Softwaretest gruppe 6

Afl 3 – ATM system

**Deltagerliste:**

|  |  |
| --- | --- |
| Navn | Studienr |
| Frederik Kastrup Mortensen | **201607221** |
| Søren Schou Mathiasen | 201605264 |
| Martin Haugaard Andersen | 201605036 |
| Alexander Lichtenstein Davidsen | 201608479 |

**Github link:** https://github.com/ZuperZam/SoftwareTest3  
**Jenkins link:** http://ci3.ase.au.dk:8080/job/Gruppe6\_Aflevering3\_UnitTest+Coverage/

Indhold

[Indledning 2](#_Toc512385677)

[Krav 2](#_Toc512385678)

[Design 3](#_Toc512385679)

[Implementering 5](#_Toc512385680)

[Test 8](#_Toc512385681)

[Konklusion 9](#_Toc512385682)

Indledning

Denne opgave omhandler ATM (Air Traffic Monitoring). Når mange fly befinder sig tæt på i hinanden i et givent luftrum vil der være fare for at flyene kolliderer. En mulig løsning på problemet er derfor at udstyre flyene med en Transponder der kontinuert sender informationer omkring flyets position, fart og højde. Denne information sendes til en ATC (Air Traffic Controller), og virker som en bedre løsning i modsætning til at ATC’en skal overvåge hvert fly via radar. Dette aflaster ATC’en ved at give den mere værdifuld information om flyene i luftrummet end den kunne få via radarudstyr.

En transponder receiver modtager transpondersignalerne fra flyene og information fra hvert fly fremvises som et ”track”. Dette track viser så gemt information om det individuelle fly, og sparer ATC fra at skulle gemme på informationen.

Krav

­­Til opgaven er der stillet en række krav, der skal tages højde for under designfasen. Systemet skal monitorerer et luftrum. Alle de fly der befinder sig i luftrummet skal renderes på skærmen. Renderingen skal forekomme på skærmen i konsollen. Alle tracks skal renderes hver gang nyt transponder data modtages. Hvert fly track skal displaye følgende data:

* Tag(tekst streng på 6 karakterer)
* Nuværende position (x og y koordinater, begge i meter)
* Nuværende altitude (i meter)
* Horisontal hastighed (m/s)
* Compass retning (i grader, 0 grader er nord)

Luftrummet kan betegnes som en boks defineret at dets minimum og maksimum koordinater samt minimum og maksimum højde, begge i meter.

* Det syd-vestlige hjørne er ved koordinaterne (10.000-10.000)
* Det nord-østlige hjørne er ved koordinaterne (90.000-90.000)
* Den laveste højde er 500 meter og den højeste højde er 20.000 meter.

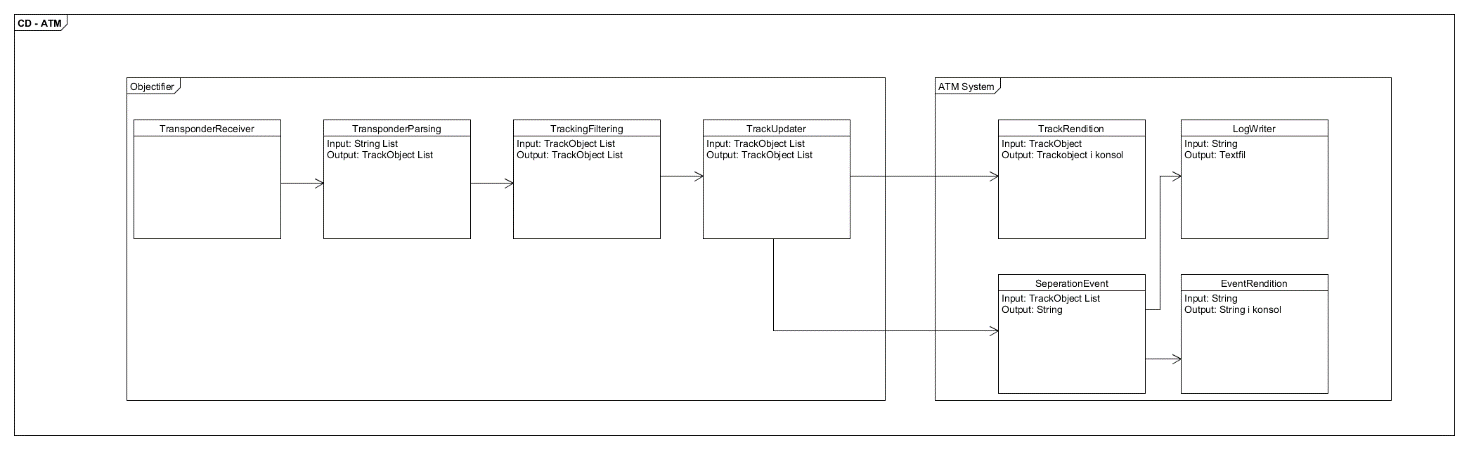
Der skal forekomme et event, hvis to fly er for tæt på hinanden. Hvis den vertikale forskel mellem to fly er mindre end 300 meter og den horisontale forskel er mindre end 5.000 meter, på samme tid, er de to fly inden for hinandens luftrum. Tidspunktet for konflikten samt tags fra de konflikterende fly skal renderes. Det skal fortsat renderes sålænge de to fly er for tætte på hinanden. Når flyende ikke længere opfylder konflikt kravene, skal eventet brydes og skal ikke længere renderes. Udover at rendere en konflikt mellem to fly, skal informationen også logges i en tekstfil.

# Design

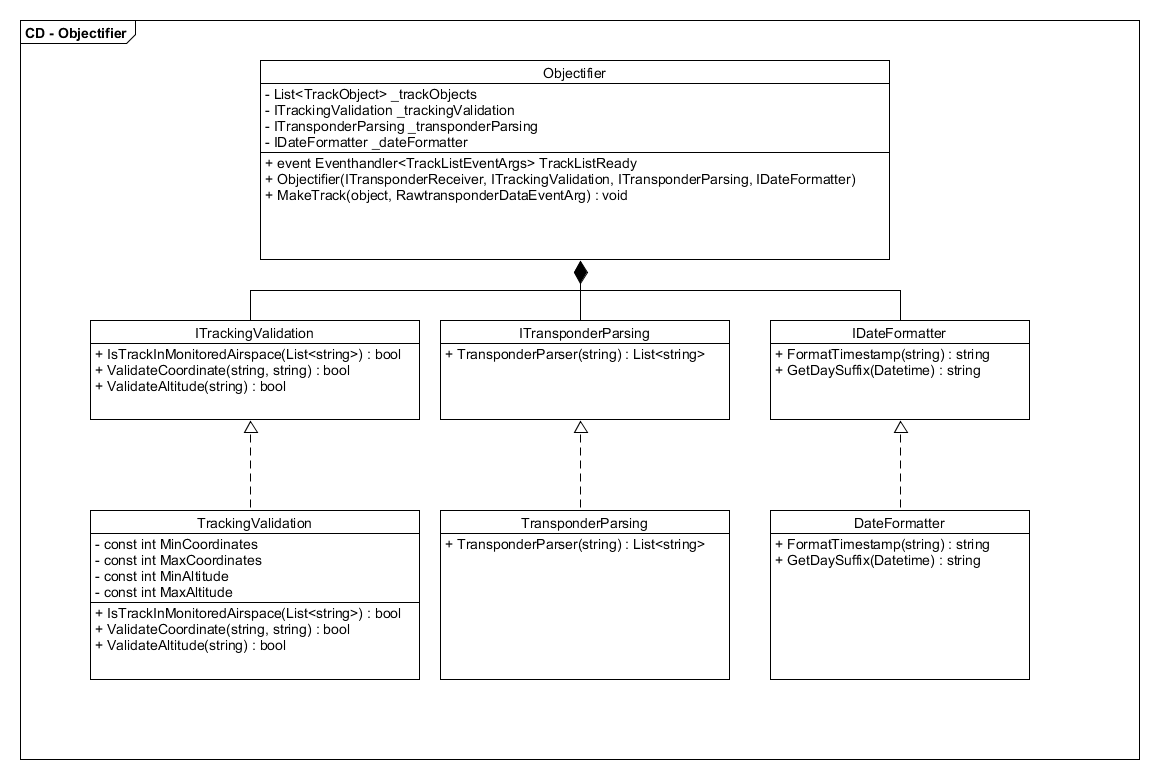
Designet har taget udgangspunkt i SOLID principperne, her med fokus på open/closed og single responsibillity principle. De nødvendige metoder der skal til for at opfylde de stillede krav, er uddelt i en række klasser. Klasserne kan opdeles i to hovedområder, Objectifier og ATM system. Objectifier består af klasserne, transponderparsing, trackingfiltering og trackupdater. ATM system består af klasserne, trackrendition, logwriter, seperationevent og eventrendition.

Objectifiers ansvar er at sørge for det data der modtages fra transponderreceiver.dll behandles korrekt og formateres korrekt. Klasserne her sørger også for at validere dataene, for at se om de overholder de krav der er stillet til koordinaterne.

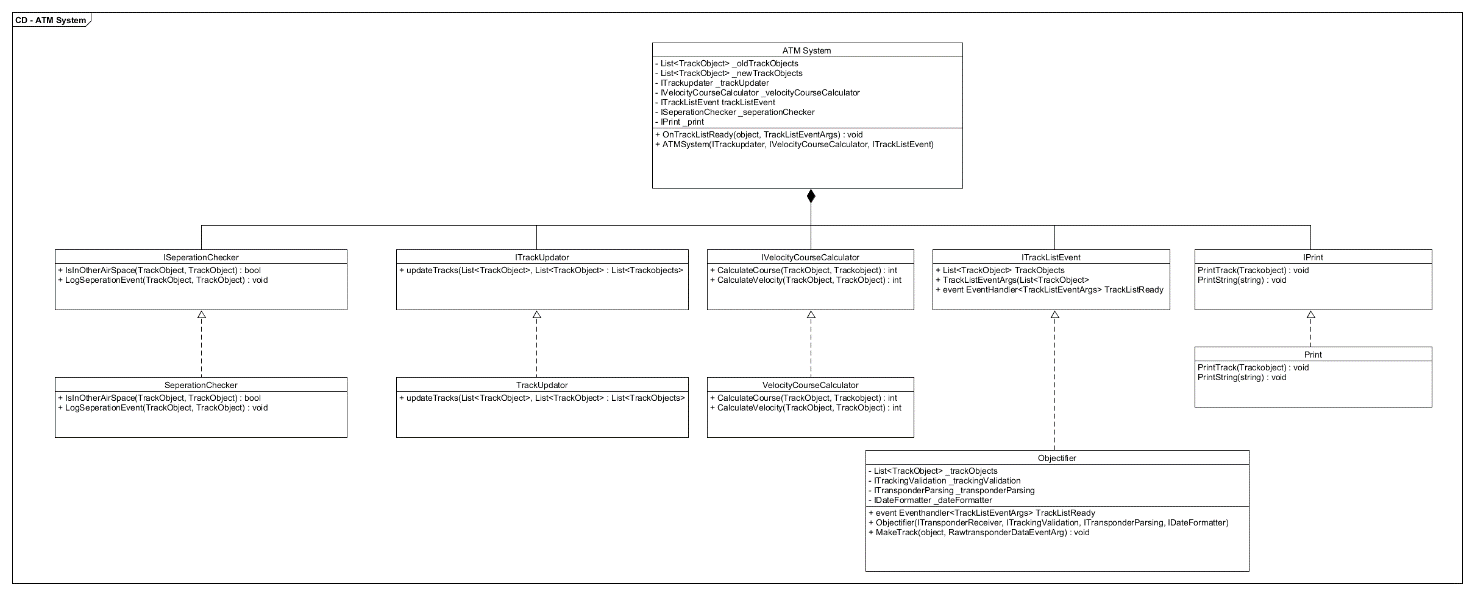
ATM systemet skal sørge at se om et givent fly stadig er inden for grænserne til flyrummet. Det er også her der tjekkes om der forekommer et event mellem to konflikterende fly. Renderingen foregår inde i dette system, samt logging af konflikter.

Der tilføjes interfaces til alle klasser hvor det giver mening for at reducere koblingen mellem de mange klasser, samt muliggøre substitueringer under nogle tests.  
  
Systemet er opbygget som en pipeline uden controllers, da controllers besværliggøre test. En klasse udfører sit ansvar på det input den får ind, hvorefter den sender det opdaterede data videre i pipeline, ved at kalde en funktion fra den næste klasse i pipelinen.   
  
Nedenunder ses et overordnet diagram der viser pipelinen, der er implementeret. En pil symboliserer hvilken klasse der kalder den næste klasse.  **Diagrammer kan ses i fuld format i de vedhæftede PDFer.**   


Her ses klassediagrammet over Objectifier og de klasser den benytter. Objectifier har et object af ITrackValidation, ITransponderParsing og IDateFormatter i sig. Objectifier subscriber på et event fra transponderReceiver kaldet TransponderDataReady. Der følger en strengliste med eventet, hvor hver eneste streng i listen er et fly. Den parser her dataet i strenglisten i et TrackObject, hvorefter Timestamp formateres til noget lettere læseligt. Til sidst gemmes de i en TrackObject liste. Når alle tracks er objectified, raises et event, der sendes til ATM System.



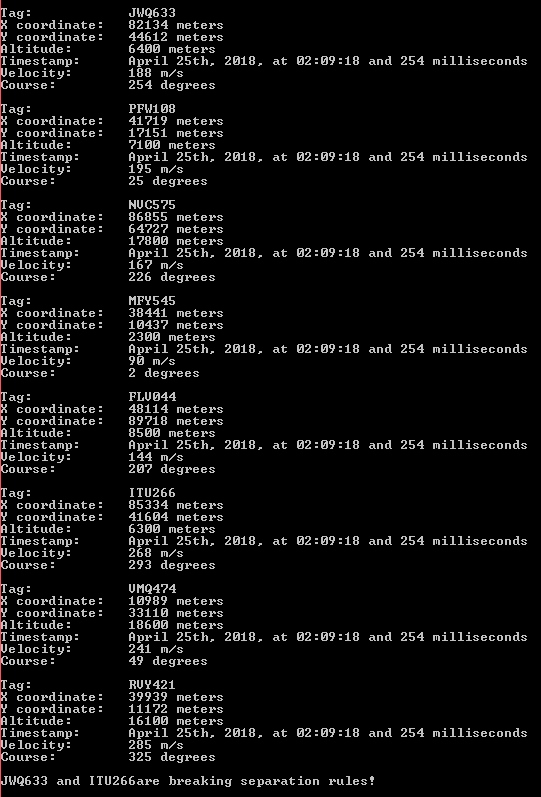
Klassediagrammet over ATM System og de tilhørende klasser. ATM System indeholder et objekt af ISeperationChecker, ITrackUpdator, IVelocityCourseCalculator, ITracklistEvent og IPrint. ATM System subscriber på et trackListEvent kaldet TrackListReady.



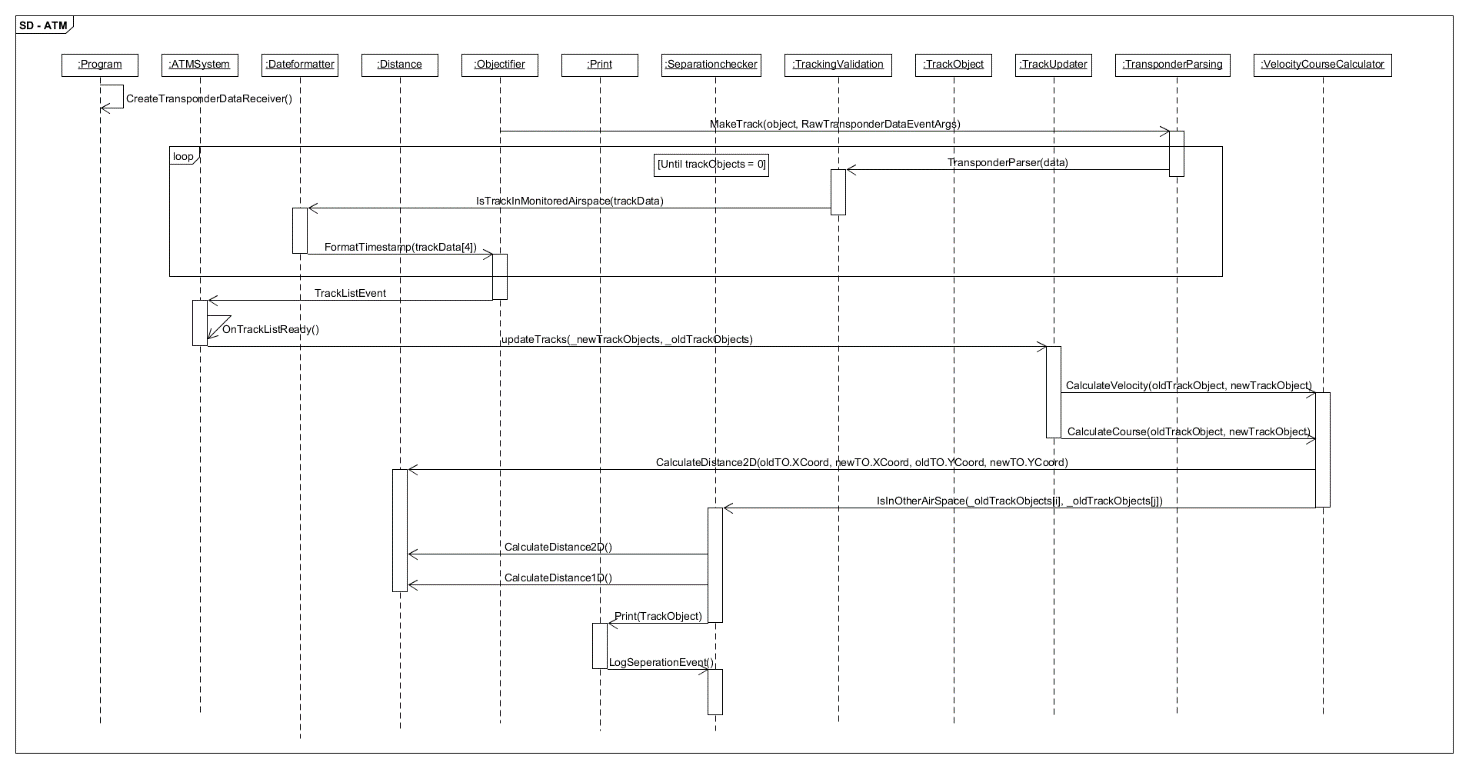
# Implementering

Implementeringen af programmets klasser blev opdelt efter ønske. Gruppemedlemmerne skiftedes mellem at skrive unit tests og implementering. Hvis et gruppemedlem blev træt af at lave unittests, var det muligt at bytte med en der skrev implementering og omvendt.

Herunder kan ses et Konsolvindue der viser en del af de tracks systemet monitorer. To Tracks er i hinandens luftrum, og derfor er der en besked om dette i bunden af vinduet.



Nedenunder ses vores sekvensdiagram over ATM.



# Test

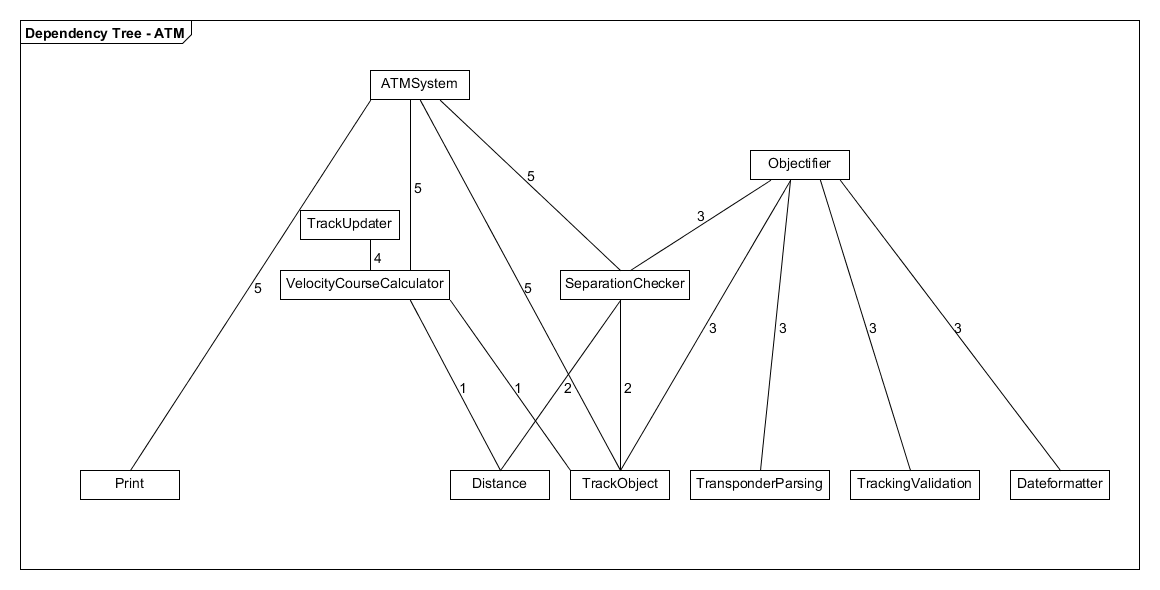
Som gruppe gør Jenkins at vi kan se vores gruppemedlemmers push til git. Men da vi i denne opgave langt størstedelen af tiden arbejdede i samme lokale, gav det ikke helt det store udbytte. Vi brugte her i stedet de funktioner der er indbygget i Resharper til at se coverage og unittest resultater. Men det er fedt at have en central test så der kan være en computer der skal være fokus på at kunne køre testene.

Under test af klassen ATMSystem stødte vi på et problem angående substituerede metoder som lavede en af vores variable til NULL. Mange forsøg blev brugt på at få den til at returnere noget korrekt også gennem NUnits, Returns() funktion, men til sidst måtte vi opgive forsøget, da vi ikke kunne finde en løsning på problemet.   
  
Fra klassen SeparationEvent er LogEvent funktionen ikke testet med NUnit. Det samme gælder for klassen print. Dette er fordi de er boundary klasser som giver et tekstuelt output på Console og i tekstfil. Test kan laves for at tjekke at de modtager de korrekte strings, og en test kunne laves for at læse en tekstfil der lige er skrevet til. Dette er dog ikke gjort da der i tidligere eksempler ikke er testet tekst outputs.

Til integrationstest benytter vi os af bottom up metoden. Grunden til dette valg falder på at vi føler det er den mest ingeniørfaglige korrekte metode at gribe integrationstesten an på. På denne måde bliver boundary og boundary nære klasser testet først. Disse kan ofte være komplicerede da de kan være hardware nære. Vi har i gruppen også haft gode erfaringer med denne metode fra den tidligere aflevering   
  
Nedenunder ses en tabel over vores integrationsplan. Den er lavet ud fra det dependency der er lavet på baggrund af sekvensdiagram for implementering.

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Step | ATMSystem | Objectifier | VelocityCourseCalculator | SeparationChecker | TrackObject | Print | TrackUpdater | Distance | Dateformatter | TransponderParsing | TrackingValidation |
| 1 |  |  | D |  | X |  |  | X |  |  |  |
| 2 |  |  |  | D | X |  |  | X |  |  |  |
| 3 |  | D |  | X | X |  |  | X | X | X | X |
| 4 |  |  | X |  | X |  | D | X |  |  |  |
| 5 | D |  | X | X | X | X | X | X | X |  |  |

Herunder ses vores dependency tree.



# Konklusion

Gruppen kan konkludere at selvom ATM programmet ser ud til at køre helt korrekt, har det ikke været muligt at lave alle testene på grund af fejl med nogle substitutions.