# Handin 2 – ATM Del 1

Gruppe nummer: Gruppe 17

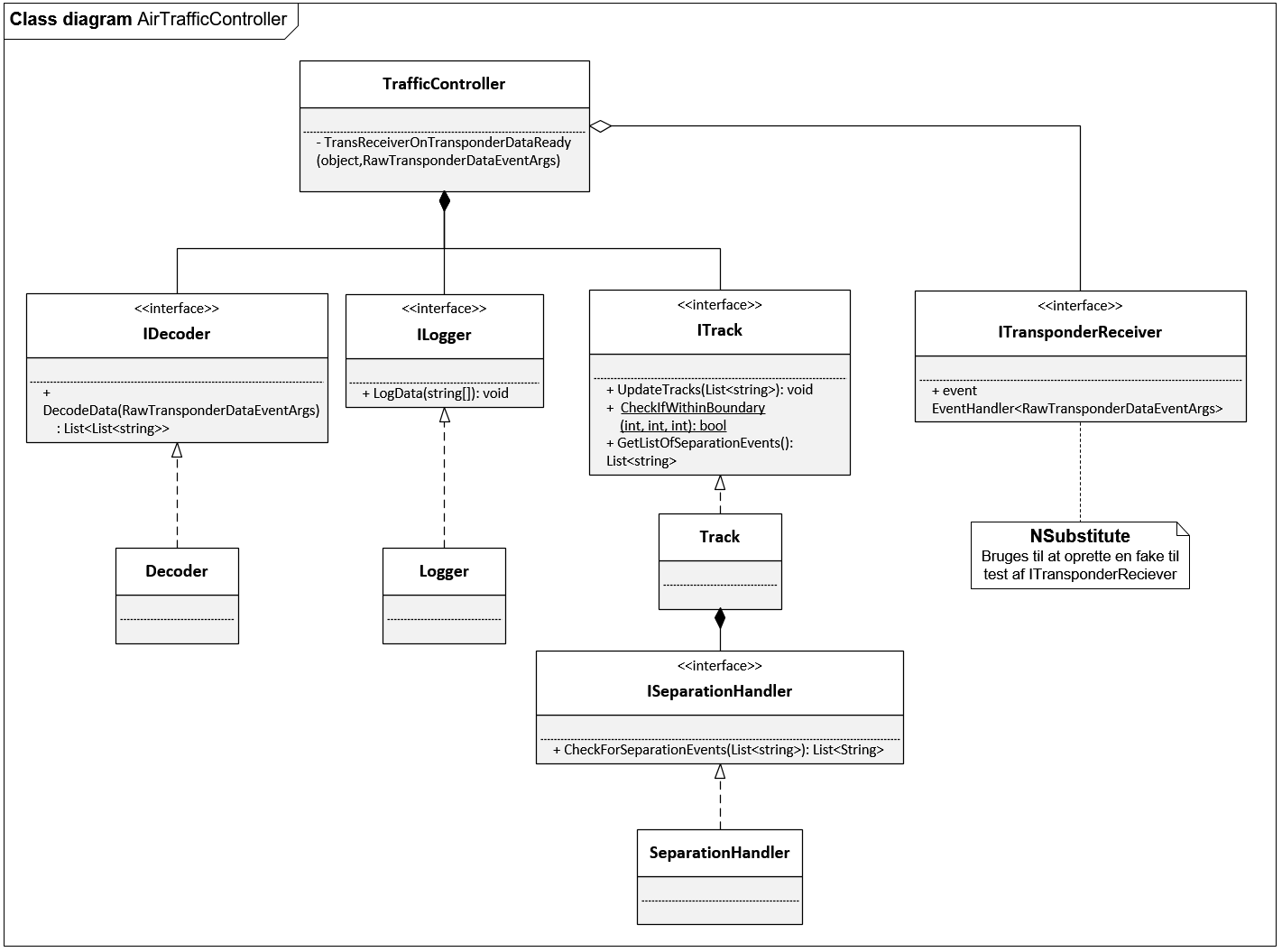
|  |  |
| --- | --- |
| **Navn** | **Studienummer** |
| Anders Fibiger | 201611672 |
| Viktor Søndergaard | 201610466 |
| Arni Medic | 201611674 |
| Aram Al-Sabti | 201406080 |

**URL til Jenkins build job: http://ci3.ase.au.dk:8080/job/SWT2\_AAA/**

**URL til GitHub repository:** [**https://github.com/andersfibiger/SWT2**](https://github.com/andersfibiger/SWT2)

# Software Design

Der er på baggrund af systembeskrivelsen er der udarbejdet et design af AirTrafficController.



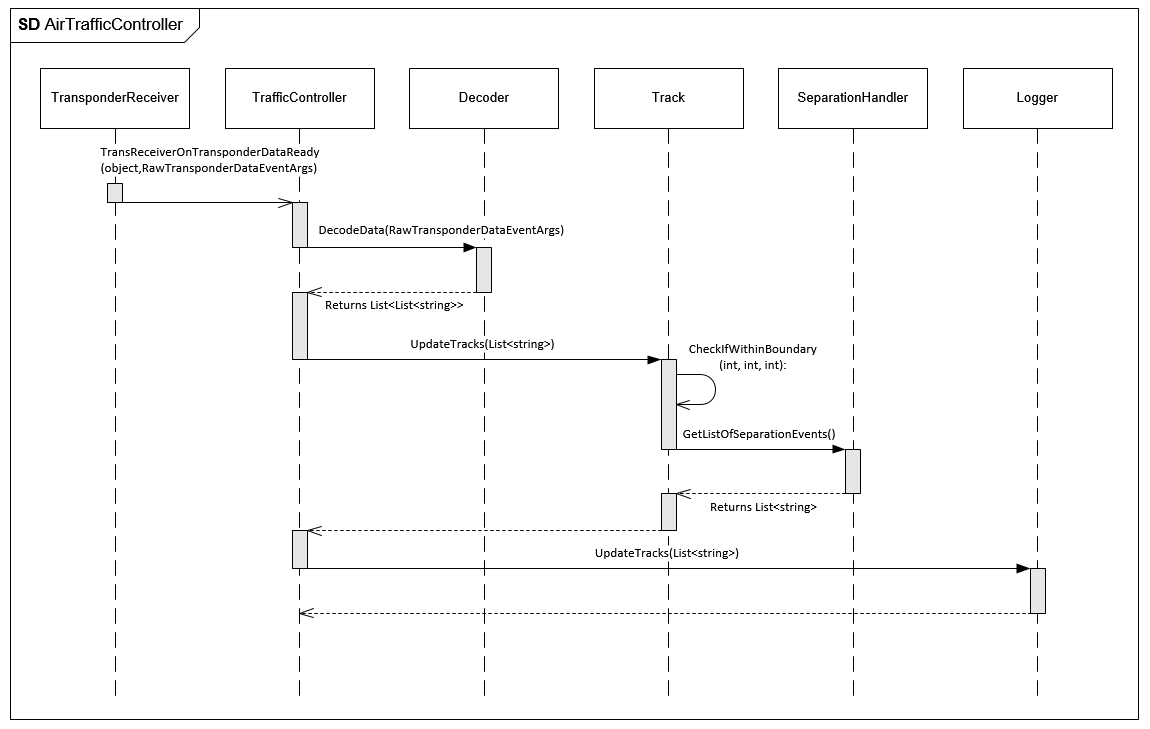
Figur 1 - Klassediagram for AirTrafficController

På overstående figur 1 kan klassediagrammet for AirTrafficController ses. Klassen TrafficController bruges til at knytte underklasserne sammen på en struktureret måde, samt at benytte klassen til at subscribe på eventet fra TransponderReviever. TrafficController består af en række interfaces, herunder IDecoder, ILogger, ITrack, samt ITransponderReciever. ITrack består yderligere af ISeparationHandler interfacet.

At der gøres brug af interfaces, gør først og fremmest at koblingen er mindre mellem baseklassen og den konkrete klasse. Derudover er det er smart at benytte i forbindelse med unit.test, idet vi kan implementere en fake i stedet for at ændre den reelle klasse. Da vi netop gør brug af interfaces er baseklassen ligeglad med de konkrete klasser. Dette betyder at der simpelt kan gøres brug af dependency injection, ved at konstruere baseklassen med de fake afhængigheder.

I forhold til AirTrafficController-systemet er der valgt at lægges vægt ITransponderReciever, i forhold til at gøre brug dependency injection. Det er netop fordi klassen har ukontrolleret adfærd (output), hvilket betyder at en fake er nødvendig. Her er der valgt at der benyttes af frameworket NSubstitute. Vi har valgt at benytte dette framework, da det er nemt at substituere en ”ægte” klasse med denne fake, samt at der ikke er behov for at explicit implementere vores fakeklasser.

For at vise klassernes interaktioner nedad en tidslinje, er der udover klassediagrammet blevet udarbejdet et sekvensdiagram.



Figur 2 - Sekvensdiagram for AirTrafficController

På overstående figur 2 kan sekvensdiagrammet for AirTrafficController ses. Her ses det at eventet opstår fra TransponderReciever og derefter sendes videre til TrafficController, som håndterer eventet. Herfra kaldes Decoder, som behandler det modtagne data på ønsket vis. Når den proces er færdig, overtager TrafficController igen og kalder Track-klassen med kommando om at opdatere listen af Tracks. Da Track kun måler indenfor et bestemt område tjekker den for om de modtagne Tracks er indenfor området. Hvis flyene så er for tæt på, hvilket er bestemt af nogle specifikke begrænsninger, så bliver eventet for SeperationHandler kaldt. Denne klasse returnerer listen for de fly som er i farezonen. Til sidst returneres der til TrafficController, som sørger for at printe den opdaterede liste ud via Logger-klassen.

# Software Classes

How did you divide the software classes between group members for implementation and test? Why did you divide it as you did?

Vi har lavet softwarearkitekturen i skolen og lavet en del pair programmering sammen, det har vi valgt at gøre for at alle har den fulde forståelse af systemet. Så delte vi de forskellige test klasser ud, vi delte det lidt ud efter lyst, men sørgede for hver klasse var i sin egen fil. Da man kan spare meget tid ved at dele opgaverne ud, men effektiviteten falder en del hvis man skal bruge tid på merge errors hver gang.

# CI Server

How did the use of a CI server help you – did it help you at all? How/why not?