Acronym

Wacmop

Project

Web-based Active Aircraft Monitor with Map

Doctype

Requirements

Author

Hans-Gerhard Gross, Kai Warendorf

Contact

hggross@hs-esslingen.de;
Kai.Warendorf@hs-esslingen.de

Client

Esslingen University

Contact

Faculty of Information Technology

Version

1.0

Date

March 2, 2015

Contents

Chapter 1

Project Drivers

1.1 Purpose of the Project

1.1.1 Vision Statement

This project aims at developing an application that shows the active aircraft in range of the ADS-B receiver on a map provided by a web server.

1.1.2 Project Outcomes

The Java application reads ADS-B messages.

The Java application decodes ADS-B messages.

The Java application transforms ADS-B message data into aircraft data.

The Java application displays decoded message data and aircraft data.

The Java application displays the decoded aircraft positions on a map.

1.1.3 Learning Objectives

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

- develop Java Mashups.
- develop simple Java Web applications.

1.2 Stakeholders

1.2.1 Project Team

Various members and roles.

1.2.2 Product Users

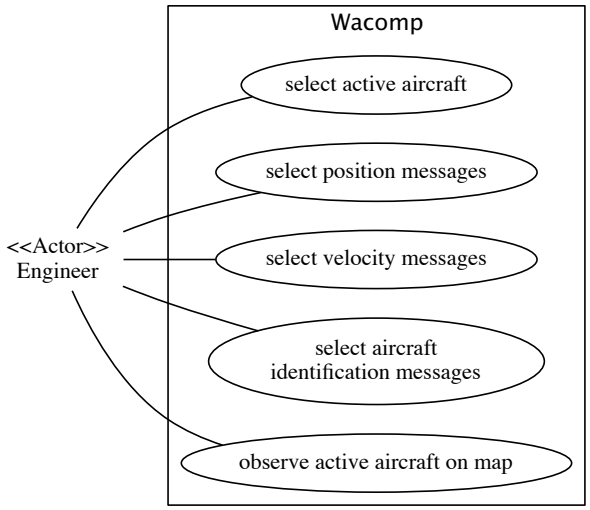
Local Flight Control Engineer, User. Priority: **Key User**.

Chapter 2

Functional Requirements

2.1 Data Model and Data Dictionary

2.1.1 Use Case Diagram



2.2 Wacmop Functional Requirements

2.2.1 Existing Functionality from Acamo

Wacmop.F.10 Select Active Aircraft **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 the aircraft that are currently active.

Scenario

Given the application does not show the active aircraft

When I select to observe the active aircraft

Then the application should show the active aircraft

Active aircraft send any arbitrary new ADS-B message within a time span of 4 minutes.

Scenario

Given an ADS-B message for aircraft ICAO 02A1A2 at time t

When an ADS-B message is received from aircraft 02A1A2, and the current time is less than or equal to $t + 240$ sec

Then aircraft 02A1A2 is an active aircraft, and shall be shown in the list of active aircraft

Scenario

Given an ADS-B message for aircraft ICAO 02A1A2 at time t

When no more ADS-B messages are received for ICAO 02A1A2 after $t + 240$ sec

Then aircraft 02A1A2 is an inactive aircraft, and shall be removed from the list of active aircraft

Feature Each active aircraft shall be shown with the following information:

- ICAO of the aircraft
- Timestamp of the last activity
- Call Sign of the aircraft if available
- Most recent 3-dimensional position with latitude, longitude and altitude
- Most recent velocity with horizontal and vertical speeds, and heading

Feature The application shall show the active aircraft upon application startup.

Scenario

Given the application is off

When I start the application

Then the application should show the active aircraft

Wacmop.F.20 Select Position 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 the decoded incoming position messages.

Scenario

Given the application does not show the position messages

When I select to observe the position messages

Then the application should show the position messages

Feature Each position message shall be shown with the following decoded information:

- ICAO of the aircraft
- CPR format
- CPR encoded latitude
- CPR encoded longitude
- Altitude

Wacmop.F.30 Select Velocity 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 the decoded incoming velocity messages.

Scenario

Given the application does not show the velocity messages

When I select to observe the velocity messages

Then the application should show the velocity messages

Feature Each velocity message shall be shown with the following decoded information:

- ICAO of the aircraft
- Horizontal speed
- Vertical speed
- Heading

Wacmop.F.40 Select Aircraft Identification 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 the decoded incoming aircraft identification messages.

Scenario

Given the application does not show the aircraft identification messages

When I select to observe the aircraft identification messages

Then the application should show the aircraft identification messages

Feature Each aircraft identification message shall be shown with the following decoded information

- ICAO of the aircraft
- Call Sign of the aircraft

2.2.2 New Functionality in Wacmop

Wacmop.F.40 Observe Active Aircraft on Map **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 the active aircraft on a map.

Scenario

Given an aircraft

When the aircraft is in range of the ADS-B base station (it is active)

Then it shall be shown as dot on the map at its correct latlon position

Chapter 3

Non-Functional Requirements

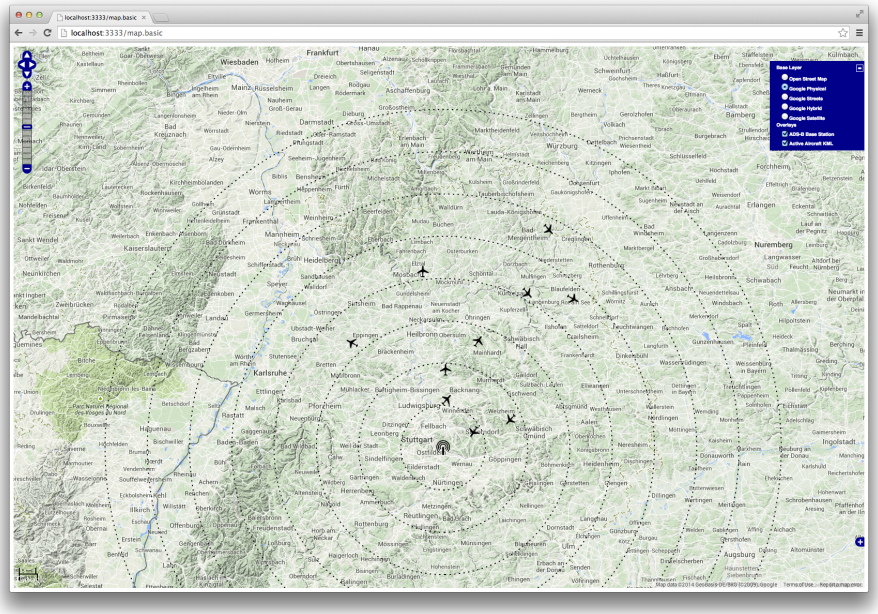
3.1 Look and Feel Requirements

Wacmop.NF.10 Graphical User Interface (GUI) essential

Feature The application user interface shall be realized as graphical user interface with a map to be shown in the web browser.

Feature : Existing Functionality from Acamo The GUI window shall be organized in terms of four tabbed panes, one for each of the previously defined use cases.

Feature : New Functionality in Wacmop The display of the map in the web browser shall be according to the figure below.



3.2 Performance Requirements

Wacmop.NF.20 Timing — essential

Feature The list of active aircraft shall be updated at least once per second.

3.3 Maintainability Requirements

Wacmop.NF.70 Documentation — essential

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

Wacmop.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.

Wacmop.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.

Chapter 4

Implementation Support

4.1 Communication Middleware

The Web server for providing aircraft information in browser-based maps could be realized as part of the application *ACAMO*. However, this would lead to convoluted architectures. Following the Model-View-Controller architectural pattern, the model, i.e. the provision of the aircraft information, and the view, i.e. the map in the browser, should be logically separated by the controller, i.e. the web server. That way, the web server cannot only provide *ACAMO* data, but can also be used to access other sources of information in future applications.

Separating these three concerns requires three different separated processes and some communication means between them. The communication between the controller layer and the presentation layer is standardized through readily available web technologies, e.g. by using *HTTPExchange* in Java (refer to `com.sun.net.httpserver.HttpServer` and `com.sun.net.httpserver.HttpExchange`).

The communication between the model layer and the controller can be realized easily through *Redis*[1], a sophisticated in-memory key-value store, which can be accessed via a Java binding, referred to as *Jedis*[2].

Once a local *redis-server* instance is running, it can be accessed from within Java by using the *Jedis* interface in the following way, where <key> and <value> are arbitrary strings:

```
import redis.clients.jedis.*;

...
```

[1] http://redis.io
[2] http://redis.io/clients

```
Jedis jed = new Jedis ("localhost"); // create new redis client

jed.set ( <key>, <value> );          // store a data set with a key in redis

jed.get ( <key> );                   // retrieve the data for this key
```

4.2 Web-based Geographic Data Processing

4.2.1 Keyhole Markup Language

A standard for processing geographic data is Google's Keyhole Markup Language[3] (KML). Data represented by a KML string (or file) can be projected on top of a map provided via internet, e.g. OpenStreetMap, or Google Maps, Google Earth. The following code represents the KML string for a single active aircraft. The model part of the application is supposed to generate such a string for the active aircraft and store it in *Redis*. The web server can, then, fetch this string from *Redis* and send it to a connected web browser that displays the map plus the KML data.

```
<kml xmlns="http://www.opengis.net/kml/2.2">
<Document>
    <Style id="40612C">
        <IconStyle>
        <scale>0.7</scale>
        <heading>137</heading>
        <Icon>
            <href>http://localhost:3333/icons/plane09.png</href>
        </Icon>
        </IconStyle>
    </Style>
    <Placemark>
        <name>40612C</name>
        <description>
        EZY19PE Lon: 9.645923815275493 Lat: 49.78056335449219 Alt: 11292m Dir: 137deg Vel: 483kn Clm: 0ft/min
        </description>
        <styleUrl>#40612C</styleUrl>
        <Point>
            <coordinates>9.64592382, 49.78056335, 11292</coordinates>
            <altitudeMode>relativeToGround</altitudeMode>
            <extrude>1</extrude>
        </Point>
    </Placemark>
</Document>
</kml>
```

4.2.2 HTML Map

The map is provided as *HTML* file with its own business logic in the form of *Javascript* code. The map functionality is provided through the open source Javascript library *Openlayers*[4] which is related to the *OpenStreetMap*[5] project. The following HTML code illustrates how the libraries are loaded via the Internet.

[3] https://developers.google.com/kml/

[4] http://openlayers.org

[5] www.openstreetmap.org

```
<!-- access Openlayers API (OpenStreetMap) -->
<script type="text/javascript" src="http://openlayers.org/api/OpenLayers.js"></script>
<!-- access Google Maps Layers -->
<script type="text/javascript" src="http://maps.google.com/maps/api/js?v=3&amp;sensor=false"></script>
```

Once the HTML code is opened by a web browser, its contained Javascript code is executed by the web browser. This entails the fetching of the Javascript libraries as well as the actual business logic from the web server that provides the Active Aircraft data. This is shown in the following Javascript code snippet. It defines where in the Internet the KML data can be found with the aircraft data to be displayed on the map. In this case, a local URI. The web server must provide the KML string under this address.

```
// ----- define a url for fetching kml strings
// ----- must be provided by a web server at this uri
// ----- you will probably have to amend this line, depending on your web server URI
kmlUrl = "http://localhost:3333/active.kml";
```

In addition, the map must define a layer for displaying this information, plus determine that the format of the data is KML. This is illustrated in the following code.

```
actAircraftKml = new OpenLayers.Layer.Vector("Active Aircraft KML", {
    strategies: [new OpenLayers.Strategy.Fixed()],
    protocol: new OpenLayers.Protocol.HTTP({
        url: kmlUrl,
        format: new OpenLayers.Format.KML({
            extractStyles: true,
            extractAttributes: true,
            maxDepth: 2
        })
    })
});
```

Finally, it has to be decided how often the map requests new KML data periodically from the web server. The following example code defines the update of the KML layer to take place every 500 ms. In other words, the map requests new KML data from the web server every half a second.

```
// ----- set the kml aircraft layer to be reloaed from http every second
window.setInterval(UpdateKmlLayer, 500, actAircraftKml);
}

function UpdateKmlLayer(layer) {
    // this is the update function called every second by window.setInterval()
    layer.loaded = false;
    layer.setVisibility(true);
    layer.refresh({ force: true, params: { 'key': Math.random()} });
}
```

The only remaining HTML code is used in order to load and display the map.

```
<body onload="adsbMapInit();">
    <div id="mapdiv"></div>
</body>
```

4.3 Web Server

The architecture of a Java-based Web server is typically organized through so-called servlets[6]. Every servlet is associated with a specific function or resource that the web server provides. Every resource in the Internet is defined uniquely through its Uniform Resource Identifier[7] (URI).

When a client (web browser) requests a resource from a server, an instance of a servlet is created (Java task) that handles this request. That way, a web server can handle many new incoming requests in parallel without having to wait until any older request has been completed. The performance of a web server is, therefore, to a large extent dependent on the number of tasks it may create, and hence on the size of the memory available for the web server.

Because the standard web kit for building web servers in the JDK is awkward to use, it is advisable to work with a more sophisticated web library, such as the one provided by *com.sun.net.httpserver.HttpServer*. This also comes with an HTTP exchange class, i.e. *com.sun.net.httpserver.HttpExchange*, which holds the data that is exchanged between the client (browser) and the server, and an HTTP handler interface, i.e. *com.sun.net.httpserver.HttpHandler* that is used to implement the Servlet funcationality, or the business logic of the web server.

A web server for providing aircraft data is rather straight-forward. It requires a servlet for providing the map, and a servlet for providing the KML string with the aircraft data. In addition it may have to provide some additional bits for the web page to display properly, such as dynamicaly determined icons used in KML, for example. The *main* function of the web server binds it all together.

```
public static void main(String...args) throws Exception {
    HttpServer server
            = HttpServer.create(new InetSocketAddress(3333), 0);
    server.createContext( "/resource", new ResourceHandler () );
    server.setExecutor(null); // create a default executor
    server.start();
}

...

static class ResourceHandler implements HttpHandler {
    public void handle(HttpExchange t) throws IOException {
        String response =  .... // create a string response
        t.sendResponseHeaders(200, response.length());
        OutputStream os = t.getResponseBody();
        os.write(response.getBytes());
        os.close();
    }
}
```

[6] https://en.wikipedia.org/wiki/Java_Servlet
[7] https://en.wikipedia.org/wiki/Uniform_resource_identifier

4.4 Wacmop – Component Architecture