Acronym	Messer
Project	**Aircraft Message Server**
Doctype	**Requirements**
Author	**Kai Warendorf**
Contact	Kai.Warendorf@hs-esslingen.de
Client	Esslingen University
Contact	Faculty of Information Technology
Version	3.0
Date	28. Mar. 2021

Contents

Chapter 1

Project Drivers

1.1 Purpose of the Project

1.1.1 Vision Statement

This project aims at developing a server that provides aircraft messages locally in a Java application.

1.1.2 Project Outcomes

The Java application reads aircraft sentences.

The Java application transforms each sentence into a basic aircraft.

The Java application prints a string representation of each aircraft onto the screen.

1.1.3 Learning Objectives

After having completed this project, as student, you can ...

- develop and integrate Java classes.
- apply the Java extension mechanism.
- perform advanced String transformation operations in Java.
- apply the Java observation/observable pattern.

1.2 Stakeholders

1.2.1 Project Team

Various members and roles.

1.2.2 Product Users

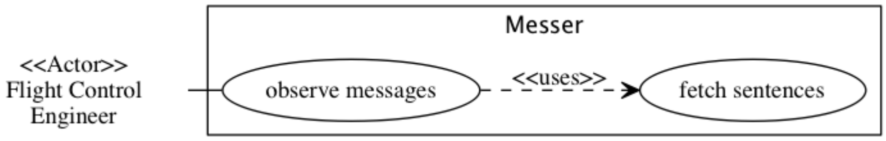
Local Flight Control Engineer, User. Priority: **Key User**.

Chapter 2

Functional Requirements

2.1 Data Model and Data Dictionary

2.1.1 Use Case Diagram



2.2 Messer Functional Requirements

Messer.F.10 Observe Aircraft Messages essential

Feature In order to get an overview of the local flight traffic, as a flight control engineer, I want to be able to observe each incoming aircraft message.

Messer.F.20 Fetch Messages essential

Feature In order to provide aircraft messages locally, the system shall fetch the corresponding sentences from the following web service:
https://opensky-network.org/api/states/all

Feature In order to integrate seamlessly with other OS operations, the web service address shall be provided as input parameter upon application start.

Chapter 3

Non-Functional Requirements

3.1 Look and Feel Requirements

Messer.NF.10 Text Output per aircraft message essential

Feature The system shall display each aircraft message received in the following form (example):

```
BasicAircraft [icao=4ca798, operator=AZA402  , posTime=Sun Jun
16 10:39:20 CEST 2019, 48,33 / 9,2048,33 / 9,20, speed=233.89,
trak =6.19]
```

3.2 Implementation-Specific Requirements

3.2.1 Process

Messer.NF.50 Test Driven Development **essential**

In order to ascertain sufficient testing of the product, the implementation must be carried out following a test-driven development approach.

3.2.2 Architecture

Messer.NF.60 Implementation of Messer **essential**

Feature In order to serve several clients at the same time in terms of a publish/subscribe architecture, the module *Messer*, i.e. the ADS-B Message Server, must be realized following the Observer architectural pat- tern.

Messer.NF.65 Use of Classes **essential**

Feature The organization of the system implementation shall reflect the classes and interfaces shown in the following class definitions:

```
Class Coordinate:
        double latitude;
        double longitude;


Class BasicAircraft:
        String icao;
        String operator;
        Date posTime;
        Coordinate coordinate;
        Double speed;
        Double trak;
        Integer altitude;
```

3.3 Maintainability Requirements

Messer.NF.70 Documentation essential

In order to ascertain high understandability, the source code must be self-explanatory.

Messer.NF.80 Cohesion and Coupling essential

In order to support high maintainability, the modules of the system must be realized with high-cohesion and low coupling.

Messer.NF.90 OO Design Principles essential

In order to support high maintainability, the other well-known principles of good object-oriented design must also be applied.

4.1 JSON Format

An aircraft sentence provided by Lab 1 has the following format:

```
"4402cd",           //icao
"LDM10N  ",         //operator
"Austria",
1560674480,         //posTime
1560674480,
9.5577,             //longitude
48.6497,            //latitude
4259.58,
false,
191.94,             //speed
82.61,              //trak
13.66,
null,
4366.26,
"7713",
false,
0
```

The yellow marked lines come from the aircraft data and must be used.

4.2 How to start

Try to solve the lab in the following order:

1. Implement BasicAircraft.java and Coordinate.java.
2. Create AircraftFactory.java and AircraftDisplay.java:

   a. Use the “sentence” from Lab1 (see 4.1 above) and create a BasicAircraft. Extract the yellow marked values and assign them to the correct attribute.

3. In Senser.java: Make the class observable for an AircraftSentence. Each created AircraftSentence will invoke the setChanged() – notifyObservers(sentence) sequence. This would be in run().
   HINT: In the script examples zip file (in Moodle) under VL7 Pattern project Senser would be the ConcreteSubject
4. In Messer.java:

   a. Create the class, use Senser.java as starting point

   b. Make the class the observer for an AircraftSentence.
   HINT: In VL7 Pattern project Senser would be the PriceObserver

   c. Implement the update method, where you store each sentence in a thread save queue.

   d. In run():
   - Create an object named display from the AircraftDisplay class
   - Create an object named factory from the AircraftFactory class
   - get a sentence from the queue
   - create a BasicAircraft using the factory
   - display it

5. In the starter class: Start Messer and connect Messer and Senser
   HINT: In VL7 Pattern project you find this in TestObservers