AARHUS UNIVERSITET
INGENIØRHØJSKOLEN

SWT Aflevering 3
Air Traffic Monitoring
Team number: 3
25/04-2018

Navn	Studienummer
Fatima Kodro	201609565
Søren Bech	201604784

Jenkins build jobs

Unit og coverage:

http://ci3.ase.au.dk:8080/job/Team18103HandIn3_UnitAndCoverage/

Integration:

http://ci3.ase.au.dk:8080/job/Team18103HandIn3_Integration/

Software metrics:

http://ci3.ase.au.dk:8080/job/Team18103HandIn3_SoftwareMetrics/

Github repository

https://github.com/FatimaAU/HandIn3

Indhold

Indledning 3
Krav 3
Design 3
Klassediagrammer 4
Sekvensdiagrammer 6
Dependency tree 6
Klasser 6
Konklusion 7

Indledning

Der skal i aflevering 3, laves et system der kan overvåge luftfarten i et givent område, samt kunne holde øje med interessante event der måtte forekomme.

Problemet med luftfarten, er at der er mange fly, der flyver omkring samme område. Dette kan medføre stor farer, hvis nogle fly kommer for tæt på hinanden. Derfor skal hvert fly kunne registreres med position, fart og højde, for at sikre sig at ingen fly er for tætte på hinanden.

Krav

Systemet skal kunne overvåge et bestemt luftrum, og registrere alle fly inden for dette luftrum, og ingen andre. Alle fly registreres med Tag, position, højde, fart(m/s) og kurs(grader).

Hvis nogle fly skulle flyve for tæt på hinanden (5000 meter horisontalt og 300 meter vertikalt) skal der kunne hæves et event indtil de er ude for farer igen. Når dette event er hævet, skal de fly der er for tætte på hinanden udskrives til en fil, med både flyenes tag og tidsrummet.

Der blev udgivet en Transponder Receiver dll i starten af afleveringen, som kunne modtage transponder data for flyene i luftrummet. Ud fra disse data, kunne flyenes Tag, position, højde registreres, samt tidspunktet for de forskellige transponder data.

Design

I starten af afleveringen blev der udgivet en dll-fil, hvorefter der blev arbejdet sammen i gruppen med at få printet de simple strenge ud. Herefter når dette var muligt, blev der oprettet nogle klasser, som kunne håndtere dette data. Det, som skulle håndteres, var at dataen blev parsed korrekt, samt at tidsstemplet blev omdannet og printet ud på en bestemt måde. Der blev herefter lavet test til disse klasser.

Herefter startede den egentlige implementeringsfase, som inkluderer resten af systemet. Der blev her anvendt TDD i en blanding af pair-programming og individuel programmering, hvor opgaverne blev delt ud. Dette gav mening idét systemet har mange dele, og det ville være en gode idé at definere krav (og test) før koden blev implementeret, dvs. der blev defineret en feature list.

Hvis en fejl fandtes i implementeringen, ville en test blive lavet omkring fejlen, og derefter ville den blive rettet.

Herefter blev koden refaktoreret ind i klasser. Koden havde en naturlig lav kobling, da tests kræver at en klasse eller kode skal være afkoblet fra sine afhængigheder.

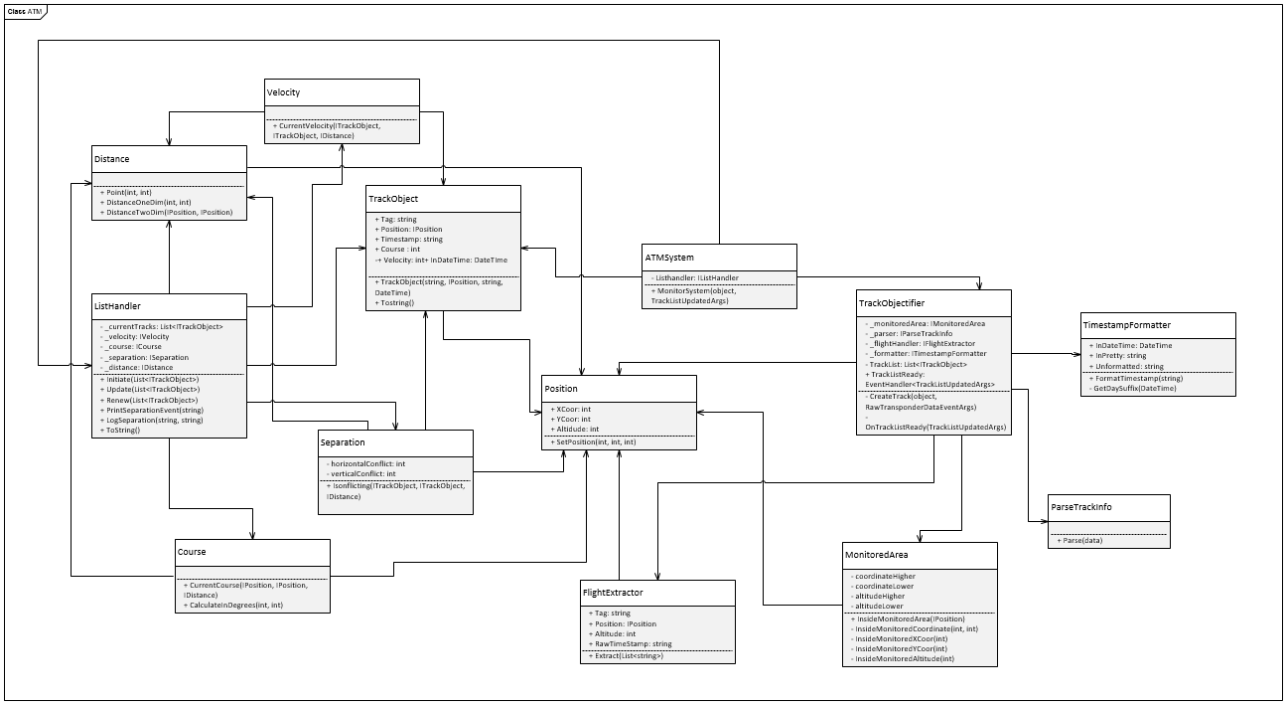
Efter koden nu var blevet refaktoreret ind i klasser, skulle der laves unit tests til alle klasser. Dette blev opdelt i grupperne, så begge gruppemedlemmer havde ansvar for test af forskellige klasser.

Da unit tests blev lavet færdigt, skulle der nu planlægges en integrationsplan. Der blev kigget på de forskellige strategier til at kunne lave integration sammen i gruppen. Herefter blev der lavet et sekvensdiagram over hele systemet, og der blev derefter lavet intergrationstests ud fra både sekvensdiagrammet og klassediagrammet.

Der er taget udgangspunkt i Single responsibity, idet ansvaret i systemet er blevet opdelt i mange forskellige klasser. Dette blev gjort for at få en lavere kobling mellem klasserne, samt er det nemmere at kunne stubbe klasserne i de forskellige tests.

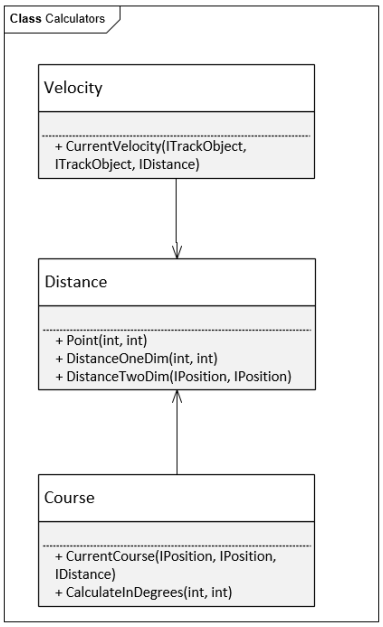
Klassediagrammer

På Figur 1 ses et klassediagram over hele systemet med deres metoder og attributter, samt hvilke klasser der er afhængige af andre klasser. Det mest afhængige klasser er ATMSystem, ListHandler og TrackObjectifier. For at gøre det mere overskueligt, er klassediagrammet også delt op i 3 dele: Calculators, Objectifier og UpdateAndCheck, som kan ses på Figur 2, Figur 3 og Figur 4.



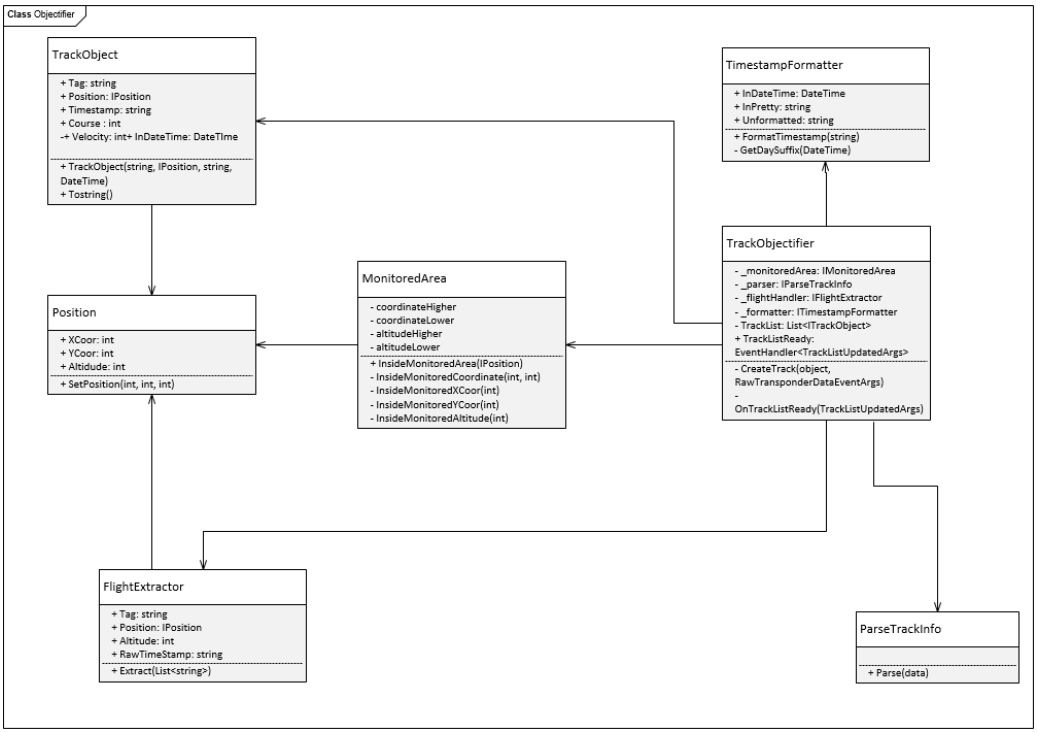
Figur 1: Klassediagram over hele systemet

På Figur 2 ses et klassediagram over de klasser der har noget med en Calculator at gøre. Det er de klasser der regner farten og kursen ud for flyene.



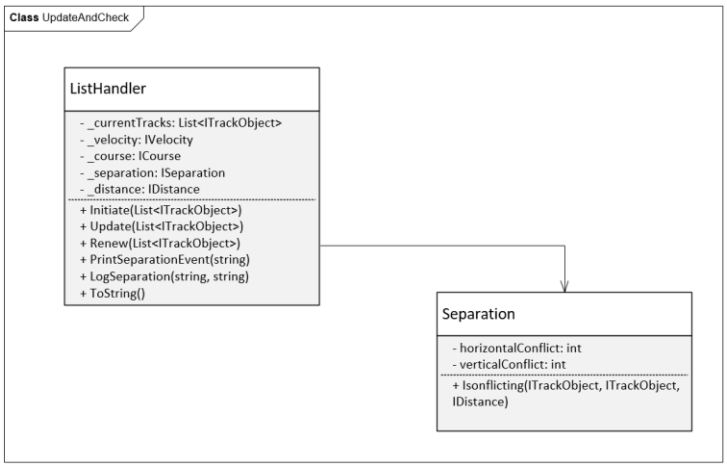
Figur 2: Klassediagram over Calculators

På Figur 3 ses et klassediagram for de klasser der har noget med Objectifier af flyene at gøre. Det er de klasser der kigger på data af flyene som position, tag, tid ovs.



Figur 3: Klassediagram over Objectifier

På Figur 4 ses et klassediagram over de klasser, der har noget med håndteringen af flyene at gøre. Det er de klasser der tjekker om flere fly er alt for tæt på hinanden, samt det der sørger for at logge denne data i en fil, hvis dette skulle ske.



Figur 4: Klassediagram for UpdateAndCheck

Sekvensdiagrammer

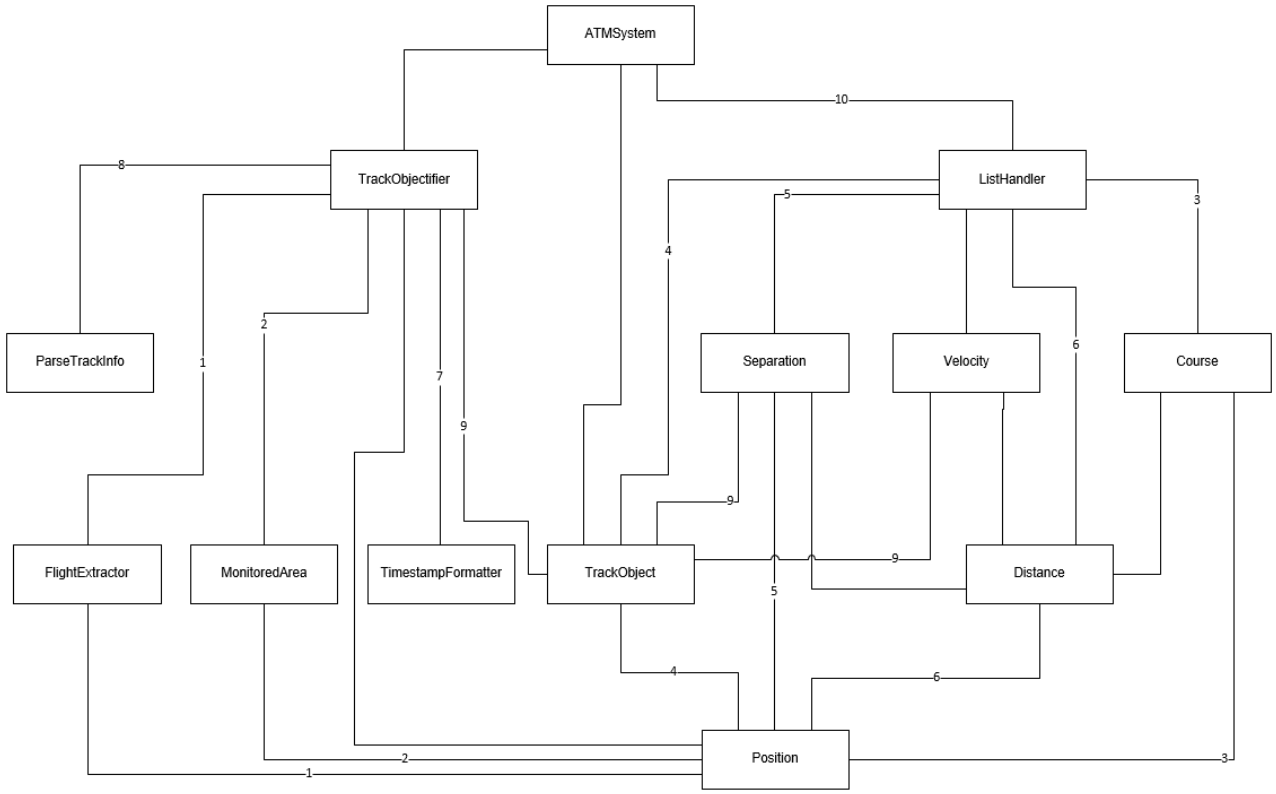
Sekvensdiagrammet er meget stort og er derfor indsat som en vedhæftet fil.

På sekvensdiagrammet bliver alle klasser brugt, samt et system som i dette tilfælde vil være main-programmet. Systemet kalder de 3 klasser: ATMSystem, TrackObjectifier og ListHandler. Disse 3 klasser kalder på de andre klassers funktioner, så hele systemet har en sammenhæng.

Dependency tree

På Figur 5 ses det dependency tree, der er brugt til at lave integrationstest på ATM. Som strategi for integration, er der blevet brugt buttom-up, da det gav mest mening at bruge, da de klasser nederst i dependency-træet, altså dem med få afhængigheder, skulle bruges for at teste de klasser længere oppe.

En anden strategi til overvejelse var sandwich-test, men dette var besværligt da "bunden" og "toppen" kolliderer med det samme.



Figur 5: Dependency tree

Klasser

Vigtige klasser at snakke om er ATMSystem og TrackObjectifier.

TrackObjectifier er den klasse, der sørger for at objectificere den streng af data der bliver modtaget, så der kan arbejdes med det.

ATMSystem er den klasse, der overordnet holder styr på alt data om flyene og har en afhængighed igennem alle de andre klasser.

Konklusion

Der er i løbet af projektet blevet gjort brug at jenkins serveren. Jenkins er god til at holde styr på alle unit- og integrationstest, da den er uafhængig af de forskellige platforme og systemer, som holdes af de forskellige brugere. Den er god til at kunne vise forskellige oplysninger om ens tests, som f.eks. hvor optimalt systemet er. Ellers har der været god brug af hele tiden at holde øje med hvor meget af systemet der er blevet coveret, da der blev sat et mål på at coverage skulle være på 100%.

På Tabel 1 og Tabel 2 ses en tabel over hvilke klasser der blev brugt i de forskellige trin.

D: Dette modul er inkluderet, og den der er drevet
X: Dette modul er inkluderet
S: Dette modul er stubbed eller mocked

Step #	Position	Flight-Extractor	Monitored-Area	Timestamp-Formatter	TrackObject	Distance	ParseTrackInfo
1		D	S				X
2		X	D	S			S
3					X	S	
4	S				D	S	
5						S	
6					X	D	
7		X	X	D			S
8		X	S	S			D
9		X		X	D		X
10		X	X	X			X

Tabel 1: Buttom-up planner

Step #	Separation	Velocity	Course	TrackObjectifier	ListHandler	ATMSystem
1				X		
2				X		
3	S	S	D		X	
4	S	S	S		X	
5	D	S	S	X	X	
6	S	S	S		X	
7						
8				X		
9				X		
10				X	S	D

Tabel 2: Buttom-up planner