AU Engineering

I4SWT Mandatory Exercise

Air Traffic Monitoring

Team 16-1-6

Name	**Student ID**	**E-mail**
Ao Li	*201407737*	liao0452@gmail.com
Cecilie Bendorff Moriat	*201405949*	ceciliemoriat@gmail.com
Jonas Møgelvang Hansen	*201407199*	jonas_jmh@gmail.com
Morten Sand Knudsen	*201270955*	mortensandknudsen@gmail.com

CI build jobs

Unit tests:
`http://ci1.ase.au.dk:8080/job/Team%2016-1-06%20ATM%20(Unit%20Test)`

Integration tests:
`http://ci1.ase.au.dk:8080/job/Team%2016-1-06%20ATM%20(Integration%20Test)`

Code metrics:
`http://ci1.ase.au.dk:8080/job/Team%2016-1-06%20ATM%20(Code%20Metrics)`

April 20, 2016

Contents

1 Introduction

The purpose of this journal is to reflect upon the design, implementation and test of the Air Traffic Monitor system.
The exercise required not only a working system, but a special effort had to be made to obtain a generic design with an appropiate amount of tests which should be simple to maintain if changes in the exercise requirements were to be made.

2 Design

As earlier stated the design of this solution was given thought as it had to be extensible and adaptive to changes in requirements. This section describes the process of obtaining such design and the outcome of the reflections.

2.1 Design considerations

An effort were made to design the system based on the five basic principles of object-oriented programming and design, SOLID. These principles applied to a system tend to make this maintainable and extendable.

To further create abstraction from the provided .DLL and follow the Dependency-Inversion principle, the ATM makes use of a modified repository-pattern[1]. This allows for a quick change in the data-source to a database for example. The modification only allows for reads in the repository, the implementation can be seen in the 'IReadOnlyRepository<TModel>' interface.

Two sequence diagram for the back-end can be seen in figure **??** and **??**. Where the repository-pattern is used to connect the monitor to the datasource, in this case the provided .dll.

The Console Application follows the Single Responsibility to certain degree, where our Notification- and MonitoredTrackInfoDisplayer only has one responsibilty; Formatting data to construct a List of strings. The way those two classes are implemented involves The Open Close principle aswell. They both uses an interface IRenderable; It opens several advantages towards test-, maintain- and extendability, and aswell a easier code to read.

2.2 Implementation

is this nhow i make comments???

It works

[1] `https://msdn.microsoft.com/en-us/library/ff649690.aspx`

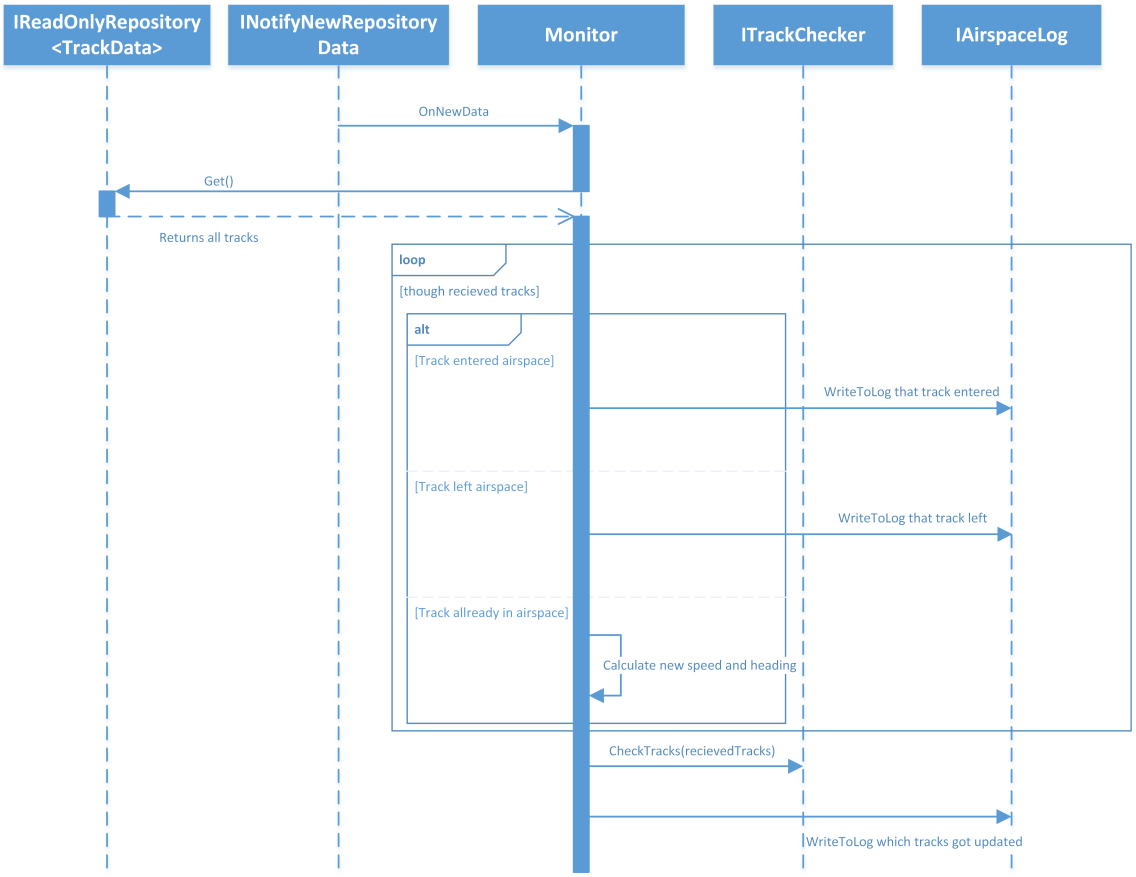

Figure 0.1. Sequence diagram for monitor

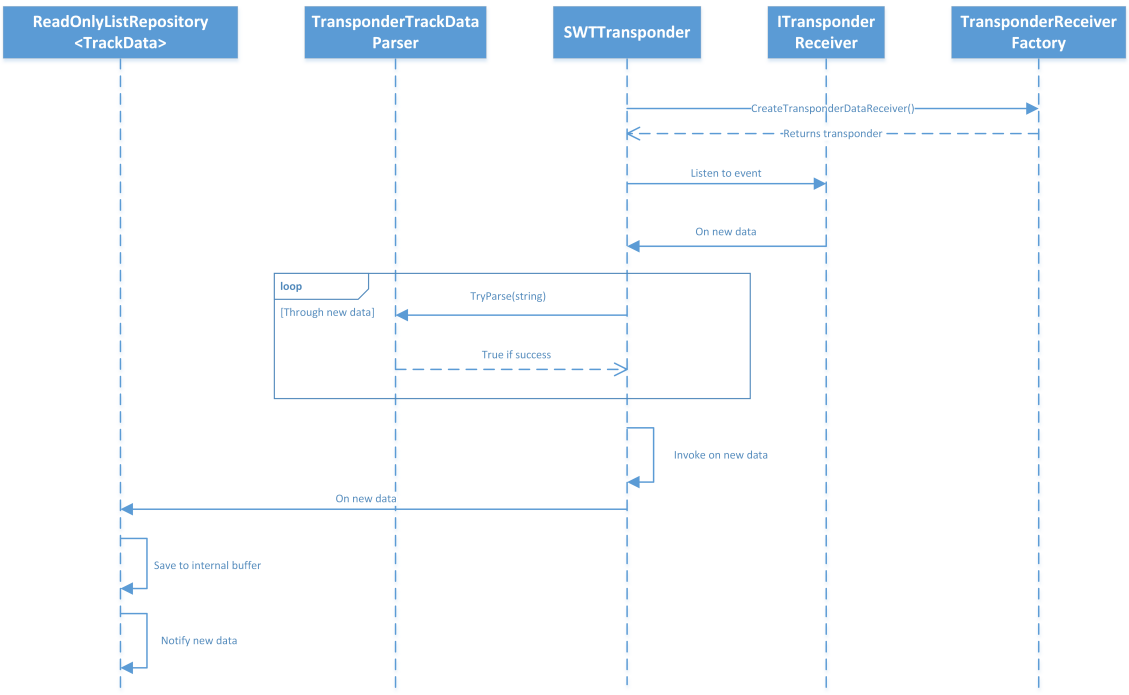

Figure 0.2. Sequence diagram for data acquisition from the transponder .DLL

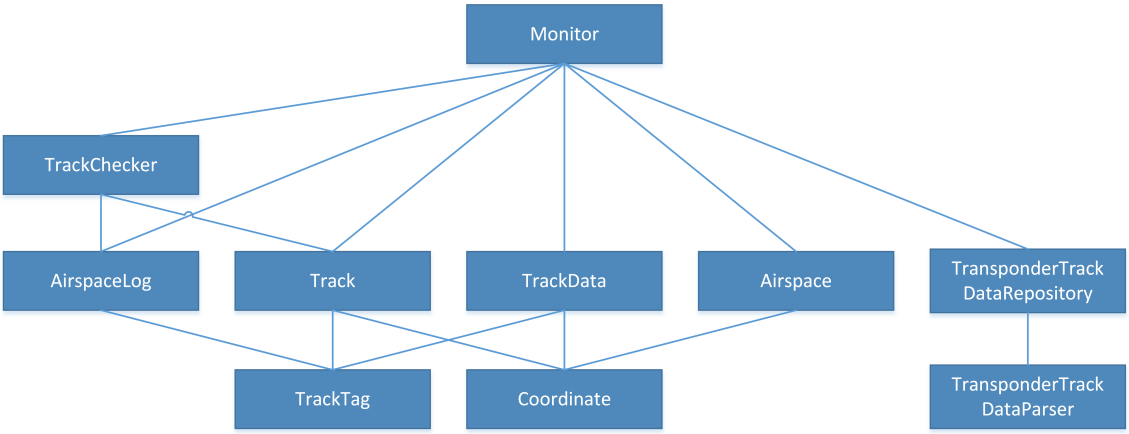

Figure 0.3. Dependency tree

3 Test

3.1 Strategies

4 Teamwork

In this section the teamwork is described. The requirements were 3-4 people in a group and no more than two persons should share a computer while programming. Another requirement was the use of *Continuous Integration*, which helps the developers commit to a shared build server multiple times during the development process.

4.1 Strategies

We have earlier tried working with some of the strategies from Extreme Programming which were found suitable for this project. One of these was *Pair programming*. Code is then written by pairs which shares the workstation. One will be in control of the keyboard and write the code while the other will watch the code and work towards the best implementation. The pair switches place every now and then. This ensures that both programmers are engaged in the software.

4.2 Continuous integration

As the group was divided into two pairs of developers each working on classes of their own, the continuous integration helped the groups gaining a shared understanding on the software development process.

5 Conclusion