

# Air Traffic Monitoring

## I4SWT Mandatory exercise

### 1 Introduction

In this exercise, you are going to create a system, which can monitor and present the air traffic in a given region and detect interesting events in the air traffic.

This note contains information on the following:

- A description of the problem domain, including terminology, for the exercise
- A description of the exercise (terms, resources, etc.)
- Requirements for the ATM system
- A time schedule for the hand-in of your exercise
- Rules etc. pertaining to your hand-in.

**NOTE:** This exercise is *mandatory*. This means that it is a prerequisite for your admission to the exam of this course that you have handed this exercise in and have a grade of **PASSED** (see details below).

You shall hand in in groups – no solo work. The guidelines for group work are stated in the note “Teams, lab exercises and hand-ins in I4SWT.pdf” available on the course Blackboard site under Lecture 01.1.

#### 1.1 Problem domain



Figure 1: Air traffic. Each yellow or orange icon represents a commercial flight (flightradar24.com)

The airspace, particularly over the northern hemisphere, is crowded. A traditional radar picture showing “blips” of aircraft within a given region can easily become very complex, task-saturating the air traffic controller (ATC) and lead to dangerous situations.

To alleviate the ATC’s task and reduce the reliance on traditional radar systems, all commercial and military aircraft are equipped with a *transponder*, which actively and continuously transmits the aircraft’s position, speed, altitude,

etc. The transponder signals are received by a *transponder receiver* (e.g. at an airport<sup>1</sup>) which decodes the radio signal and forwards the contained information to a system like the ATM and displayed as a “Track”. This allows the air traffic controller to read a lot of information on the individual aircraft instead of remembering it or looking it up.

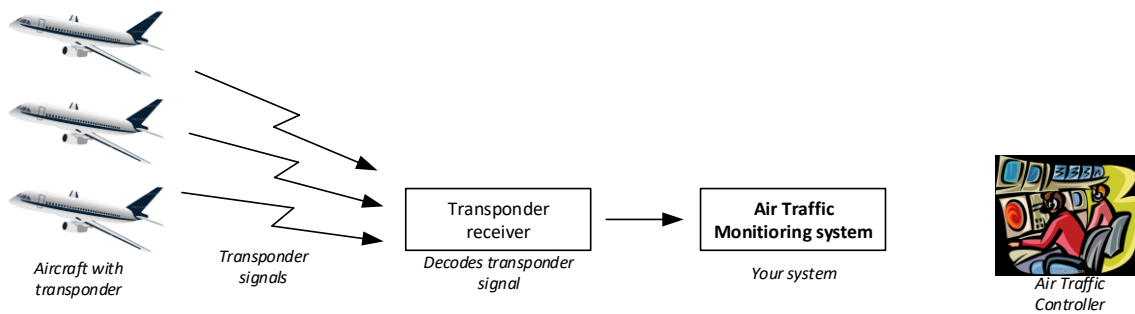


Figure 2: The ATM system's context: Aircraft, transponders, their signals, the transponder receiver, the ATM and the ATC

## 2 System requirements

The ATM system shall adhere to the following requirements:

### 2.1 Track rendition:

1. The system shall monitor one (1) airspace.
2. The system shall render all tracks currently in the monitored airspace.
3. The system shall not render any tracks that are outside the monitored airspace.
4. Rendition of the tracks in the airspace shall be either to a file or to the console<sup>2</sup>
5. The monitored airspace can be considered a box which is defined by its minimum and maximum coordinates and minimum and maximum altitudes, both in meters:
  - a. The south-west corner is at coordinate (10.000, 10.000)
  - b. The north-east corner is at coordinate (90.000, 90.000).
  - c. The lower altitude boundary of the monitored airspace is 500 meters and the upper boundary is 20.000 meters.
6. All tracks shall be rendered every time new transponder data is received.
7. Each track shall be displayed with the following data:
  - Tag (text string, 6 characters)
  - Current position (x-y, both in meters)
  - Current altitude (meters)
  - Current horizontal velocity (m/s)
  - Current compass course (degrees, 0 degrees is north)

### 2.2 Event detection, logging, and rendition:

8. When transponder data is received, the data shall be investigated for the occurrence of a “separation” event (see below)

<sup>1</sup> The reception of transponder signals is not limited to ATC centers. As a matter of fact, the signals can be received with a simple [USB dongle](#) (US\$ 6.49) and shown on your computer screen – or, one may access [flightradar24.com](http://flightradar24.com) to view the transponder signals mapped on geographical areas.

<sup>2</sup> You may create a more sophisticated rendition such as a Graphical User Interface, but it is not required and will not give extra credits

9. All occurrences of separation events shall be logged to a file.
  - Time of occurrence
  - Tag of the involved tracks
10. All current events shall be rendered onto the screen.
11. Only current events shall be rendered onto the screen, i.e., no history of events shall exist in the rendition of events.

#### 2.2.1 Details of the “Separation” event

12. If the vertical separation between two tracks is less than 300 meters, and the horizontal separation of two tracks is less than 5.000 meters at the same time, the tracks are deemed as *conflicting*.
13. When two tracks become conflicting a “Separation” event shall be raised.
14. A “Separation” event shall remain raised as long as the two tracks are conflicting.
15. When two conflicting tracks are no longer in conflict, the associated “Separation”-event shall be deactivated.
16. The rendition of a “Separation” event shall include the tag of the involved tracks and the time of occurrence.

### 3 Exercise solution requirements

Your task in this exercise is to design, implement and test an ATM system. You must also document what you have done in a journal of the exercise. By doing this, you must demonstrate that you are proficient in both the course theory and the tools we have covered so far in the course.

The exercise forms the basis of the exam, which means two things:

- You must have your hand-in approved to be eligible for examination
- The more effort you put into this exercise, the better the result will be at the exam – this is a historical fact.

#### 3.1 General requirements

There are certain requirements to your way of working:

- You must work in teams of 3-4 members of this system. Not 2, not 5 (unless explicitly allowed by the teacher of the course)
- You must split the responsibility of implementation and testing of the system classes between you - no 3-4 man peer programming!
- You must use a Git repository to share and version-control your code.
- You must use continuous integration to run your tests etc. The CI server should be used in accordance with the guidelines for continuous integration, e.g. “frequent commits”.
- For the sake of unit testing, the core functionality is best encapsulated in a class library with your classes, but there must also be an application (a console application is enough, see footnote 2 above) that integrates all of the required functionality.

#### 3.2 Requirements for continuous integration

You must set up at least 3 dependent build jobs<sup>3</sup>:

1. Unit testing build job – runs all unit tests of your project, and code coverage.
2. Integration testing build job – Runs all integration tests of your project.
3. Code metrics & quality build job – runs all code metrics and quality tests of your project (e.g .FxCop, Code Metrics, ...)

---

<sup>3</sup> “Dependent”: When one job finishes successfully, the dependent build job(s) execute.

### 3.3 Requirements for your journal

You are required to hand in a short report (6-10 pages) along with an implementation of your system. The purpose of the report is for you to demonstrate *reflection* on how you designed, implemented, and tested your system. A non-exhaustive list of questions that can help you reflect on the exercise is:

- What software design did you arrive at for the system, and how did you arrive at it?
- How did you divide the software classes between group members for implementation and test? Why did you divide it as you did?
- What strategy did you select for your integration test? Why?
- How did the use of a CI server help you – did it help you at all? How/why not?

In general, describe anything good or bad you have learned from this exercise in the report.

Apart from this, the report *must* contain:

- A front page identifying the following:
  - Your team number
  - All participants in the team (name, study number and email address)
  - The URL of the GitHub repo with your code
  - The URL of each of the CI build jobs of your project.
- Class diagrams and sequence diagrams to explain the design, structure, and behavior of your system
- A dependency tree for your software to support your integration tests.

## 4 Resources

This section lists the resources available to you for the solution of this exercise.

### 4.1 Transponder Receiver

You are provided with a C# class library (DLL) representing the driver for a *Transponder Receiver*, i.e. a driver for hardware that is capable of receiving transponder data. You can create a new *ITransponderReceiver* object by means of a factory and sign up to the C# event exposed and start receiving raw transponder data. Using this data, you can derive more information on the individual tracks and detect interesting events as required.

Details of the interface of the *ITransponderReceiver* interface are given below:

#### 4.1.1 ITransponderReceiver

The *ITransponderReceiver* interface contains the following, see below

- A .NET event data type, [RawTransponderDataEventArgs](#)
- A .NET event, *TransponderDataReady*,

To receive transponder data (see Section 4.3), simply subscribe to the *TransponderDataReady* event of an implementation of *ITransponderReceiver*.

```
namespace TransponderReceiver
{
    public class RawTransponderDataEventArgs : EventArgs
    {
        public RawTransponderDataEventArgs(List<string> transponderData)
        {
            TransponderData = transponderData;
        }

        public List<string> TransponderData { get; }
    }

    public interface ITransponderReceiver
    {
        event EventHandler<RawTransponderDataEventArgs> TransponderDataReady;
    }
}
```

## 4.2 TransponderReceiverFactory

The TransponderReceiverFactory class is a factory for objects which implement the ITransponderReceiver interface. Use the static CreateTransponderReceiver() this factory to obtain an ITransponderReceiver object. You only need one object.

```
namespace TransponderReceiver
{
    public class TransponderReceiverFactory
    {
        public static ITransponderReceiver CreateTransponderDataReceiver()
        {
        }
    }
}
```

## 4.3 Transponder data format

As evident from the above, transponder data is provided in “raw” format. When you receive a TransponderDataReady event, the event data will contain raw data on the tracks in a list of several strings. Each string represents a track and contains individual track data items, separated by a semicolon (;). The format is as follows:

Data item	Description	Description
1	Tag	Track tag (text)
2	X coordinate	Track X coordinate in meters
3	Y coordinate	Track Y coordinate in meters
4	Altitude	Track altitude in meters
5	Timestamp	Timestamp of the above data ("yyyymmddhhmmssfff")

As an example, a raw transponder data item containing “ATR423;39045;12932;14000;20151006213456789” can be interpreted as follows:

- Tag: ATR423
- X coordinate: 39045 meters
- Y coordinate: 12932 meters
- Altitude: 14000 meters
- Timestamp: October 6th, 2015, at 21:34:56 and 789 milliseconds

## 5 Evaluation of your hand-in

The evaluation is based on the criteria given in the rubrics for the individual hand-ins. Each evaluation will have 1 of 3 possible grades:

### “PASSED”

Your hand-in is of sufficient quality to pass the criteria for approval

### “FAILED – RESUBMIT”

Your hand-in is of insufficient quality to pass the criteria for approval. You are granted one resubmittal. The requirements and deadline for this resubmittal will be conveyed along with the grade. The resubmittal will be graded either “PASSED” or “FAILED” (no 2<sup>nd</sup> resubmittal)

### “FAILED”

Your hand-in is of insufficient quality to pass the criteria for approval. You are not granted resubmittal.

Note that both parts of your hand-in must be graded PASSED (in either first or second attempt) for the members of the group to be admitted to the exam in the course.