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 0.2 and **??**. Where the repository-pattern is used to connect the monitor to the datasource, in this case the provided .dll.

2.2 Implementation

It works

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

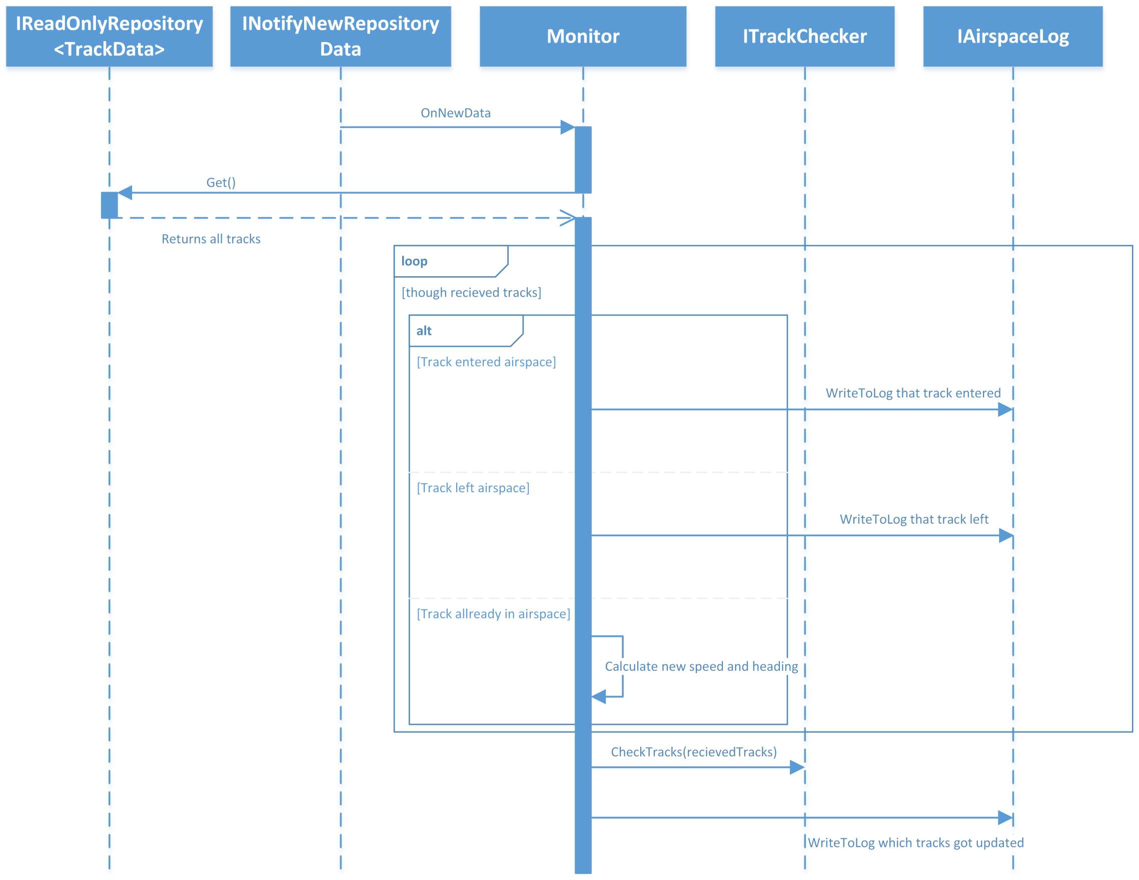
IReadOnlyRepository <TrackData>
INotifyNewRepository Data
Monitor
ITrackChecker
IAirspaceLog
OnNewData
Get()
Returns all tracks
loop
[though recieved tracks]
alt
[Track entered airspace]
WriteToLog that track entered
[Track left airspace]
WriteToLog that track left
[Track allready in airspace]
Calculate new speed and heading
CheckTracks(recievedTracks)
WriteToLog which tracks got updated

Figure 0.1. Sequence diagram for monitor

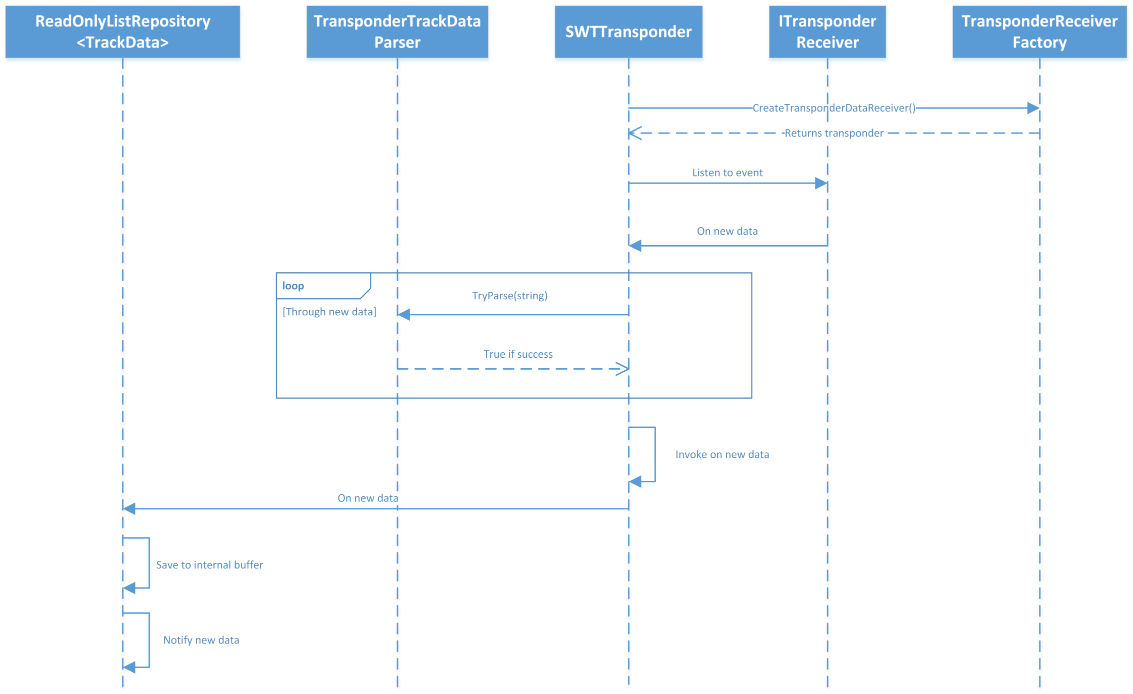

Figure 0.2. Sequence diagram for data acquisition from the transponder .DLL

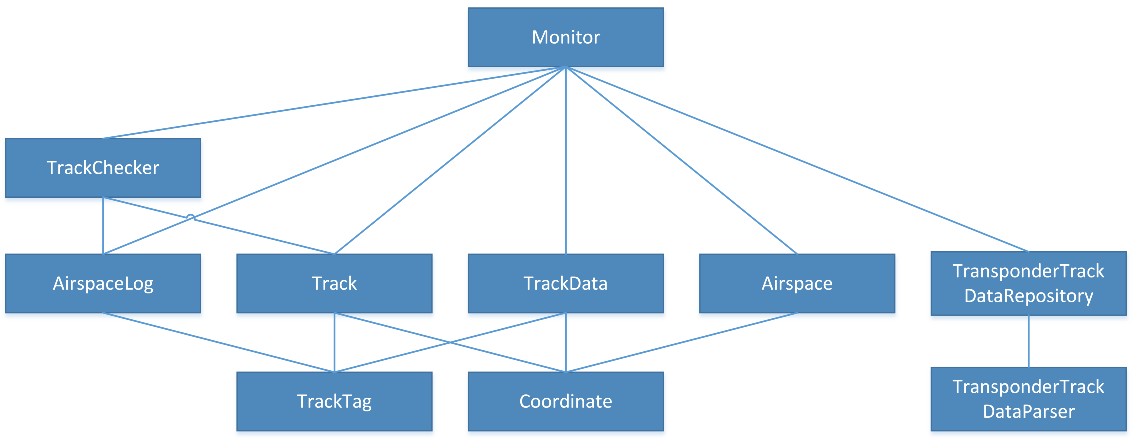

Figure 0.3. Dependency tree

3 Test

3.1 Unit Test

There are parallel with code development been written tests to see if the code reacted as intended.

Jonas and Cecilie chose to use TDD to the development of their code, so there are tests written before the code itself was written, to ensure that all code was covered by the test from the start.

Ao and Morten decided to develop code first, then write their test . This was the choice as there were some difficulties with TDD for them. It worked fine. However, this probably took a little longer than if you had used TDD, as there were some areas of the code you forgot to get tested at the start

3.2 Integrations Test

Integration testing is used to ensure that all our classes work together, after we have been in two teams and made code to the ATM.
Below are our dependency tree for our ATM system

Figure 0.4. Dependency Tree for ATM

After making all of our unit tests we designed a dependency tree to find out how we should write our integration test. The Dependency tree shows the integration and interdependence of our different classes. We have chosen to use a bottom-up integration. The choice behind this is that a lot of our test and integration test lies in our unit test. Therefore, all the lower classes is tested and work properly before being coupled to a different class.

3.3 Jenkins

There has previously been described how CI fungrerer as a group tool.
The way we have chosen to use it is via Jenkins, a program that ensures CI on our repository. When comiting code to the repository Jenkins will first run a build that looks at all the unit tests, if this goes well, Jenkins will subsequent run a project that tests all integration test, once approved it will finally run a project that looks at our code metrics.

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

Earlier experience from working with some of the strategies from *Extreme Programming* in the course I4SWD, this was found suitable for the software development of the Air Traffic Monitor too.
One of these strategies 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 two groups to gain a shared understanding on how the software development progress.
Another benefit was the automatic generated code coverage report and software quality metrics which were used to determine whether the written software and tests were satisfying. If not, it was easy to gather information where the code standards should be optimized for better statistics.
As with every other git project a version history was obtained making it simple to revert to a previous build if changes caused a broken build.

5 Conclusion

The process of developing the Air Traffic Monitoring system and compile this journal was made in two weeks. During these weeks multiple strategies from the courses I4SWT and

I4SWD were used to satisfy the requirements given.