata” → ”ändra” och sedan väljer man ”spann” fliken. Härifrån kan man sedan mata in de spann man vill ha och deras värde, så som spannlängder, trådläge, systemhöjd, tillsatsrör och mycket mer. Denna del av

programmet är inget nytt i och med detta arbete, men denna del spelar en central roll i byggande av modellen. Figur 10 visar ett exempel på detta.

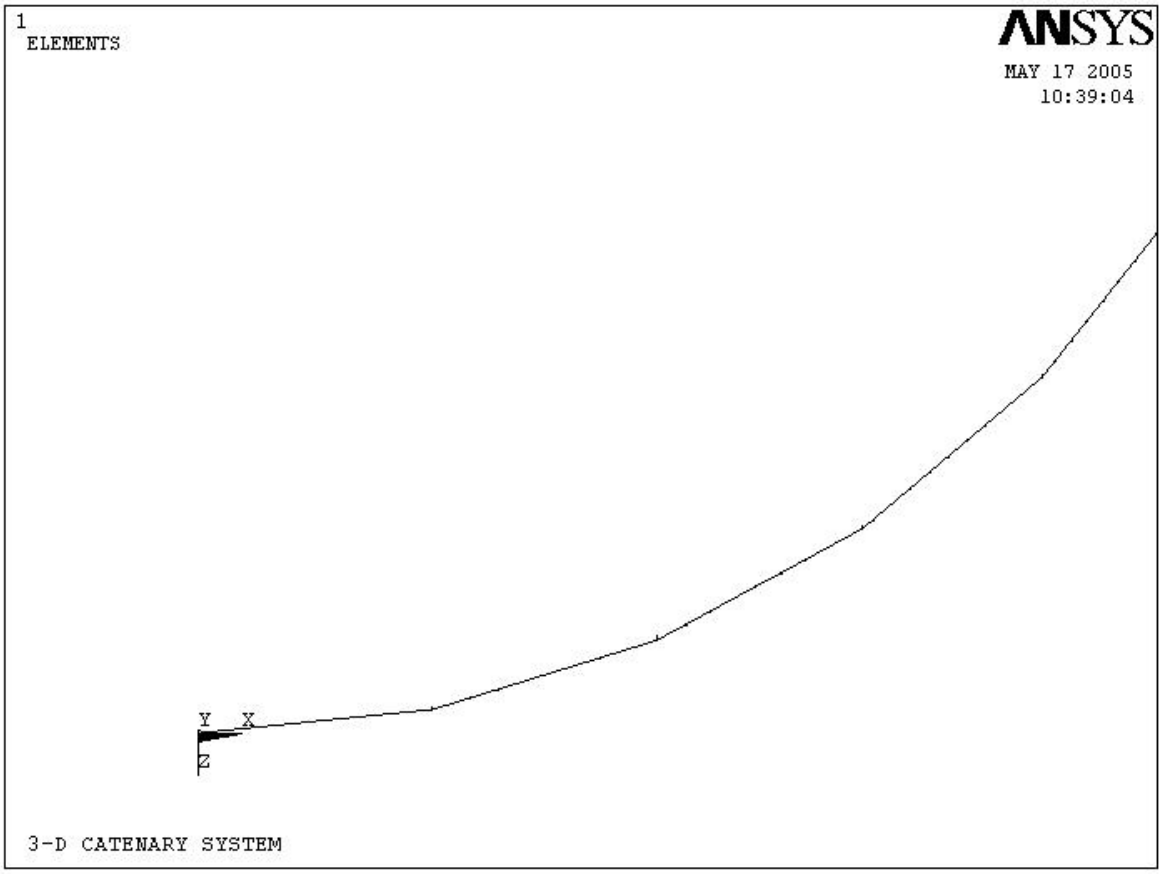
6.1.2 Skapandet av en kurva

Skapandet av en kurva går till på så vis att man aktiverar det dynamiska läget i programmet och sedan klickar man på ”Indata” → ”ändra” och sen på ”Dynamic” fliken, se Figur 23. Längst ner i det högra hörnet kan man mata in vilken radie man vill ha. Om radien är 0 eller en bokstav antar programmet automatisk att man vill ha ett rakspår. Skulle radien anta ett heltalsvärde kan man också välja om man vill ha en höger kurva eller en vänster kurva. Skulle värdet inte vara ett giltigt heltalsvärde ignoreras kurvans riktning och ingen kurva skapas överhuvudtaget. Figur 24 visar ett exempel på detta.



Figur 24 visar den nya biten för inmatning av kurva. Denna del hör till fliken Dynamic

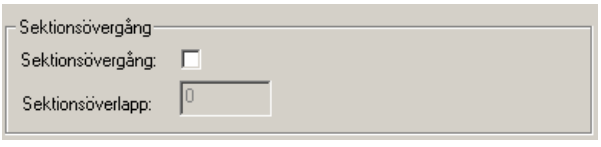
Denna del i programmet är helt ny och den del som heter ”Kurvegenskaper” fanns inte i det tidigare programmet. Resultatet av kurvgenereringen kan ses i Figur 25.



Figur 25 visar en kurva uppifrån, så som modellen såg ut i Ansys/LS-Dyna

6.1.3 Skapandet av en sektionsövergång

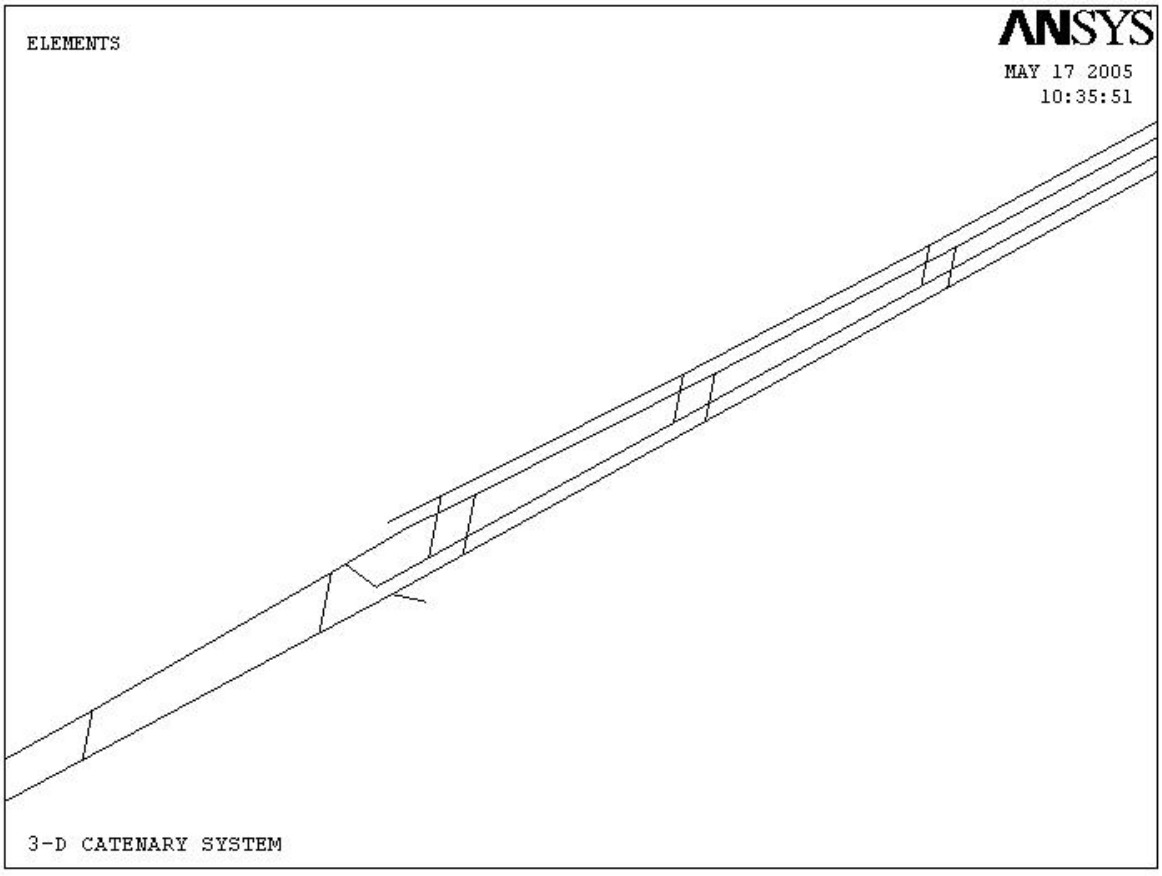
Figur 26 visar hur man väljer att programmet skall skapa en sektionsövergång. Antalet metrar som kan matas in i fältet påverkar inte skapandet av sektionsövergången då dess längd är beroende av kontaktledningssystemet.



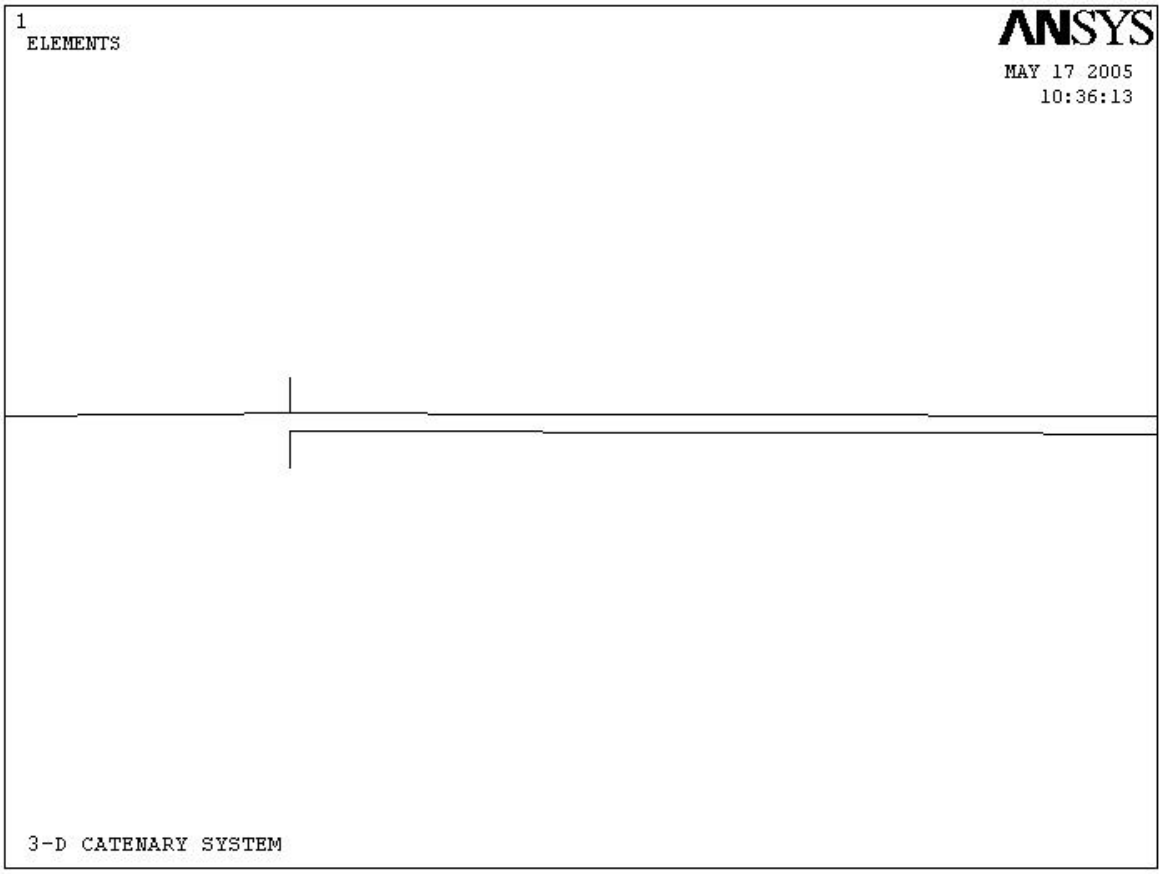
Figur 26 visar hur val av sektionsövergång sker.

Kontaktledningssystemet kan man ange under fliken ”Rubrik” och det finns ett visst antal förbestämda system, och det som avgör hur stor sektionsövergången blir är antalet spann som det nuvarande systemet accepterar i sektionsövergången.

Innan detta arbete fanns dessa inmatningsrutor redan på plats och hänvisar då till 2D varianten av programmet. Programmet läser då av inmatningsrutan och gör sedan sektionsövergången i det antal metrar som har angetts i rutan. Resultatet av skapandet av en sektionsövergång kan ses i Figur 27 och i Figur 28.



Figur 27 visar en sektionsövergång, så som modellen såg ut i Ansys/LS-Dyna



Figur 28 visar en sektionsövergång uppifrån, så som modellen såg ut i Ansys/LS-Dyna

6.1.4 Beräkningskommandon

Beräkningskommandomodulen genererar kommandon som skall användas vid simulering med LS-Dyna. Dessa kommandon hamnar sist i filen vid valet ”Dynamisk” → ”LS-Dyna” från menyn. Dessa kommandon är ett måste om man skall köra 3D modellen, dessa går därför inte att välja.

De parametrar som ingår i denna simulerings återfinns under fliken ”Dynamic” under ”Indata” → ”Ändra”. I dagsläget är det endast de viktigaste parametrarna som överförs till LS-Dyna filen, men modulen i sig är konstruerad så att det är väldigt enkelt att lägga till fler variabler som skall vara dynamiska, se avsnitt 4.4.3 för mer information om hur detta går till.

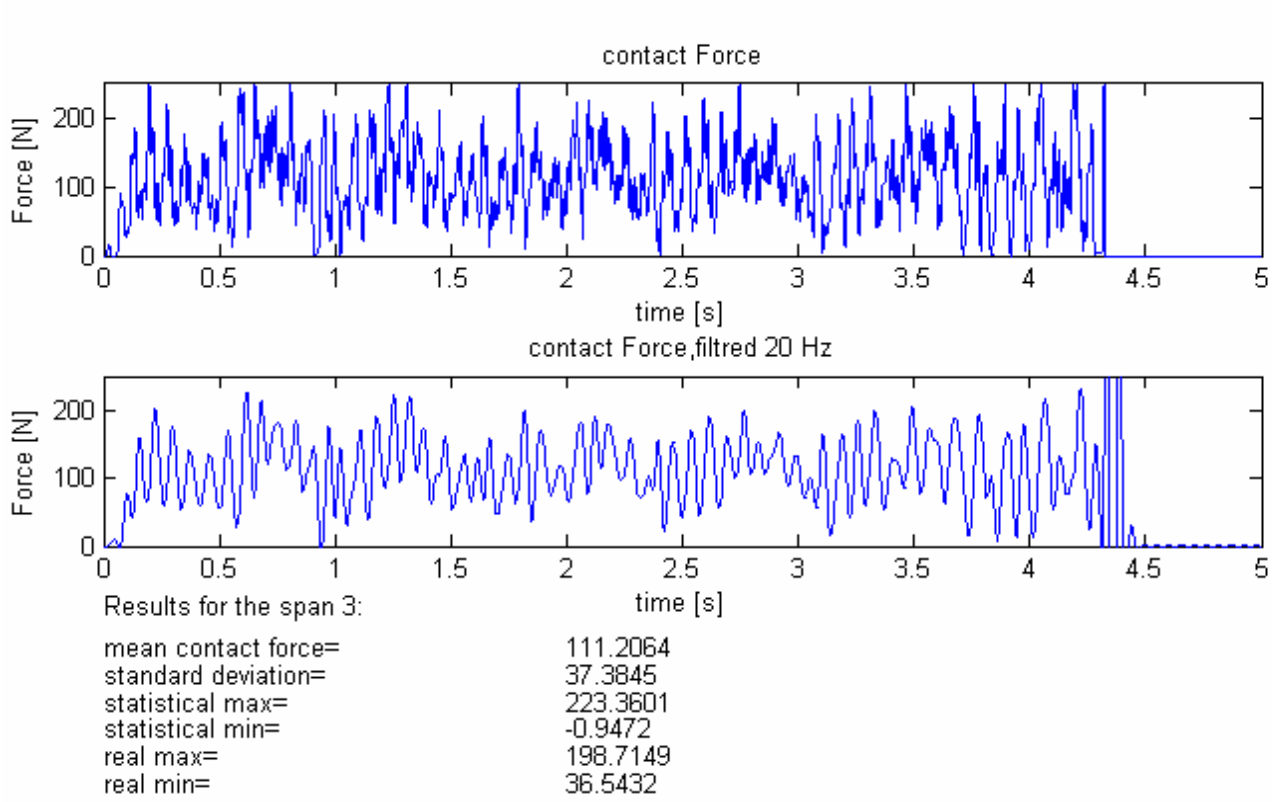
6.2 Simulering

Simuleringarna utfördes först på en 2D-modell som redan fanns implementerad i programmet, och sedan på den 3D-modell som detta arbete implementerade.

2D-simuleringen utfördes på ca 8 timmar medan 3D-simuleringen utfördes på ca 12 timmar. 3D-simuleringen utfördes på en äldre dator och därför tog det längre tid. På grund av den långa simuleringstiden så utfördes simulering endast på 120 N som statisk upptryckskraft och 300 km/h som hastighet. Enligt (SEK 2002) ska man göra två test, ett test med 120 N och 300 km/h och ett test med 120 N och 250 km/h.

De grafer som presenteras nedan visar med vilken kraft som strömavtagare har haft mot kraftledningen under simuleringen. På x-axeln kan man se tiden som utgör själva simuleringen och på y-axeln kan man se kraften som strömavtagaren trycker med mot kontaktledningen. Den övre delen av graferna är rådata som inte blivit filtrerat, medan i den undre delen av graferna har blivit filtrerad med 20 Hz.

6.2.1 2D-simulering

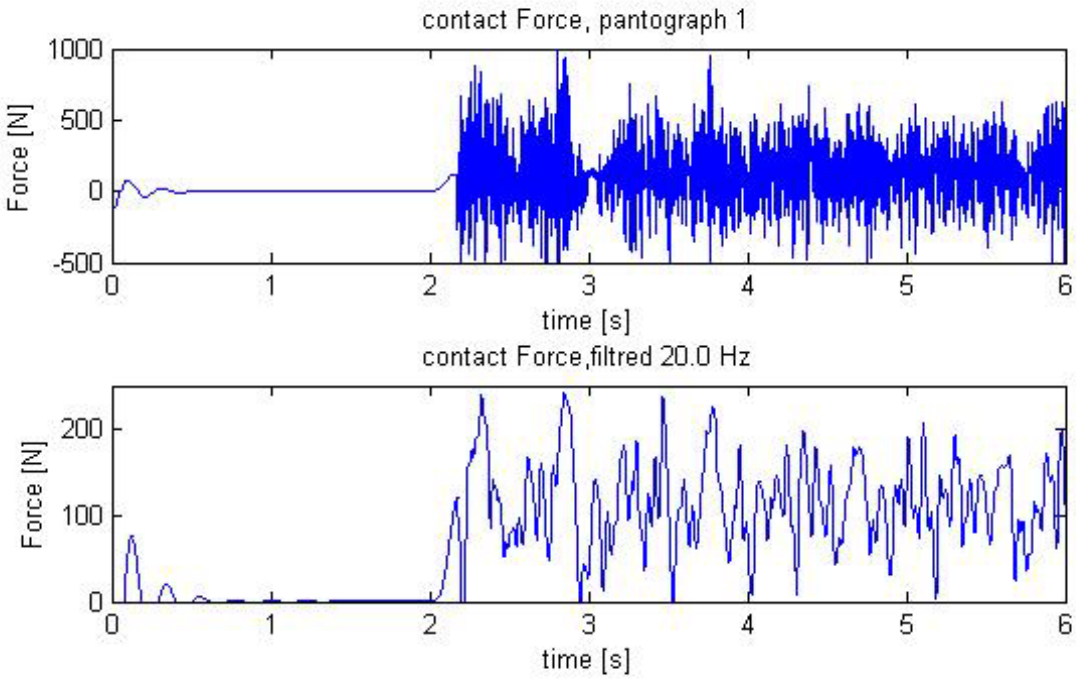


Figur 29. Visar resultatet av 2D simuleringen.

I Figur 29 kan man se en graf över resultatet av 2D-simuleringen. Under själva grafen finns de värden som man är intresserad av att jämföra med. Om man jämför med Figur 19 som är hämtad ur (SEK 2002) ser man att värdena i grafen stämmer överens med de värden som

godkänner simuleringen. Denna simulering utfördes med Ansys och de resultatfiler som producerades kördes i ett program som heter main.exe som producerade en fil som kunde köras i beräkningsprogrammet MatLab, som producerade graferna.

6.2.2 3D-simulering



Analys sektion between 240 and 360 meter (3.5 and 5.2 seconds)

Mean Contact Force [N]=	114.9694
Standard Deviation[N]=	47.3816
Statistical Max (MCF+3*SD)[N]=	257.1144
Statistical Min (MCF-3*SD)[N]=	-27.1755
Real Max[N]=	237.4238
Real Min[N]=	-15.7917

Figur 30. Visar resultatet av 3D-simuleringen.

Diagrammet i Figur 30 visar resultatet av 3D-simuleringen. Under själva grafen presenteras de värde som är intressanta att jämföra mot (SEK 2002). Det visar sig då att denna simulering inte blev godkänd. De värden som presenteras under grafen i Figur 30 är inte inom det +/- 20 % intervall som krävs för att simuleringen skall bli godkänd, enligt (SEK 2002). Tabellen som presenteras i (SEK 2002) kan ses i Figur 19. Man kan se att

standardavvikelsen är alldeles för hög, och detta påverkar i sin tur *Statistical Max* och *Statistical Min* då dessa är medelkontaktkraften adderat/subtraherat med 3 standardavvikelser. Medelkontaktkraften i sin tur är inom intervallet för en godkänd simulering. Verkliga max och min är långt utanför intervallet för en godkänd simulering.

De krafter som uppstår i början av diagrammet är resultat av att modellen inte helt har stabiliserat sig i Ansys/LS-Dyna. Efter denna stabilisering går det fram till och med 2 sekunder innan strömavtagaren får någon hastighet, detta göra man för att vara helt säker på att modellen har stabiliserat sig innan man börjar simulera. Efter upprepande simuleringar visar det sig att det inte endast var ovan simulering som inte var godkänd, utan att det var något som inte blev som det skulle. Detta kan bero på det sätt som man räknade fram kraften med. det visade sig nämligen att det inte bara var att köra resultat filerna med main.exe för att producera ut grafen. De utdata som var resultatet av simuleringen var tvungen att efterbehandlas. För att få ut grafen i Figur 30 var man tvungen att beräkna den kraft som trycker mot kontaktledningen. För att få ut kraften F som trycker mot kontaktledningen kan man använda formeln:

$$F_k = m*a + F_{dämpare} + F_{fjäder} - m*g$$

där F_k är den kraft som trycker mot kraftledningen, och m är den totala massan på strömavtagaren. $F_{dämpare}$ och $F_{fjäder}$ är den totala kraften som fjädern och dämparen har och a är den acceleration som sker i y led för massa 1 i Figur 22. Vid de första testerna verkade det som om LS-Dyna kunde plocka ut accelerationen och $F_{dämpare}$ och $F_{fjäder}$ för strömavtagaren, och massan visste vi att den var konstant 7,2 kg och att g är också en konstant på 9,82 m/s^2. Men sedan visade det sig att så inte var fallet, av någon anledning klarade inte LS-Dyna av att beräkna de korrekta värdena på dämparna och fjädrarna. LS-Dyna kunde däremot ge oss en korrekt förskjutning av massorna i Figur 22 som man sedan kan använda för att beräkna $F_{dämpare}$ och $F_{fjäder}$ genom att låta

$$F_{fjäder} = ((massa_2 uyl - massa_1 uyl) * \frac{k1}{2} + (massa_2 uyr - massa_1 uyr) * \frac{k1}{2})$$

Här är $massa_2$uyl och $massa_1$uyl den förskjutning som sker för massorna i Figur 22 i y-led för den vänstra sidan, medan $massa_2$uyr och $massa_2$uyr är samma sak för den högra sidan.

K1 är den styvhetskonstant som nämns i tabell A.1 i (SEK 2002, sid 14).

Med hjälp av hastigheten för dessa kunde man sedan beräkna

$$F_{dämpare} = ((massa_2 vyl - massa_1 vyl) * \frac{r1}{2} + (massa_2 vyr - massa_1 vyr) * \frac{r1}{2})$$

Här är $mass_2$vyl, $massa_1$vyl , $massa_2$vyr och $massa_1$vyr hastigheter för dämparna och fjädrarna i y led.

R1 är den dämpningskonstans som nämns i tabell A.1 i (SEK 2002, sid 14).

Med hjälp av detta kan man nu beräkna den kraft som strömavtagaren trycker med på kontaktledningen i en funktion av tiden och sedan plotta detta med hjälp av Matlab.

7 Diskussion

Målet med detta arbete var att utveckla moduler som kunde skapa en utdatafil som klarades av att simuleras i Ansys/LS-Dyna. Dessa moduler skulle sedan vara kapabla att kopplas till ett redan befintligt program för bärtrådsberäkningar. Simuleringerna skulle dessutom klara kraven ur EN 50 318. Det visade sig att det resultat som presenterades inte blev godkänt enligt EN 50 318. En anledning till att resultatet inte blev godkänt kan ha att göra med hur kolet på strömavtagaren byggdes upp. Kolet på strömavtagaren representerades i Ansys/LS-Dyna med små skal. Dessa mindre skal användes sedan för att bygga upp ett större skal som skulle motsvara kolet. I simuleringarna ovan så användes fyra mindre skall för att bygga det större skalet, det har visat sig senare att det borde ha varit minst 32 mindre skal.

Ett annat mål var att programmet skulle klara av att generera både kurvor och sektionsövergångar i 3D. Det har visats att det nu är fullt möjligt att göra detta i och med implementation av tre moduler som beskrivs i detta arbete.

De personliga kunskaperna inom programmering har också utökats, speciellt i det avseendet att kunna följa redan existerande kod. De matematiska kunskaperna har också utvecklats.

7.1 Vad gjordes bra?

Det som gjordes bra med programmets nya moduler var deras sätt att kommunicera med det redan befintliga programmet då dessa drog stor nytta av redan existerande rutiner. Det befintliga programmets källkod kunde ha varit bättre dokumenterat, men trots det så lyckas modulerna dra nytta av redan existerande funktioner bl.a. vid genering av noder.

7.2 Vad gjordes dåligt?

Det som gjordes dåligt, eller iallafall mindre bra, vid utvecklingen av dessa applikationsmoduler var simuleringsdelen. Denna del kunde ha utfört tidigare och ägnats mer tid till detta då det visade sig att det tog längre tid än räknat och det uppstod en hel del svårigheter. Tiden som en simulering tog var beräknad till mellan åtta och tolv timmar. 2D-varianten tog endast åtta timmar. 3D-simuleringarna gjordes på en sämre dator än 2D-simuleringarna, som tog tolv timmar. Dessa långa simuleringstider resulterade i att det var svårt att testa olika parametrar och det var också svårt att felsöka vid simuleringar som gick snett.

8 Felkällor

Vid simulering användes inte verkliga mätdata som man borde göra enligt (SEK 2002), utan den existerande 2D-modellen fick bli de data som man jämförde med. Ett problem som uppstod var att man var tvungen att göra en del beräkningar för hand efter att utdata presenterats i 3D-simuleringen, dessa beräkningar kan lätt gå fel, då de inte gjordes av en maskin, därför bör man utveckla någon form av program som tar hand om dessa efterbehandlingar som 3D-modellen kräver.

En annan felkälla är den mänskliga faktorn. Trots att alla beräkningar kontrollerades flera gånger så att alla värden stämde överens vid simuleringen, så är det väldigt enkelt att ett litet fel kan smyga sig in. Då en simulering tar mellan åtta och tolv timmar så försvårar detta felsökningsarbetet.

9 Framtida arbete

Eftersom detta arbete ingår i ett större projekt och är en liten del finns det mycket kvar att förbättra.

Programmet stödjer i och med detta arbete konstruktion av sektionsövergångar samt konstruktion av kurvor i 3D, men tester visar att det inte riktig fungerar att konstruera sektionsövergångar i kurvor. Detta är nog orsaken av en eller flera buggar som inte har blivit upptäckta under tiden då systemverfikationen genomfördes.

Kurvmodulen genererade inte heller helt korrekt resultat. Detta är någonting som man också kan förbättra I framtiden. I dagsläget genereras kurvor med endast en radie, i verkligheten har man kanske flera varierande radier i en kurva. Rälsförhöjning och korglutning är också något som programmet inte tar hänsyn till, man vill nämligen i kurvor luta själva tåget för att kunna dra nytta av högre hastigheter i kurvan. Dessa tre saker om kurvorna är något som inte programmet heller tar hänsyn till, detta för att det är ganska omfattande saker att implementera och har därför avgränsats i och med detta arbete.

Strömavtagaren som genereras i dagsläget är väldigt simpel och man kan bara generera en strömavtagare. I framtiden kan det vara intressant att kunna simulera med flera strömavtagare och mer avancerade modeller.

Vid simulering av 3D-modellen visade det sig att det inte var så enkelt att få ut graferna som presenteras i avsnitt 6.2.2. I 2D-modellen kan man ta det utdata som produceras och köra main.exe för att efterbehandla data och få ut de grafer man önskar. Ett sådant program borde utvecklas samt anpassas även för 3D-varianten, då det lätt kan uppstå fel om man för egen hand skall gå in och klippa - klistra ihop de grafer man önskar.

10 Tack till

- Lars Drugge för hans tålmodiga förklarande av strömavtagarmodeller och kontaktledningsdynamik
- Sven Lundbäck för all hjälp med kontaktledningssystemet, samt korrekturläsning av denna rapport
- Marten Reijm för åsikter, råd och stöd

11 Slutsatser

Målet uppnåddes i och med applikationsmoduler för beräkning av kontaktledningsdynamik för en modell i 3D har utvecklats. Resultaten i avsnitt 6 visar att modulerna klarar av att producera de nya delarna, sektionsövergång och kurva, men att simuleringarna inte ger godkänt resultat på 3D modellen enligt (SEK 2002). Detta kan bero på många saker då det finns väldigt många parametrar som är med och styr och långa simuleringstider försvårar felsökningsarbetet. Vid simulering har inte kontaktkraftsberäkningen inte optimerats alls på grund av tidsbrist.

Ett syfte var att Banverket skulle kunna använda programmet för att utöka sina kunskaper om kontaktledningsdynamik, samt ge support till banregionerna för speciella kontaktledningskonfigurationer. Med hjälp av de moduler som beskrivs i detta arbete så kan Banverket börja utföra detta, men det kommer att krävas mer arbete för att Banverket verkligen skall kunna dra nytta av programmets fulla kapacitet. Detta arbete presenterar en solid grund som senare kan användas för att implementera mer avancerade programfunktioner vid beräkningar av kontaktledningsdynamik.

12 Referenser

Kiessling Friedrich, Rainer Puschmann, Axel Schmieder (2001-10) *Contact Lines For Electric Railways*
ISBN 3-89578-152-5

Lindberg, Rune (2004-10-15) *Systembeskrivning Trådföring*
BVS 543.35001 (B04-1777/EL40)

Svenska Elektriska Kommissionen, SEK (2002-12-04) *Järnvägsanläggnmingar –validering av simulering av det dynamiska samspelet mellan strömavtagare och kontaktledning*
SS-EN 50318

Prata, Stephen (1998) *C++ Programmering* 2 uppl Upplands-väsby: Pagina förlags AB
ISBN 91-636-0416-7

Drugge, Lars (2000-01) *Modelling and simulationof Pantograph-Catenary Dynamics*, Luleå
ISSN 1402 - 1544

Hamrén, Erik. (2001-01-23). *Bevis för Pythagoras sats*. Mimers Brunn. < http://www.mimersbrunn.se/arbeten/523.asp >[2005-05-16]

Tan, Robin (2002-05-01) *The One: A Singleton Discussion*
< http://www.gamedev.net/reference/articles/article1825.asp> [2005-05-15]

Kalev, Danny *Implementing the Singleton Design Pattern*
<*http://gethelp.devx.com/techtips/cpp_pro/10min/10min0200.asp*> [2005-05-13]

Nakamura, Jun (2005-01-25) *C++ In Theory: The Singleton Pattern, Part 2* <*http://www.devarticles.com/c/a/Cplusplus/C-plus-plus-In-Theory-The-Singleton-Pattern-Part-2/1/*> [2005-05-13]