SWT – Air Traffic Monitor 1 & 2

Handin – Gruppe 10
2018

Daniel Tøttrup – au544366 – 201509520
Mathias Lønborg Friis – au547006 – 201505665
Lasse Lildholdt – au546990 – 201507170
Philip Nygaard Schmidt – ay547068 - 201506381

AARHUS
UNIVERSITET
AARHUS SCHOOL OF ENGINEERING

Indholdsfortegnelse

ATM – Del 1 – HandIn 2

Jenkins Projects and Repo URL

Jenkins Unit http://ci3.ase.au.dk:8080/job/SWTgroup10/

Github repo https://github.com/Phalap/SWTGroup10/tree/master/Handin2

Refleksion

Softwarearkitektur

For at skabe enighed om designet udarbejdes først en skitse til et klasse-diagram, som beskrev modelklasser, interfaces samt "hoved-klassen", ATMclass. Udover diagrammet udarbejdedes et dokument, som kort beskrev alle klasser, deres formål og deres hovedfunktionalitet. Diagrammet er vedlagt som Appendix 1, og dokumentet kan sendes ved efterspørgsel.
Derudover udarbejdedes et sekvens-diagram, som udgjorde en skitse for flowet i ATMclass'es handleNewTrackData. Diagrammet er vedlagt som Appendix B

Interfaces
I et forsøg på at gøre vores kode mere testbar, har vi brugt interfaces til at lave løs kobling i vores system. På denne måde kan de enkelte dele af systemet testes separat. Eksempelvis kan funktionaliteten i Airspace testes udenom resten af systemet. Dette gør det nemmere at lokalisere, hvor eventuelle fejl måtte opstå I systemet, hvorefter disse kan udbedres.

Fakes
Det at vi burger interfaces, gør at vi kan fake vores implementeringer af interfaces, så vi kan verificere at de bliver kaldt I de korrekte sammenhænge. Dette gøres simpelt ved at lave en counter, som tæller op for hvert system-kald, og eventuelt attributter til at holde på states af de passede parametre. Efter systemet har kørt i test-sammenhængen, kan man så asserte på fake-objektets counter eller attributer.

Observer Pattern
Systemet er desuden designet med et observer-pattern, hvilket resulterer i at hver gang der kommer ny data, behandles denne af alle "subscribede" airspaces. På denne måde vil det være nemt senere at koble flere Airspaces på systemet.

Integrationstest og strategien for denne

Ingen test af FileLogger og ConsoleRenderer
Vi har undladet at teste FileLogger og ConsoleRenderer.
ConsoleRenderer testes ikke, da den blot udskriver data I consolen, og da vi tester i Fake-udgaven at det givne parameter bliver modtaget ordentligt må den antages at virke. Udskrivningen til consolen verficeres istedet ved en visuel test.
FileLogger testes heller ikke, da vi ikke er helt sikre på om dette skal gøres, og I så fald hvordan det skal gøres. Her foretages istedet en manuel test.

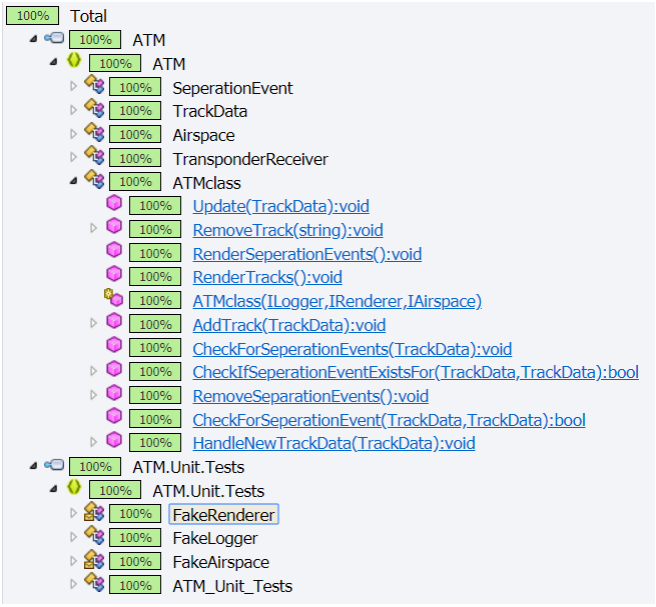
Boundary Values
I forbindelse med test af Airspace-klassen skal funktionen checkIfInMonitoredAirspace checke om koordinaterne fra det indkommende fly ligger inde for de specificerede grænser. I denne forbindelse har vi testet både x- og y-koordinaterne, når de er henholdsvis inde for, ude for og på grænserne, for at se om metoden agerer som forventet.

Continuous integration

Vi har brugt git til at kunne versionsstyre, samt til at kunne arbejde flere mand på det samme projekt samtidig. På denne måde har vi alle kunnet arbejde samtidigt, hvilket selvfølgelig har øget vores effektivitet.

Verfikation af tests
Vi har oprettet et Jenkins-job til at køre alle vores tests igennem, og rapportere hvor mange der er gået godt. På denne måde ses det tydeligt hvis noget pushet kode breaker noget. Dette har resulteret i at fejl hurtigt er blevet opdaget og rettet.

Code Coverage
Ud over at verificere at den tilføjede kode fungerer, har vi desuden brugt Jenkins til at oprette en Code Coverage report, hver gang der er blevet uploadet ny kode, som går igennem alle tests. Denne Coverage Report giver et overblik over hvor meget af den skrevne kode, der bliver berørt af tests. Vi har undladt at teste getters og setters, da disse blot er standard-implementeringerne leveret af .NET. Derudover har vi, som nævnt tidligere, undladt at teste klasserne ConsoleRenderer og FileLogger. For at dette ikke resulterer i "røde tal" i coverage reporten har vi lavet Coverage Filters i vores dotCoverCoverageConfig-fil, som tager højde for dette. Når der ses bort fra disse, har vi opnået 100% Code Coverage.



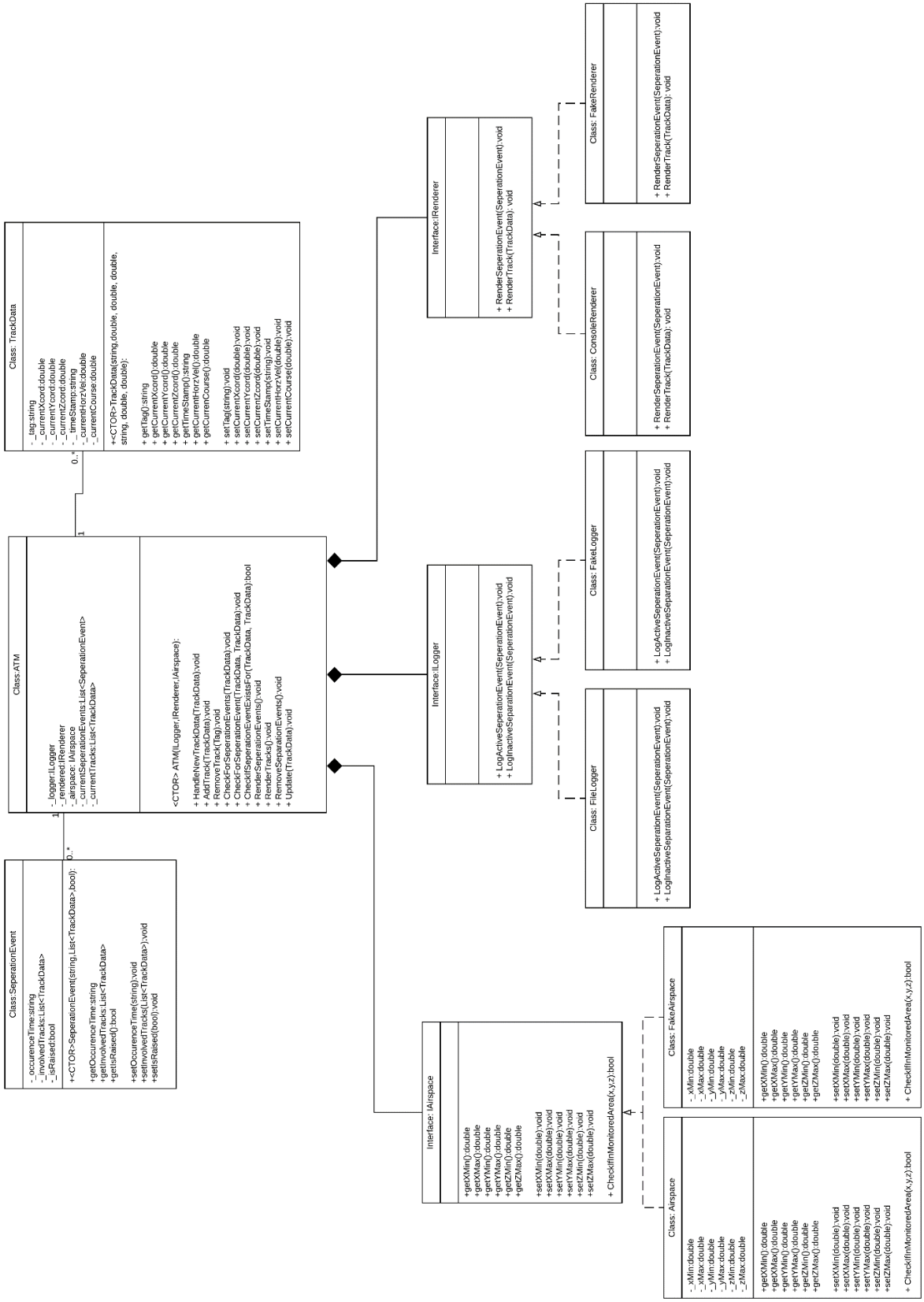
Figur 1: Coverage Report

Konlusion

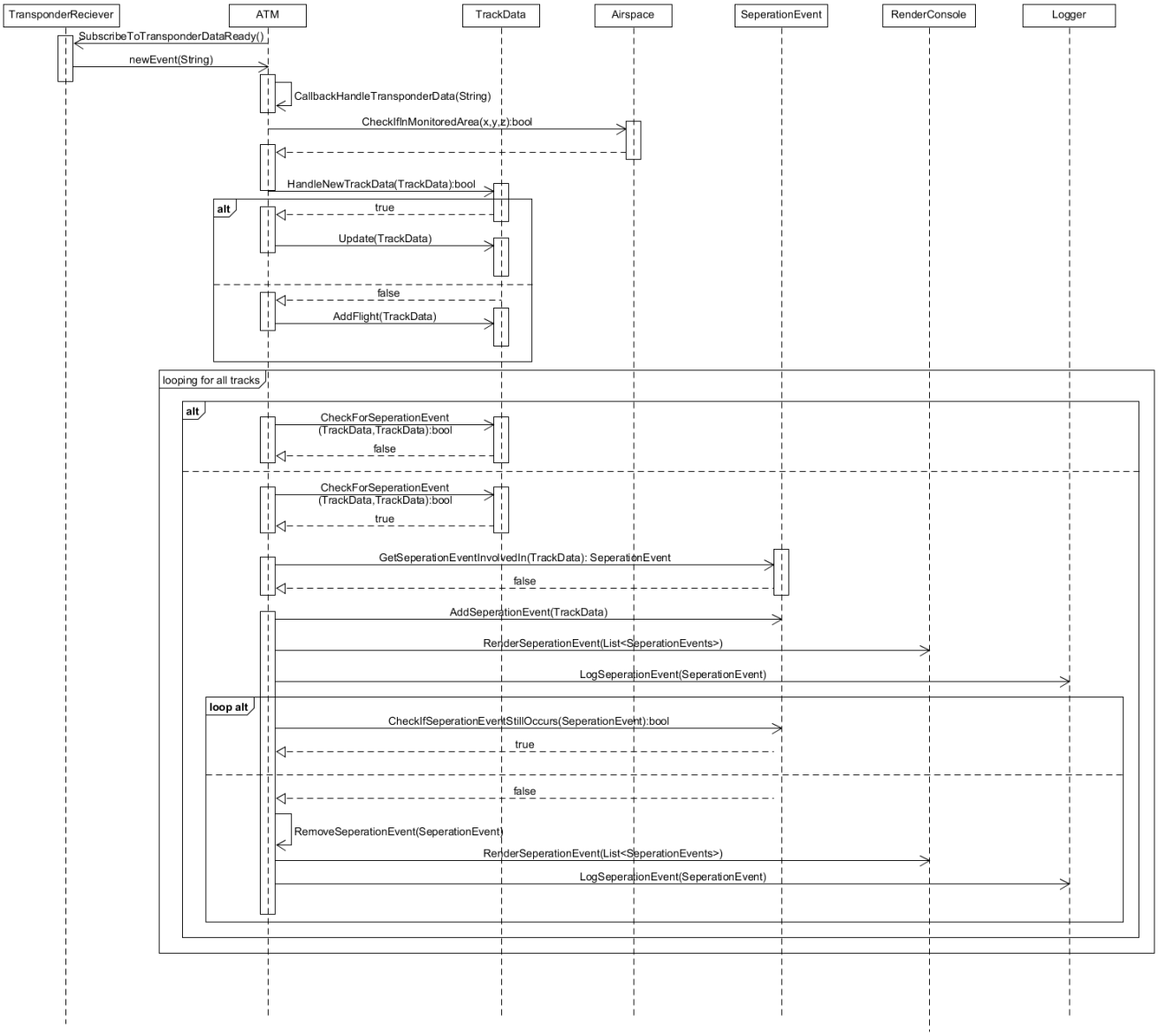
Gruppen er tilfreds med resultatet af arbejdet. Da alle gruppemedlemmer havde tidligere erfaring med at arbejde med git, var dette ikke den store øjenåbner-oplevelse, da vi allerede vidste hvor meget det hjælper i forhold til at arbejde flere mand på et projekt. Unit Tests samt Jenkins-jobbet har dog været lidt af en øjenåbner for gruppen, da dette var med til at skabe et klart overblik over hvad der virkede, hvad der ikke gjorde, og hvad der manglede at blive testet.

Appendix A

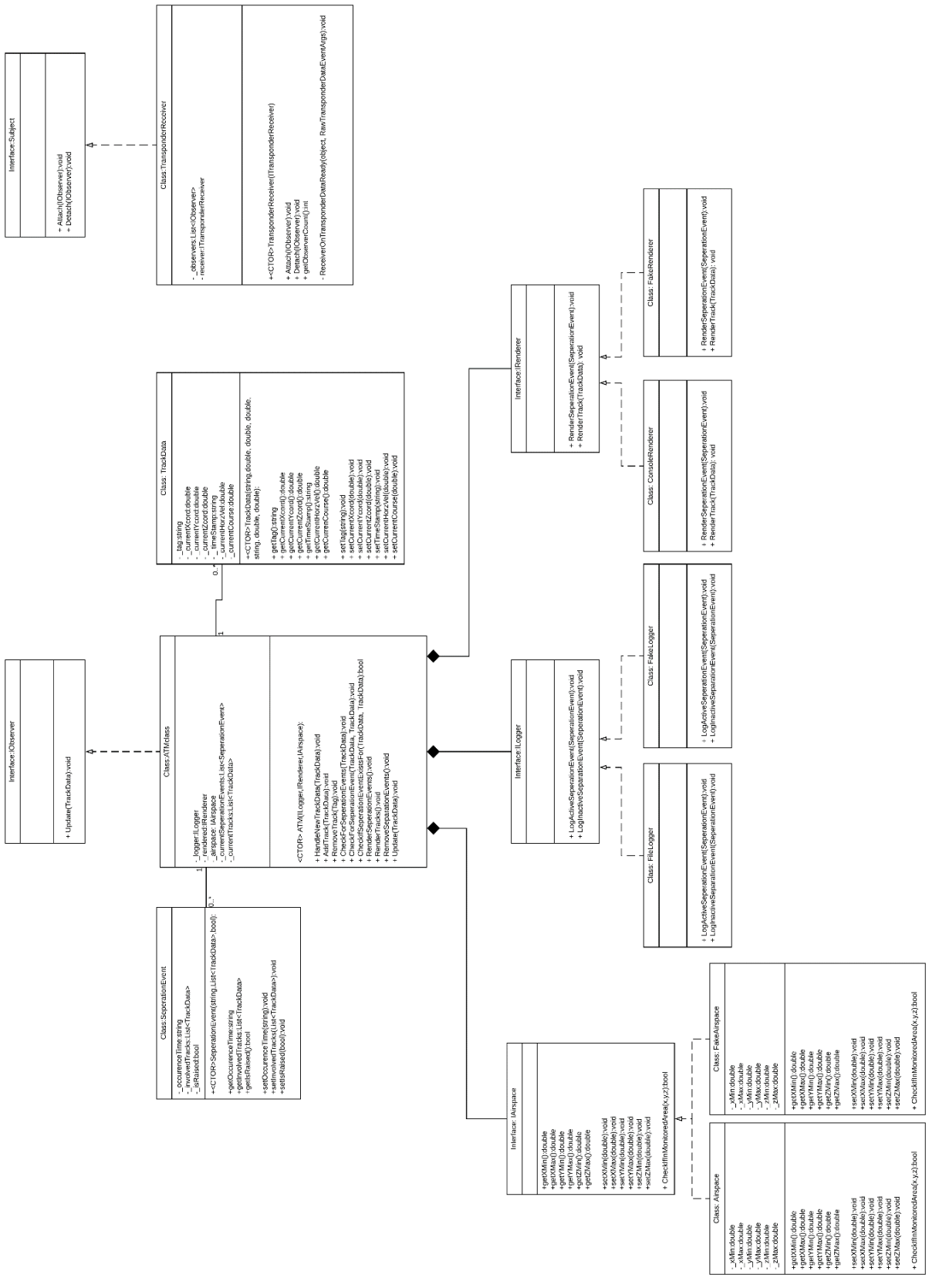
Appendix A1 – Klassediagram udkast



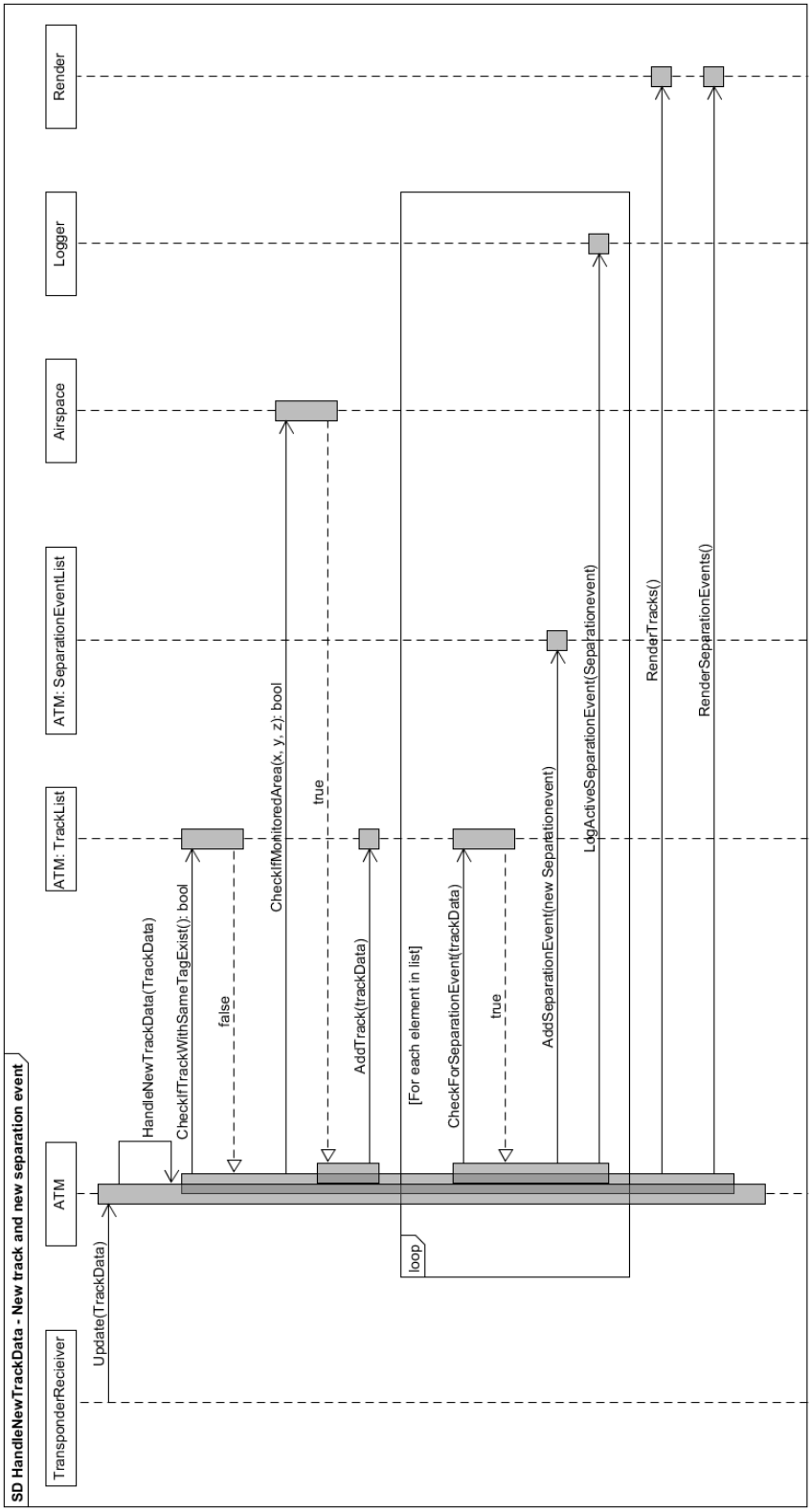
Appendix A2 – Sekvensdiagram udkast for handleNewTrackData



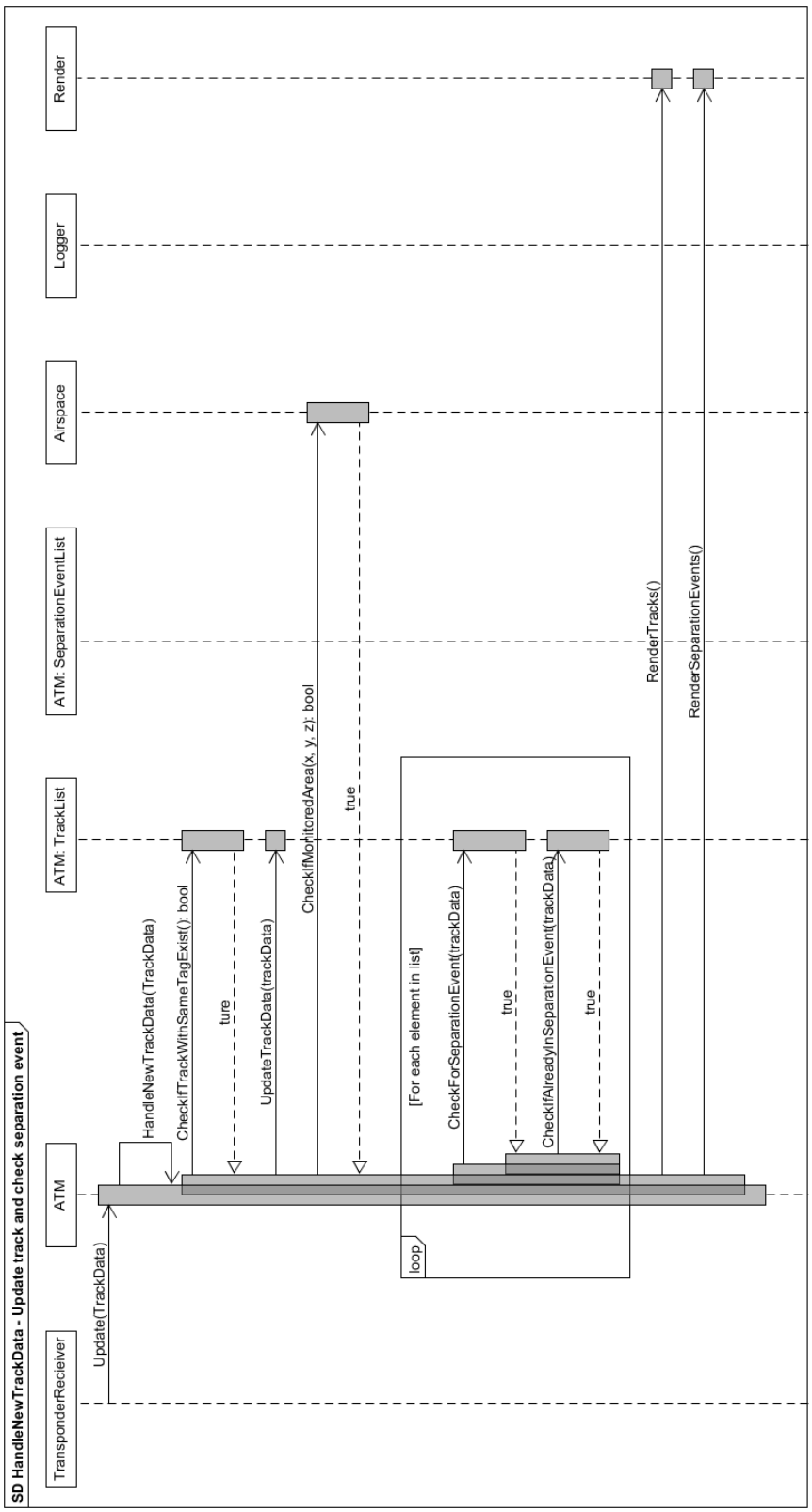
Appendix A3 – Revideret klassediagram



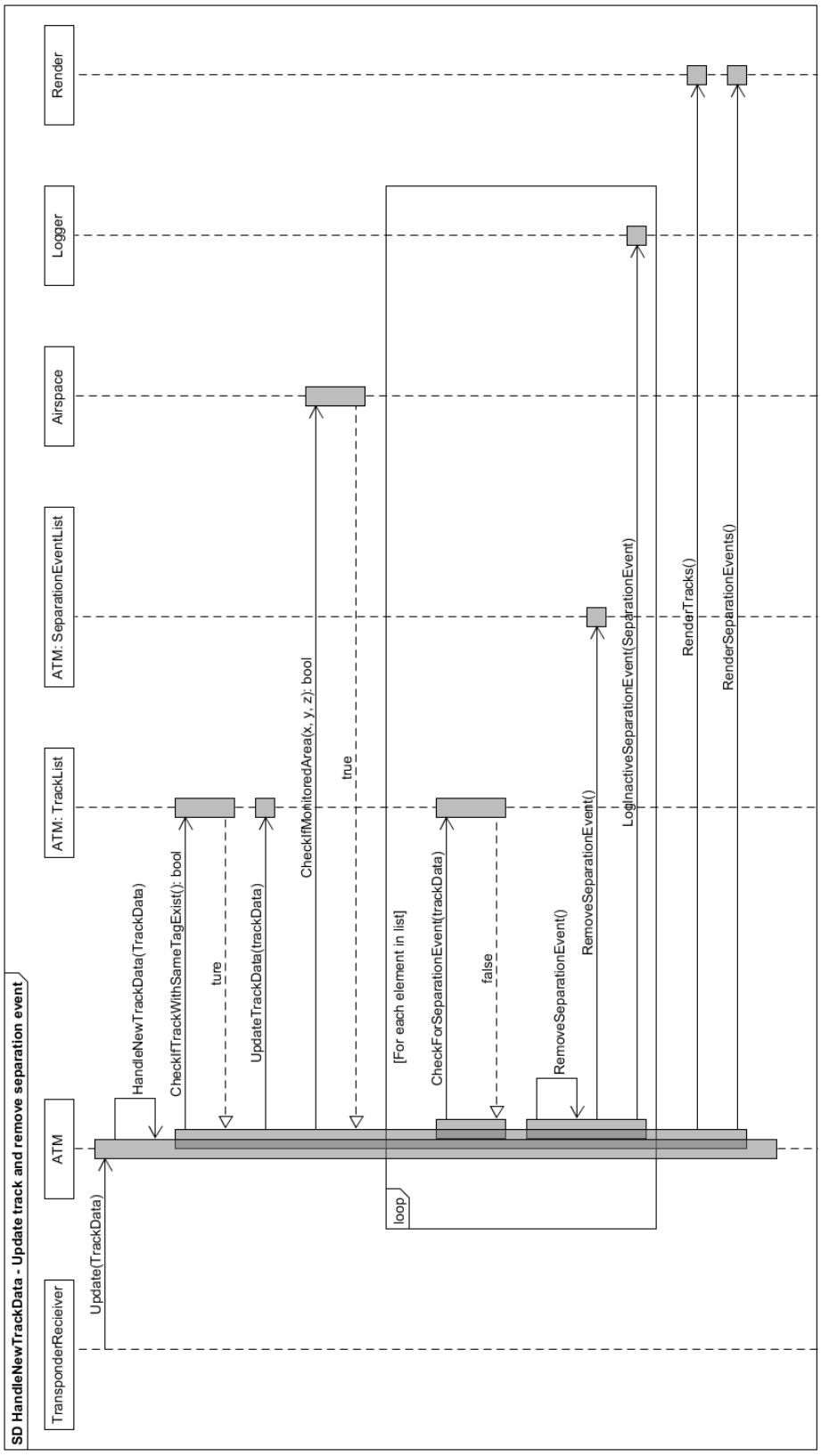
Appendix A4 – Sekvensdiagram: Ny trackData modtages, og der opstår en ny Seperation Event



Appendix A5 - Sekvensdiagram: Ny trackData modtages, og Seperation Event eksisterer allerede



Appendix A6 - Sekvensdiagram: Ny trackData modtages, og Seperation Event fjernes



ATM – Del 2 – Handin 3

Jenkins Projects and Repo URL

Jenkins Unit	http://ci3.ase.au.dk:8080/job/SWT_10_ATM2_UNITTEST/
Jenkins Integration	http://ci3.ase.au.dk:8080/job/SWT_10_ATM2_INTEGRATION/
Jenkins Coverage	http://ci3.ase.au.dk:8080/job/SWT_10_ATM2_COVERAGE/
Jenkins Static Analysis	http://ci3.ase.au.dk:8080/job/SWT_10_ATM2_STATICANALYSIS/

Github repo	https://github.com/mathiasfriis/SWT_10_ATM_part_2

Refleksioner

Softwarearkitektur

Resultat af feedback fra Handin-1
På baggrund af feedback fra Handin-2 afleveringen af ATM del 1 har gruppen i starten af design-processen af ATM del 2 besluttet at vores ATM klasse har haft behov for at skulle følge Single Responsibility-princippet bedre. Dette vil også medføre et system og en klasse som er nemmere at kunne teste.

Yderligere havde gruppen misforstået kravet om inkludering af speed og course kravet til HandIn-1. Derfor er denne funktionalitet nu implementeret og derfor en del af trackData og hele systemet ved denne handin.

Render og logger
Yderligere har gruppen valgt at refraktorere render- og logger-klasserne således at disse nedlægges. I det nye design ligger formateringen af renderingen og logging i trackdata, hvorefter stubbene consoleOutput og fileOutput håndterer selve udskrivning i hhv. konsollen og logging-filen.

Event-funktionalitet
For at kunne tilgodese den nye ønskede event-funktionalitet i ATM-systemet har gruppen valgt at lave en abstrakt event klasse: Event, samt 3 konkrete event-klasser: SeperationEvent, TrackEnteredEvent og TrackLeftEvent. Det resulterer i at vores kode og solution er mere testbar, da vi i systemet forsætter med at implementere ny funktionalitet med lav kobling. Samme tankegang blev brugt til at implementere Airspace klassen og andre i Handin-1.

Rendering af consolen
Med et krav om at alle events der sker på nuværende tidspunkt skal renderes på consollen, samt at vi nu har en del event-muligheder: SeperatonEvent, TrackEnteredEvent og TrackLeftEvent, er det en nødvendighed at vi får opdateret consollen oftere end når der blot kommer ny trackdata fra TransponderReceiveren. Derfor har gruppen implementeret en timer, som sørger for at opdatere renderingen af events hvert 50. millisekund, således vi er sikker på at vi har den nyeste og aktuelle information om de aktive fly i airspacet samt eventuelle events.

Alle design-diagrammer findes under Appendix i dette dokument.

Integrationstest og strategien for denne

Gruppen startede med at lave et dependency-tree, efter at designet for de kommende ændringer til ATM-systemet var på plads.

Undervejs i implementeringen var der dog flere forskellige ting som var nødvendige at ændre undervejs, som vi kender det fra iterative processer. Dette medførte at dependency-tree'et også skulle kigges på igen, således at dettee stemte overens med vores nuværende design af ATM-systemet.

Herefter har gruppen skulle revurdere hvorvidt vi ønskede at bruge en Bottom-Up-Plan, Top-Down-Plan, Big Bang og Sandwich.
Normalvis vil Sandwich umiddelbart være et godt valg til at lægge plan over integrationstest. I og med at vi havde valgt at omstrukturere dele af systemet, valgte vi dog en Bottom Up-approach. Dette skyldtes at vi relativt hurtigt kunne foretage ændringerne på "Low level"-modulerne, og derfor kunne begynde at teste disse samtidig med at andre i gruppen kunne arbejde på "Top level"-modulerne.

Derudover er Bottom Up-approached også nemt at forholde sig til, og da vores erfaring med integrations-test stadig er begrænset var det dét approach vi som gruppe følte os mest tilpas ved at bruge.

Dependency-tree, og en tilhørende tabel over hvilke moduler der testes i hvilke dele af testen, findes under Appendix 8.

Ved inspektion af Dependency-diagrammet kan det ses at flere af forbindelserne mellem modulerne allerede implicit er testet gennem Unit Test af modulerne. Dette har været med til at gøre Integrationstesten relativt overkommelig.

Arbejdsfordelingen

Vi har i gruppen bestræbt os efter at udarbejde arbejdsfordelingen således at enhver ny funktion der er blevet implementeret, som følge af den nye udleverede kravspecifikation, er blevet implementereret og testet af samme person. På denne måde, sikrer det at alle i gruppen får erfaring med hele forløbet af en test driven udviklingsproces.

Denne tankegang er fulgt videre over i vores brug af Jenkins som på samme måde er blevet vedligeholdt af alle i gruppen, hvor den hovedansvarlige for at bygge-jobs'ne var aktive hele tiden skiftede således alle opnåede en ønsket erfaring hermed.

Integrationstestene har i vores arbejdsproces været under revidering flere gange, grundet overvejelser om hvorledes afhængighederne i systemet udartede sig. Dette har resulteret i at flere af de skrevne unittest har kunne gøre det ud for integrationstesten på et givent niveau. Dette har gjort at arbejdsfordelingen fra unittest processen er ført videre med over til integrationstestene.

Af vores repository vil denne arbejdsfordeling ligeledes være tydelig i vores commit tree, der viser et jævnt flow af commits, fra alle gruppens medlemmer.

Konklusion

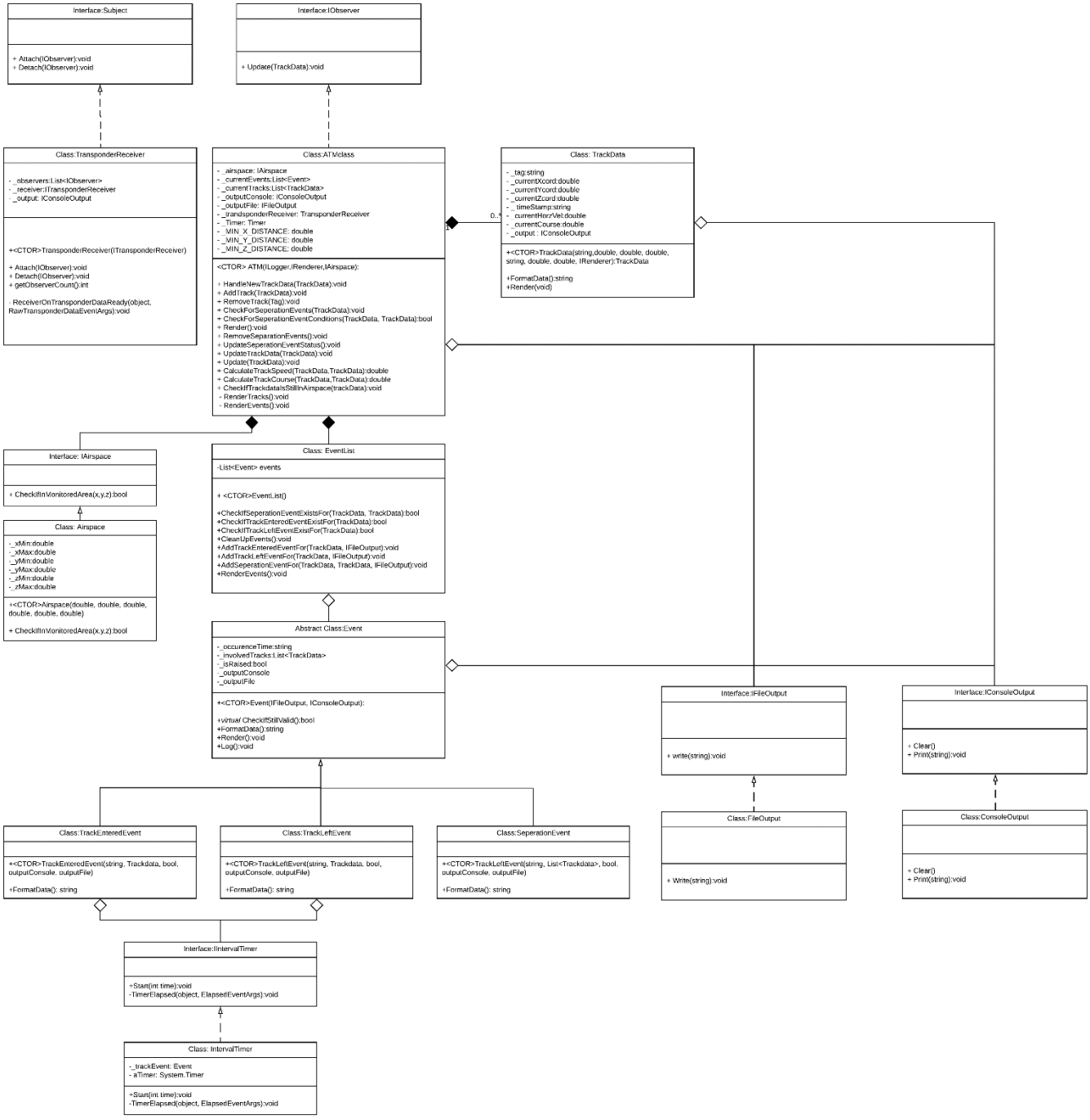
Vi har i denne opgave, skulle udvide vores tidligere aflevering med funktionalitet, hvor vi har haft speciel fokus på integrationstest. Vi startede med at lave vi et dependency-tree over vores system, for at skabe overblik. Det resulterede i at vi omstrukturerede vores system for at gøre det mere medgørligt i forhold til en integrationstest. Ansvaret for at render og logge blev uddelegeret til TrackData og Event-klasserne, for at undgå en Big-Bang Test under integrationstesten. Integrationstesten endte med at bestå af tre trin, som vi udførte via Bottom-Up-Plan. Valget af Button-Up-Plan faldt på at vi i gruppen, grundet omstrukturering af systemet, havde mulighed for at teste integrationen mellem modulerne nederst i Dependency-tree'et, samtidig med der blev arbejdet på modulerne højere oppe.

Arbejdet i gruppen har været ligeligt fordelt, hvor alle i gruppen både har været inde over implementeringen den nye funktionalitet samt integrationstesten. Via github har uddeling af arbejdsopgaver været nemt, og jenkins har hjulpet med til at holde overblikket, selvom gruppen ikke altid har siddet sammen og arbejdet.

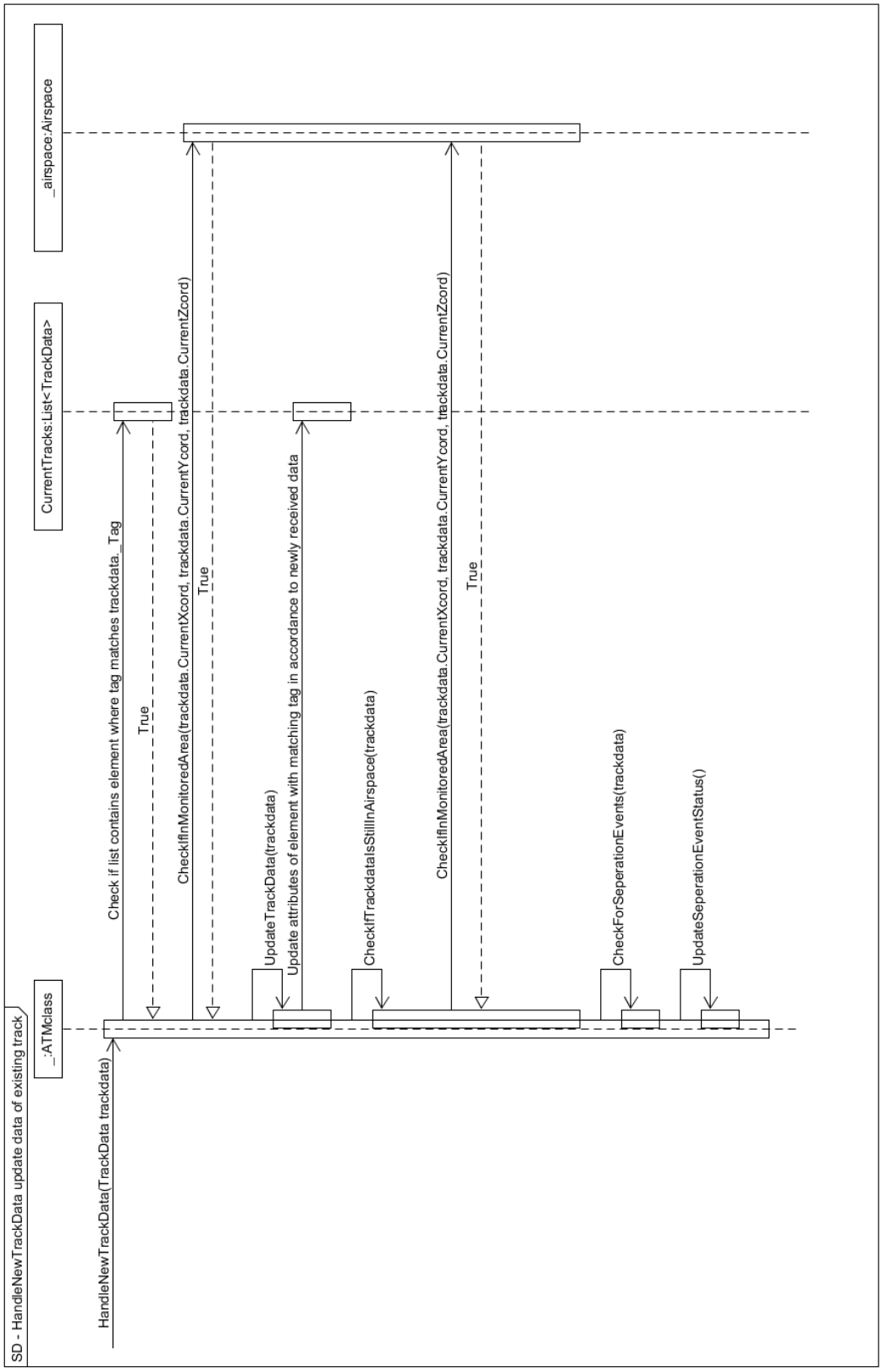
Alt i alt er gruppen meget tilfreds med resultatet af arbejdet. Ingen i gruppen har tidligere haft erfaringer med integrationstest, som har været en spænende udfordring. Yderligere arbejde med unittest har gjort os endnu mere erfarne på det område.

Appendix B

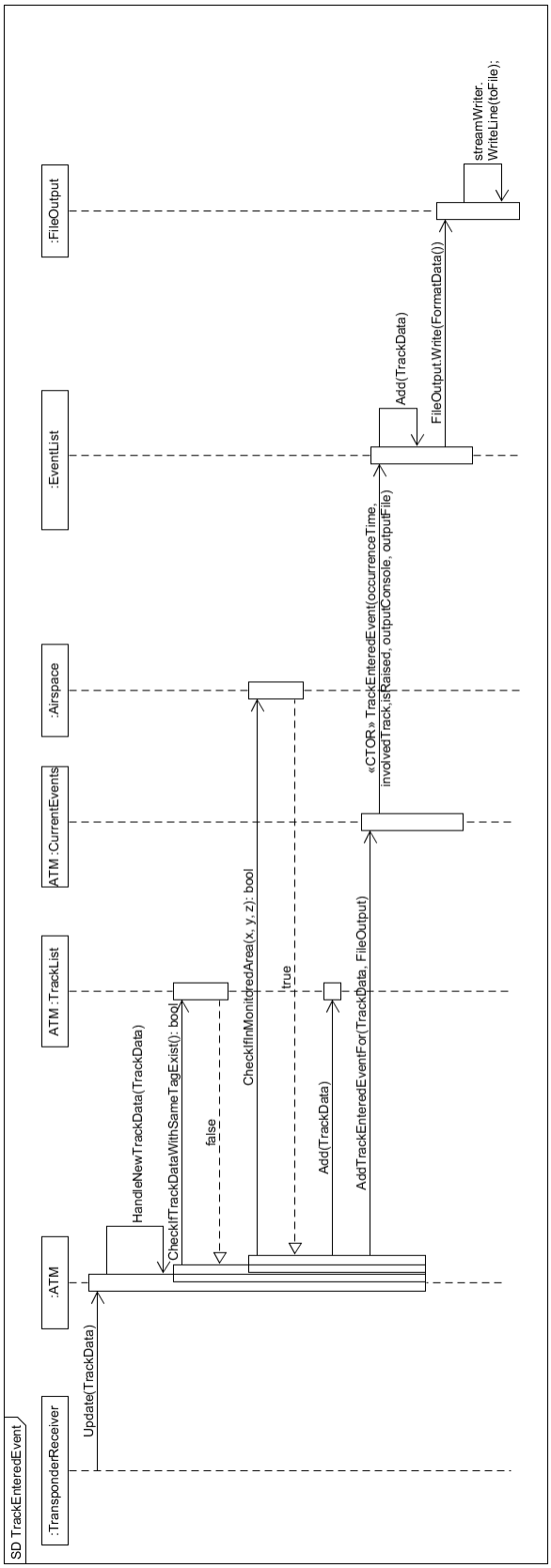
Appendix B1 – Klassediagram udkast



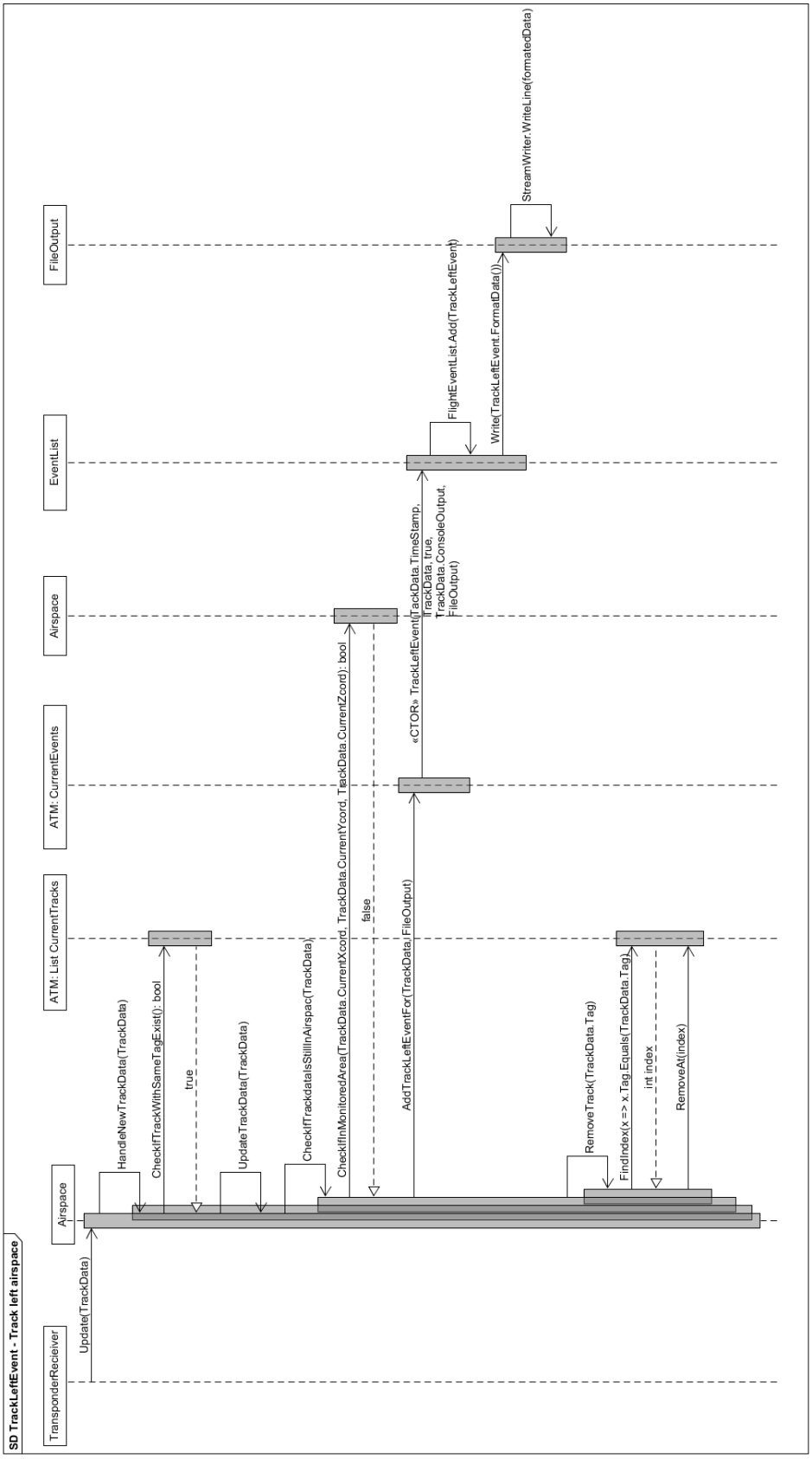
Appendix B2 – Sekvensdiagram udkast for handleNewTrackData



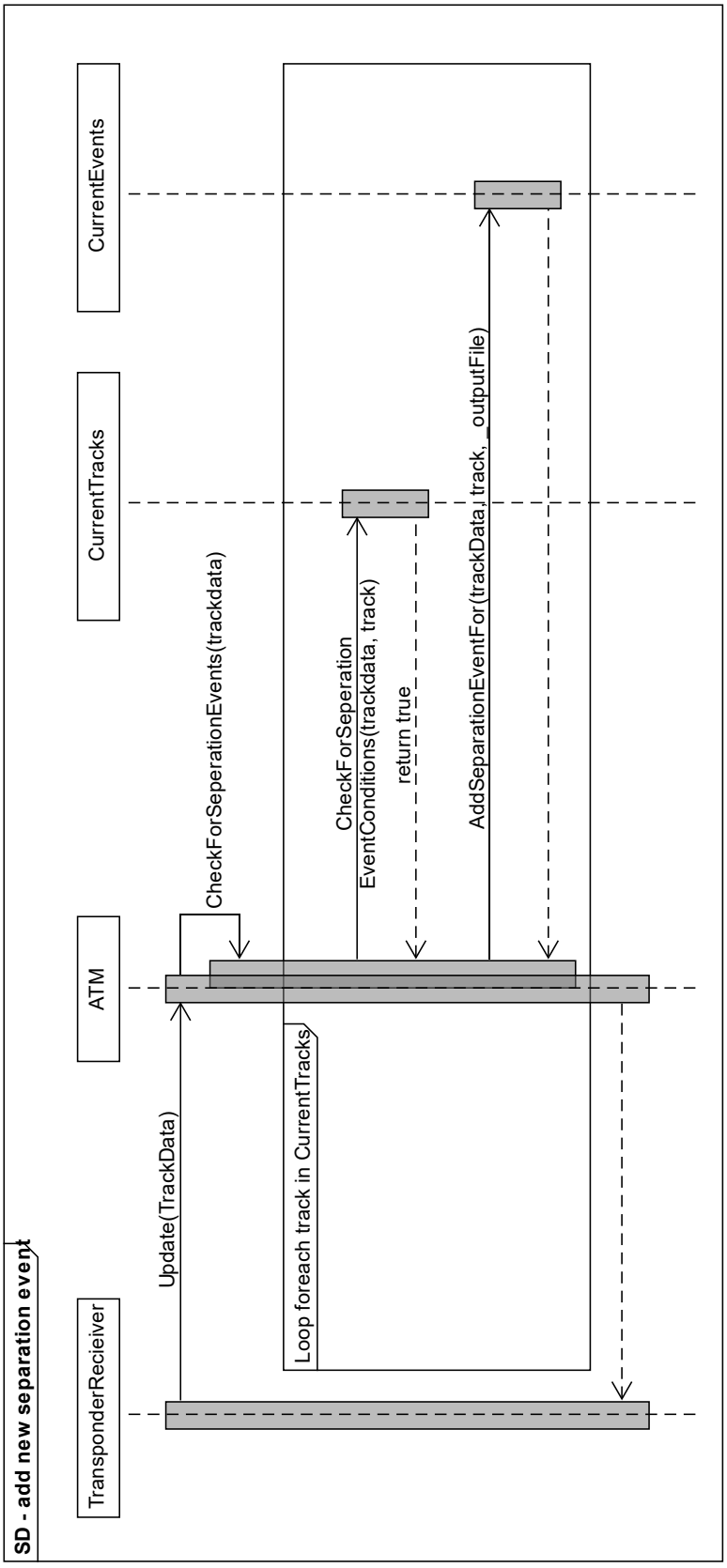
Appendix B3 – Sekvensdiagram: Ny trackData modtages, og der opstår et trackEnteredEvent



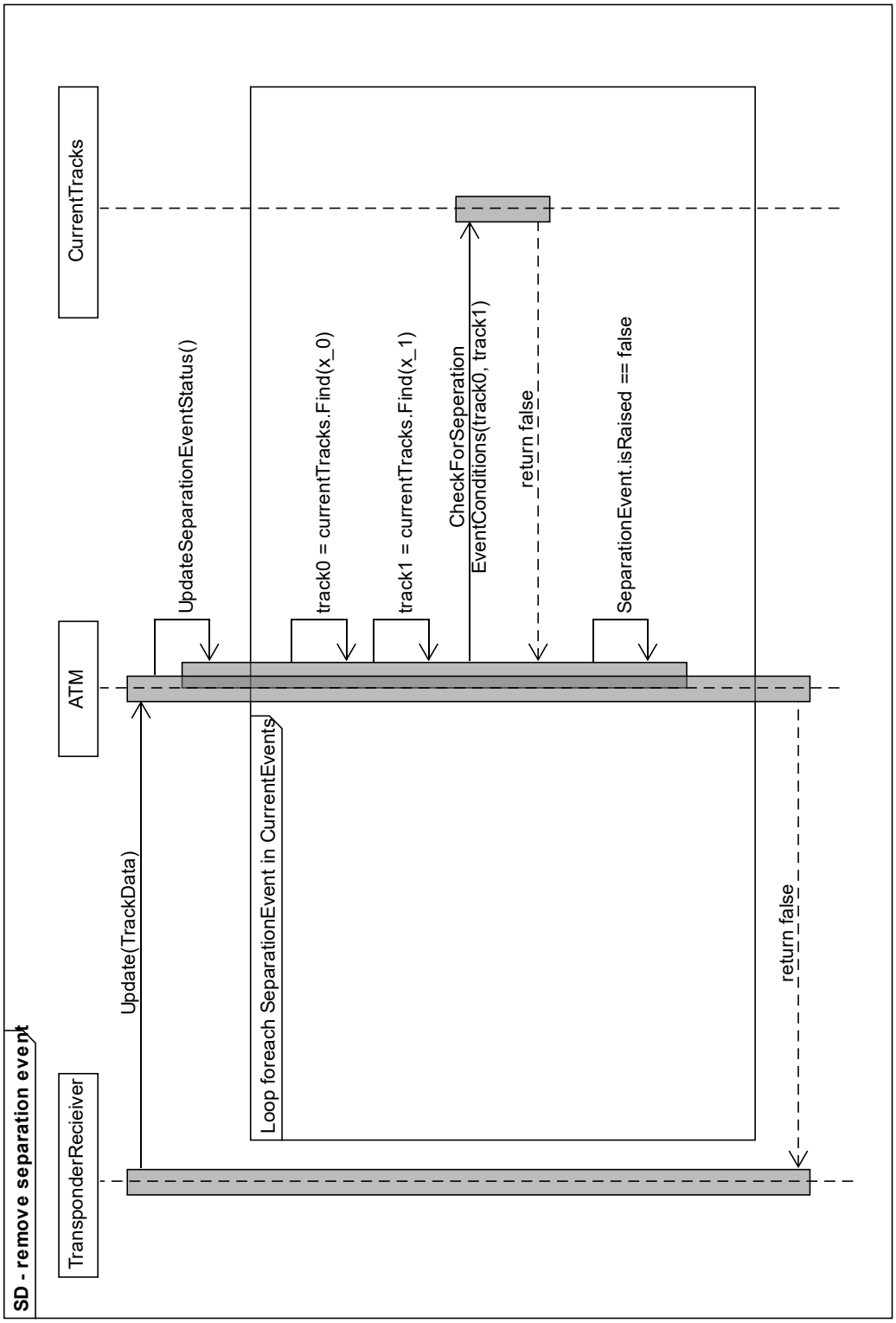
Appendix B4 - Sekvensdiagram: Ny trackData modtages, og der opstår et trackLeftEvent



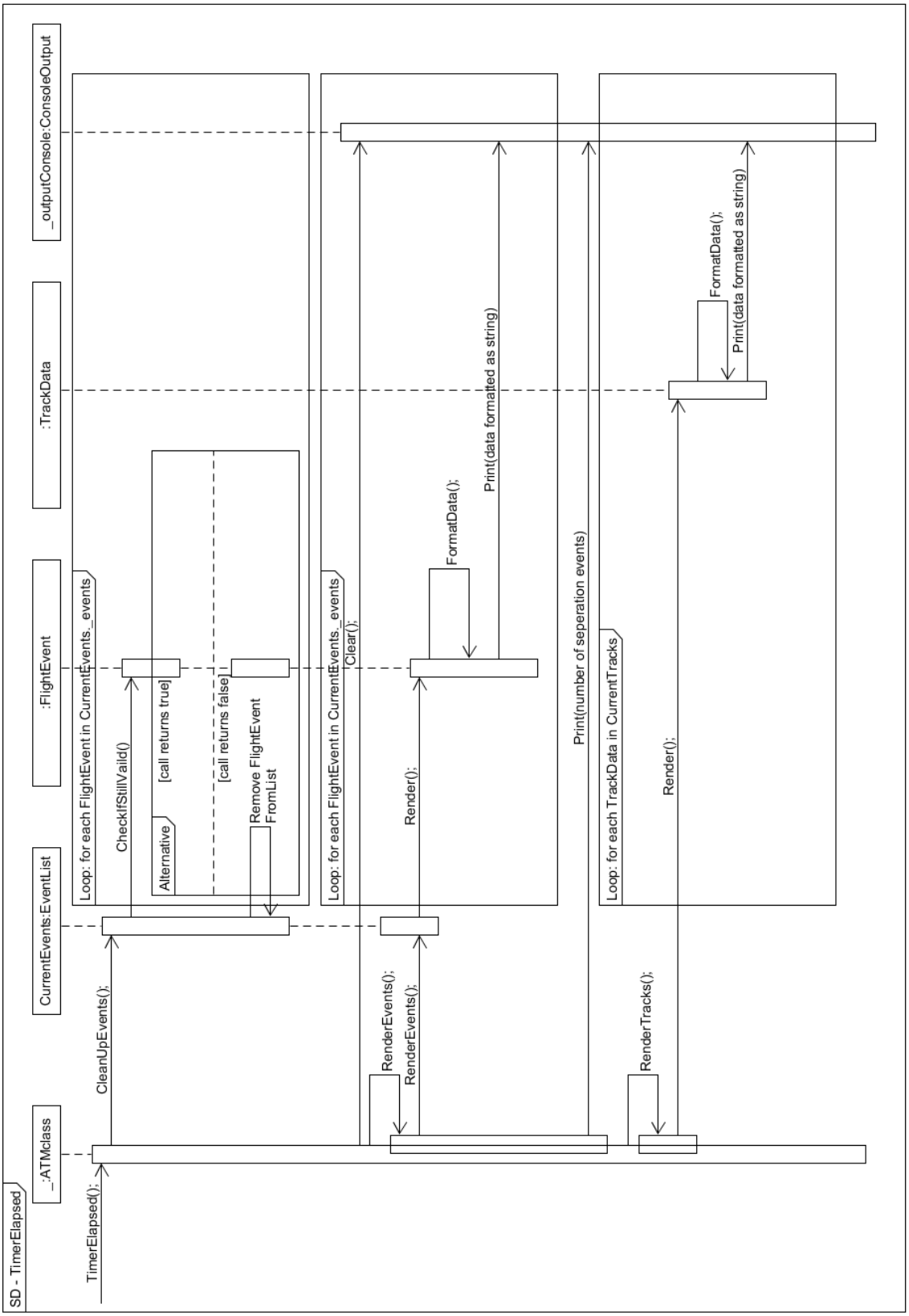
Appendix B5 - Sekvensdiagram: Ny trackData modtages, og Seperation Event opstår



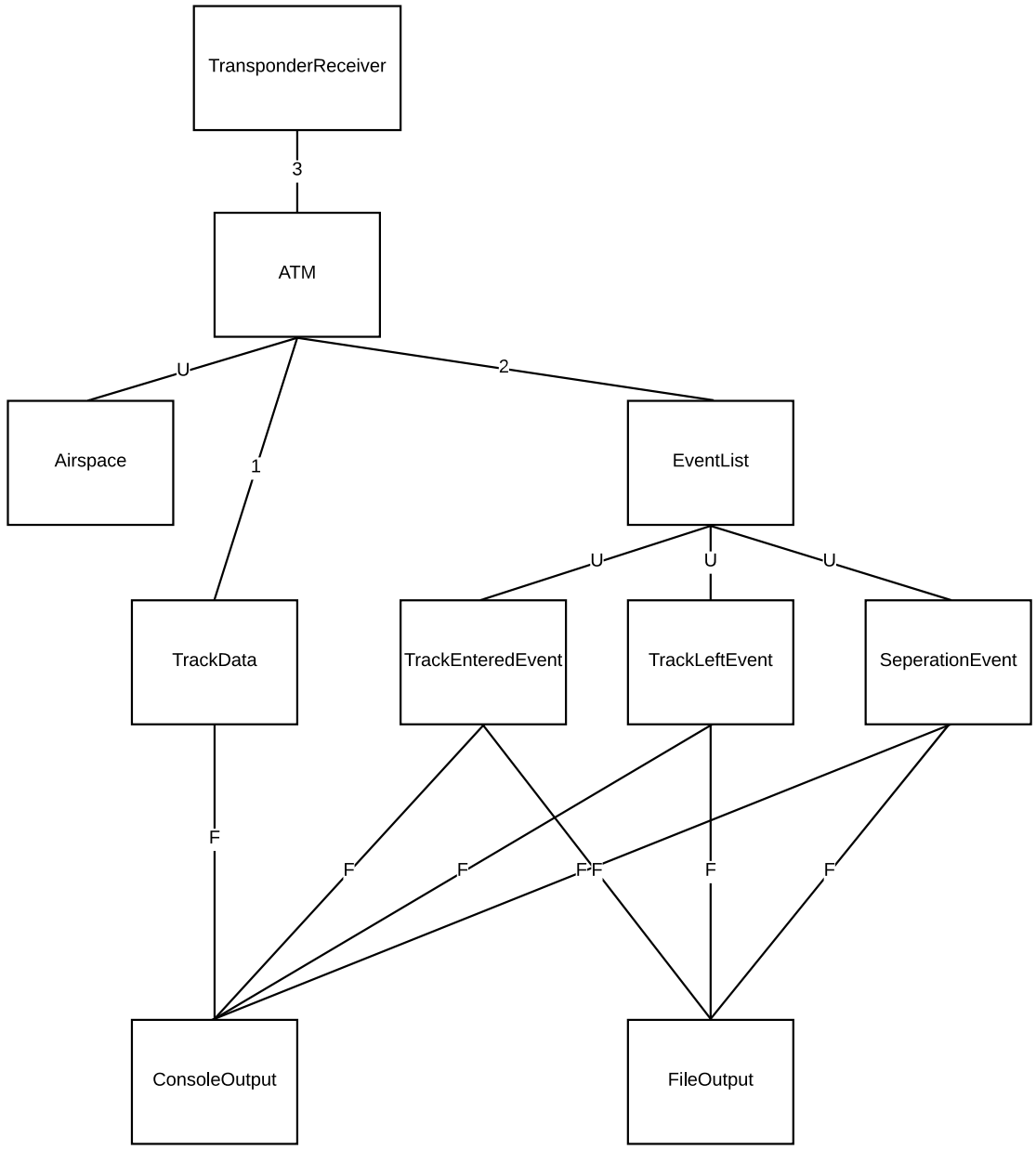
Appendix B6 - Sekvensdiagram: Ny trackData modtages, og Seperation Event fjernes



Appendix B7 – Sekvensdiagram: TimerElapsed



Appendix B8 – Part 2 Dependency Diagram



S=Stub
T=Module that is driven/Acted upon
X=Module under test

Test no./Module	*FileOutput*	*ConsoleOutput*	*TrackData*	*SeperationEvent*	TrackEnteredEvent	*TrackLeftEvent*	*EventList*	*ATM*	*Airspace*	TransponderReceiver
1	S	S	X	X	X	X	X	T	X	
2	S	S	X	X	X	X	X	T	X	
3	S	S	X	X	X	X	X	X	X	T
F	X	X	X	X	X	X	X	X	X	T