D5.3 HMIs Implementation Report

Deliverable ID:	**D5.3**
Project acronym:	**ATMACA**
Grant:	**101167070**
Call:	**HORIZON-SESAR-2023-DES-ER-02**
Topic:	**HORIZON-SESAR-2023-DES-ER-02-WA2-2**
Consortium coordinator:	**Eskişehir Teknik Üniversitesi**
Edition date:	**30 March 2026**
Edition:	**01.00**
Status:	**Official**
Classification:	**PU**

Abstract

This deliverable describes the implementation of the ATMACA Human-Machine Interfaces (HMIs), translating the previously defined and validated designs into a functional technological proof of concept. It outlines the refinement of the HMI designs based on usability feedback, the front-end implementation approach, and the integration with the simulation backend through a WebSocket-based communication architecture. The document presents the main technological choices and demonstrates the feasibility of real-time digital communication between air and ground actors. The current limitations of the implementation are also discussed, in line with the exploratory nature and early maturity of the project.

Authoring & approval

Author(s) of the document

Organisation name	Date
Tommaso Vendruscolo / DBL	20/03/2026

Reviewed by

Organisation name	Date
Fulya Aybek Cetek / ESTU	27/03/2026
Michael Regnault / ENAC	24/03/2026

Approved for submission to the SESAR 3 JU by

Organisation name	Date
Fulya Aybek Çetek / ESTU	28/03/2026
Stefano Bonelli and Tommaso Vendruscolo / DBL	28/03/2026
Raouf Hamzaoui / DMU	28/03/2026
Georges Mykoniatis / ENAC	28/03/2026
Matthew Cornwall / SAERCO	28/03/2026
Sergun Özmen / THY	28/03/2026
Rosa María Arnaldo / UPM	28/03/2026

Rejected by[1]

Organisation name	Date

Document history

Edition	Date	Status	Company Author	Justification

[1] Representatives of the beneficiaries involved in the project.

The beneficiaries/consortium confirm(s) the correct application of the Grant Agreement, which includes data protection provisions, and compliance with GDPR or the applicable legal framework with an equivalent level of protection, in the frame of the Action. In particular, beneficiaries/consortium confirm(s) to be up to date with their consent management system.

Copyright statement

Disclaimer

ATMACA

AIR TRAFFIC MANAGEMENT AND COMMUNICATION OVER ATN/IPS

ATMACA

This document is part of a project that has received funding from the SESAR 3 Joint Undertaking under grant agreement No 101167070 under European Union's Horizon Europe research and innovation programme.

The UK participant (De Montfort University) is supported by UKRI grant number 10121610.

Table of Contents

List of figures

List of tables

1 Introduction

1.1 Purpose and Scope of the Deliverable

The purpose of this deliverable is to describe the implementation of the ATMACA Human-Machine Interfaces (HMIs), building upon the design activities and usability validation presented in previous project outputs.

The document focuses on:

- the refinement of HMI designs following usability testing;
- the implementation of the front-end interfaces;
- the integration of the HMIs with the backend simulation environment;
- the development of a technological proof of concept.

The scope of the deliverable is limited to the implementation of core functionalities supporting digital communication and coordination between air and ground actors. The objective is not to provide a production-ready system, but to demonstrate the feasibility of the proposed HMI solutions and their underlying architecture.

1.2 Context within the Project

This deliverable is part of the ATMACA project activities related to the development and validation of digital communication concepts in Air Traffic Management.

It builds upon:

- the Functional Requirements Document (FRD) [1];
- the HMI requirements defined in Deliverable D5.1 [2];
- the usability validation conducted with end users, described in D5.2 [3];
- the work carried out by partners on the back-end;
- the definition of the Concept of Operations (CONOPS).

In particular, the deliverable follows the HMI design report [3], where the interaction concepts and interface structures were defined and evaluated through low-fidelity simulations. The results of those activities informed the refinement of the designs and provided the basis for their implementation.

The work presented in this document contributes to the progressive maturation of the ATMACA solution, moving from conceptual design towards a functional prototype. As such, it supports the early validation of the technical feasibility of the proposed architecture and interaction mechanisms, in alignment with the exploratory nature of the project.

Co-funded by the European Union

2 HMI Refinement after Usability Testing

2.1 Overview of Collected Feedback

The usability testing activities conducted in the previous phase provided a structured assessment of the ATMACA HMI designs, combining task-based evaluation, questionnaires, and qualitative feedback from representative end users (pilots and ATCOs).

Overall, the results confirmed the validity of the proposed interaction concepts and the alignment of the HMIs with user expectations. No critical usability issues were identified, and the core workflows (message exchange, flight management, and handoff) were consistently understood and successfully executed by users.

However, the feedback also highlighted a set of recurring themes requiring refinement. These were primarily related to:

- visual hierarchy and clarity of information;
- visibility of system status and action outcomes;
- support for workload management in operational contexts.

Given the exploratory nature of the project and its early technology readiness level, the feedback was interpreted with a focus on identifying targeted improvements with high usability impact, rather than introducing structural changes to the HMI design.

The refinement process therefore prioritised:

- issues mentioned consistently across users;
- elements directly affecting task execution;
- improvements that could be implemented without altering the overall interaction model.

2.2 Design Adjustments for Implementation

Based on the analysis of usability feedback, a limited set of design adjustments was introduced prior to implementation. These refinements aimed at improving clarity, feedback, and interaction efficiency, while preserving the overall structure and logic of the HMIs.

The main adjustments are described below.

Improved visibility of the handoff process

The handoff operation was identified as a critical interaction requiring clearer feedback. While users were able to complete the task, feedback indicated the need for a more explicit representation of its status and outcome.

To address this, the handoff process was refined by:

- improving the visual indication of the selected target unit;
- enhancing the visibility (by color coding) and the process of the handoff action;
- ensuring that the outcome of the operation is clearly communicated to the user.

This improvement directly responds to feedback related to the need for better awareness of system state during coordination activities.

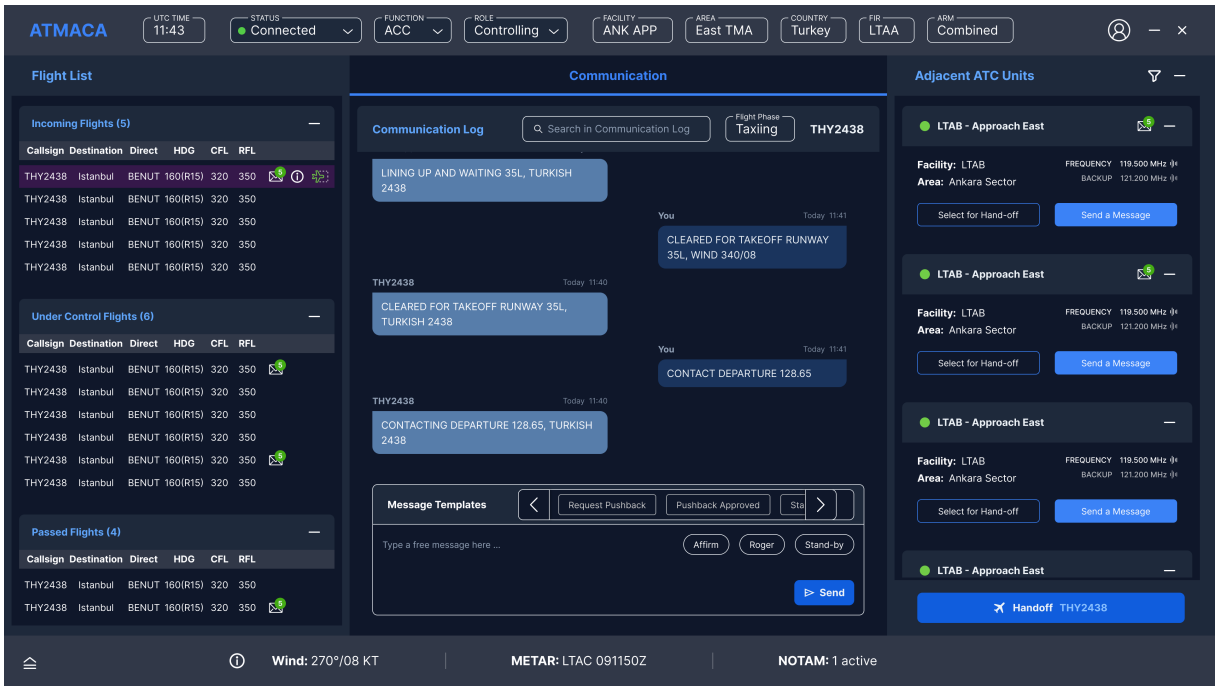

Figure 1. When the previous ATC unit initiates the handoff, the background of the aircraft's flight strip turns purple and begins flashing - indicating to the receiving ATC unit that it must assume control of the flight - along with the takeover icon.

Introduction of explicit system feedback mechanisms

A consistent theme across both ATCO and pilot feedback was the need for clearer confirmation of user actions and system responses.

To address this, snackbar-based feedback mechanisms were introduced across all HMIs. These provide immediate, transient notifications for key actions, such as:

- message sent;
- handoff completed;
- system status updates.

This enhancement improves the visibility of action outcomes and reduces uncertainty during interaction, particularly in time-sensitive situations.

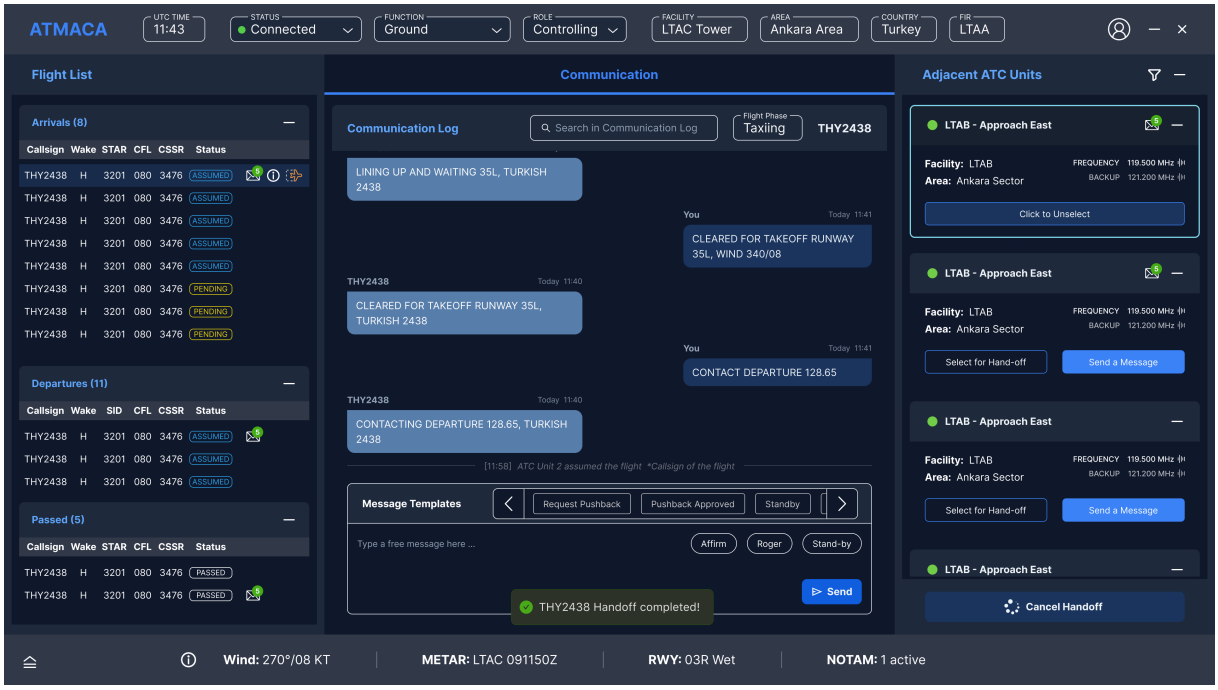

Figure 2. A snackbar appears to notify that the handoff is complete and the next ATC unit has successfully assumed control of the flight.

Enhanced colour coding and visual hierarchy

Feedback highlighted the importance of clearer visual differentiation between elements, particularly for flight status and operational priorities.

To improve readability and support faster decision-making, the following refinements were introduced:

- improved colour coding for flight states (e.g. pending, under control, handed off);
- enhanced visual distinction of handoff-related elements;
- adjustments to contrast and emphasis to support visual scanning.

These changes aim to reduce cognitive load and improve the user's ability to quickly interpret the interface, especially under higher workload conditions.

Alignment with usability feedback

The implemented adjustments directly address the main usability findings identified during testing, in particular:

- the need for improved visual prioritisation;

- the importance of clear and immediate system feedback;
- the optimisation of interaction clarity in operational contexts.

At the same time, the refinements were deliberately limited in scope to maintain consistency with the validated design and to ensure continuity between design and implementation phases.

2.3 Final HMI Specifications for Implementation

Following the refinement phase, the HMI designs reached a stable configuration suitable for implementation.

The final specifications:

- preserve the interaction logic validated during usability testing;
- incorporate targeted improvements addressing user feedback;
- ensure consistency across all HMIs in terms of interaction patterns and visual language.

These refined designs constitute the baseline for the implementation activities described in the following chapters.

3 HMI Implementation

3.1 Implementation Approach

The implementation phase focused on translating the validated HMI designs into functional front-end applications, integrated with the simulation backend and capable of supporting real-time interaction.

Three types of interfaces were implemented:

- **Tower HMI**, supporting Tower, Ground, and Delivery controllers;
- **En-Route HMI**, supporting En-Route and Approach controllers;
- **Flight Deck HMI**, supporting pilots interacting with ATC units.

All HMIs are developed within a single codebase, ensuring consistency across interfaces and enabling reuse of common components, interaction patterns, and data handling logic. Interface-specific behaviour is configured based on the role and scenario context associated with each instance.

Each HMI instance is initialised using scenario-specific parameters and connects to a dedicated WebSocket session of the banckend node through a unique URL and port configuration. This allows multiple instances of the HMIs to run in parallel, each representing a specific actor within the simulation.

The implementation is designed as a demonstration-oriented prototype, focusing on validating core interaction mechanisms and integration with the backend, rather than delivering a production-ready system.

3.2 Front-End Technology Stack

The main technologies used are:

- **ReactJS**, for component-based user interface development;
- **TypeScript**, providing type safety and improved maintainability;
- **Vite**, enabling fast development and efficient build processes;
- **Chakra UI**, ensuring consistent and responsive interface components;
- **Zustand**, used for lightweight state management.

Within this architecture, Zustand is primarily used for application context management, enabling the HMI to maintain and update the current system state received from the backend, including flight data, messages, and interaction status.

This stack supports the development of responsive Single-Page Applications (SPAs), capable of handling dynamic updates and real-time interactions without requiring page reloads.

3.3 Interface Structure and Data Handling

The implemented HMIs follow the structural organisation defined during the design phase, including:

- contextual information areas (e.g. header/context bar);
- flight lists and operational data views;
- messaging interfaces for communication between actors;
- interaction elements supporting handoff and message exchange.

The interfaces are designed to react dynamically to changes in system state. Data received from the backend is processed and directly mapped to the application state managed within the front-end.

Incoming events are handled through a **state-driven approach**, where:

- backend messages are received via WebSocket;
- events are parsed and interpreted by the front-end;
- the application state is updated accordingly;
- UI components automatically re-render based on the updated state.

This approach ensures a clear separation between data acquisition (WebSocket communication) and data presentation (UI components), while maintaining a responsive and consistent user experience.

3.4 Web-socket Connection Handling

Each HMI instance establishes a **single persistent WebSocket connection** with the backend system. This connection is used for all communication between the interface and the simulation engine.

The use of a single connection per instance ensures:

- a simplified communication model;
- consistent synchronisation between frontend and backend systems;
- efficient handling of real-time updates.

The connection is configured at startup based on the specific role and simulation scenario, allowing each HMI instance to interact with the appropriate backend session.

3.5 Multi-instance Execution and Simulation Support

The implementation supports the running of **multiple HMI instances in parallel**, each representing a different actor within the simulation environment.

This is achieved through:

- configuration of different connection endpoints (e.g. ports or URLs);
- association of each instance with a specific role (e.g. Tower ATCO, En-Route ATCO, Pilot);
- scenario-based initialisation of data and context.

This capability is essential for simulating realistic operational scenarios, where multiple actors interact simultaneously through their respective HMIs.

Although simplified compared to real operational environments, this setup enables the demonstration of:

- coordinated interactions between actors;
- real-time communication flows;
- synchronisation of system state across multiple interfaces.

4 Integration with the Back-End System

4.1 Integration Approach

The HMI front-end is integrated with a backend system implemented in C++, responsible for simulating CPDLC, DLCM, and related protocol logic. The integration follows a decoupled architecture, where the front-end and backend are developed and executed as independent components.

The backend acts as the core simulation engine, managing:

- communication protocols and message handling;
- system state and flight data;
- simulation scenarios and predefined configurations.

The front-end HMIs act as lightweight clients, focusing exclusively on:

- visualisation of system state;
- user interaction handling;
- transmission of user commands to the backend and processing messages from the backend as defined in API contract (Appendix A).

This separation enables independent development, testing, and maintenance of the two layers, while ensuring flexibility and scalability of the overall system.

4.2 System Architecture Overview

The overall architecture is based on a real-time, event-driven model, connecting the backend simulation engine with multiple HMI instances through a communication layer .

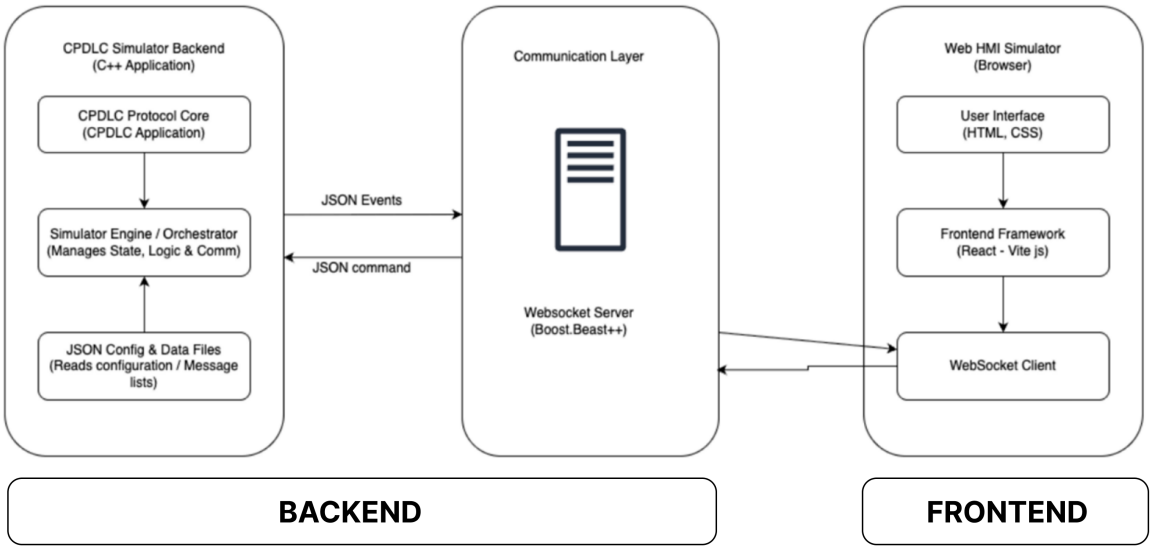

Figure 3. System Architecture Overview

The architecture consists of three main components:

- **Backend (C++ Simulation Engine)**
  A standalone application responsible for executing the simulation logic and managing protocol behaviour.
- **Communication Layer (WebSocket API)**
  A middleware layer enabling real-time, bidirectional communication between backend and front-end.
- **Frontend (Web HMI Applications)**
  Browser-based interfaces implemented as Single-Page Applications, providing the user interaction layer.

This architecture ensures that multiple HMI instances (Tower, En-Route, Flight Deck) can operate simultaneously, each connected to the backend through a dedicated session.

Further details on the structure of the exchanged messages and the API contract between front-end and back-end are provided in **Appendix A – API Contract**.

4.3 Websocket Communication Model

The HMI and backend communicate via a persistent, bidirectional WebSocket-based API, serving as the **primary** channel for real-time data exchange and state synchronization.

The WebSocket connection enables:

- persistent communication sessions;
- low-latency data exchange;

- asynchronous and event-driven interactions.

All data exchanged between front-end and backend is structured as JSON payloads.

4.4 Command and Event Flow

The interaction between HMI and backend follows a continuous command–event cycle:

- **Command (HMI → Backend)**
  User actions (e.g. sending a message, initiating a handoff) are translated into JSON commands and transmitted via WebSocket.
- **Processing (Backend)**
  The backend simulation engine interprets the command and executes the corresponding logic through the protocol core.
- **Event (Backend → HMI)**
  Once processed, the backend generates a JSON event representing the updated system state.
- **UI Update (Frontend)**
  The HMI receives the event and updates the interface accordingly in real time.

This model ensures that the interface remains synchronised with the simulation state at all times.

4.5 Back-End internal structure (high-level)

The backend is structured around a modular architecture, including:

- a **simulation engine**, acting as orchestrator of the system;
- a **protocol core**, responsible for CPDLC message handling and state management;
- a **WebSocket server**, managing connections and routing messages;
- configuration and data files defining simulation scenarios.

The backend is also based on the **Data Link Context Management (DLCM),** enabling the maintenance of communication context across different actors and system interactions.

The backend communicates internally using callback or observer mechanisms, allowing changes in the protocol core to be propagated to the HMI through events.

4.6 Scenario configuration and data management

Before execution, the system is pre-loaded with scenario data required to initialise the simulation environment.

This includes:

- airspace configuration (sectors, roles, facilities);
- flight data (identification, initial state, trajectories);
- predefined simulation events.

The HMI retrieves and displays this data through backend communication, ensuring that all actors operate within a consistent simulation context.

5 Technological Proof of Concept

5.1 Implemented Features

The developed solution provides a functional proof of concept of the ATMACA HMI system, demonstrating the feasibility of the core interaction mechanisms defined in the Operational Services and Environment Description (OSED) [5].

The implementation includes the following key features:

- **Message exchange between ATCOs and Flight Deck users**
  The system supports real-time digital communication between pilots and ATC units, enabling the exchange of CPDLC-like messages through the HMI interfaces. Users can initiate, send, and receive messages, with updates propagated in real time across the connected interfaces.

- **Flight handoff process**
  The solution implements the transfer of flight responsibility between ATC units through a digital handoff mechanism. This includes the initiation of the handoff action and the corresponding update of flight status across the relevant HMIs.

These features represent the core building blocks of the ATMACA concept and validate the ability of the system to support structured, digital interactions between air and ground actors.

5.2 Demonstration Capabilities

The implemented solution is intended as a **local technological proof of concept** and is not deployed as an online or production-ready system.

Within this scope, the system enables the demonstration of:

- **Real-time interaction between multiple actors**
  Multiple HMI instances (Tower, En-Route, Flight Deck) can be executed in parallel, each connected to the backend simulation and interacting within the same scenario.

- **End-to-end communication flows**
  The full interaction cycle-from user action to backend processing and UI update—is demonstrated through WebSocket-based communication.

- **Consistency of system state across interfaces**
  Updates generated by one actor are propagated to other relevant HMIs, ensuring a shared and synchronised view of the operational context.

- **Feasibility of the proposed architecture**
  The integration between a web-based HMI and a C++ simulation backend demonstrates the viability of the selected architectural approach for real-time ATM communication scenarios.

Although limited in scope, the proof of concept provides a concrete validation of the technical approach and supports further development and refinement of the system.

5.3 Current Limitations

As an early-stage implementation developed for demonstration purposes, the proof of concept presents several limitations.

- **Limited communication scope**
  Message exchange is currently implemented only between Flight Deck users and ATC units. Communication between ATC units is not supported in the current implementation.
- **Limited robustness of session handling**
  The system does not include advanced session management mechanisms. As a result, if an HMI instance is accidentally closed, there is a possibility of losing notifications or updates from the backend.
- **Prototype-level implementation**
  The solution is not designed for deployment in operational environments and does not include features such as fault tolerance, or advanced error handling.

These limitations are consistent with the exploratory nature of the project and its current technology readiness level (TRL 1 → TRL2). They do not affect the validity of the proof of concept, whose objective is to demonstrate the feasibility of core interaction mechanisms and system integration.

6 References

[1] ATMACA-SESAR, Deliverable D2.2, Functional Requirements Document (FRD), 2025

[2] ATMACA-SESAR, Deliverable D5.1, Determination of HMI Requirement, 2025

[3] ATMACA-SESAR, Deliverable D5.2, HMI Design and Testing Report, 2026

[4] Maguire, M. (2001). Methods to support human-centred design. *International Journal of Human-Computer Studies*, *55*(4), 587–634. https://doi.org/10.1006/ijhc.2001.0503

[5] ATMACA-SESAR, Deliverable D2.3, Operational Services and Environment Description, 2025

7 List of acronyms

Acronym	Description
API	Application Programming Interface
ATC	Air Traffic Control
ATCO	Air Traffic Controller
ATMACA	Air Traffic Management and Communication over ATN/IPS
FRD	Functional Requirements Document
HMI	Human Machine Interface
JSON	JavaScript Object Notation
PoC	Proof of Concept
SESAR	Single European Sky ATM Research
SPA	Single-Page Application
TRL	Technology Readiness Level

Table 1: list of acronyms

8 Appendix A

API Contract Detailed

- **HMI-to-Server (Client -> Server):**

1. {

   "source": "HMI", type: “HMI_LOGON”, “payload”: {"interface": "TOWER" | "EN_ROUTE" | "FLIGHT_DECK", "userID": <sectorHost | callsign: string>}

   } - triggered when HMI mounts.

2. {

   "source": "HMI", "type": "HMI_SEND_CPDLC_MSG", "payload": {“sender”: <sectorHost | callsign: string>, “recipient”: <sectorHost | callsign: string>, "callsign": <callsign: string>, "message": {“service”: <service: string>, “messageId”: <id: string>, “parameters”: [<param1: any>, <param2: any>, ..], “additionalInfo”: {“wilcoPayload”: <message: string>} } }

   } - triggered by send message button. Comment: additionalInfo is used for WILCO messages.

3. {

   "source": "HMI", "type": "HMI_REQUEST_HANDOVER", "payload": {“callsign”: <callsign: string>, "targetSector": <sectorHost: string>, "originSector": <sectorHost: string>}

   } - triggered by handoff button.

4. {

   "source": "HMI", "type": "HMI_REQUEST_ASSUME_FLIGHT", "payload": {“callsign”: <callsign: string>, "originSector": <sectorHost: string>, "targetSector": <sectorHost: string>}

   } - triggered by Tower or EnRoute controller clicks Assume button to accept flight handoff.

5. {

   "source": "HMI", "type": "HMI_REQUEST_MESSAGE_LOG", "payload": { “callsign”: <callsign: string> }

   }

- **Server-to-HMI (Server -> Client):**

1. {

   "source": "DLCM", "type": "CONNECTION_STATUS_UPDATE", "payload": { "status": "CONNECTED" | "DISCONNECTED", “role”:<“controlling”| “mirroring”| “monitoring”..: string>, "additionalInfo": {...} }

} - Comments: For simulation purposes, role is always “controlling”

2. {

"source": "CPDLC", "type": "MESSAGE_LOG_UPDATE", "payload": { "callsign": <callsign: string>, "messages": [{“sender”:<sectorName | callsign: string>, “message”: <message: string>, “timeStamp”: <timestamp: dateTime>}, {“sender”:<sectorName | callsign: string>, “message”: <message: string>, “timeStamp”: <timestamp: dateTime>}, {...}] } - Comments: message should show full message text. Messages should be in chronological order.

}

3. {

"source": "CPDLC", "type": "NODE_DATA_UPDATE", "payload": { “interface: "FLIGHT_DECK", "callsign":<callsign: string>, "flightLevel": <level: string | number>, "flightPhase": <phase: string>, “currentATCUnit”: <sectorHost>, “currentFrequency”:<freq>, “currentBackupFrequency”: <freq>}

} - Comments: essentially intended to populate dynamic fields in flight data in FlightDeckHMI.

4. {

"source": "SYSTEM", "type": <"ERROR“ | “NOTIFICATION”>, "payload": { "message":<messageText: string> }

} - Comments: if there is a need to handle particular errors in HMI then different error message types should be developed to properly process them - Guillermo?

5. {

"source": "CPDLC", "type": "HANDOVER_NOTIFICATION", "payload": {callsign: "callsign":<callsign: string>, “originSector”: <sectorHost: string>, "targetSector": <sectorHost: string>}

}

6. {

"source": "CPDLC", "type": "FLIGHT_LIST_UPDATE", "payload": [{ "callsign": <callsign: string>, “departureAirport”: <departureAirport: string>, “destinationAirport”: <destinationAirport: string>, “flightPhase”: <phase: string>, “flightLevel”: <level: number | string>, “runWay”: <runWay: string | null>, “heading”: <heading: string | null>, “SID:<SID: string>, … , "towerHMISection": "departures"| “arrivals | null”, “assumed_tag”: <”assumed”| “pending”> }, {...}]

}

7. {

"source": "CPDLC", "type": "HANDOVER_TERMINATION", "payload": {callsign:

"callsign":<callsign: string>, “originSector”: <sectorHost: string>, "targetSector": <sectorHost: string>, "targetFrequency": <frequency: number>, "targetBackupFrequency": <frequency: number>}

}

8. {

"source": "CPDLC", "type": "METAR_UPDATE", "payload": {“windSpeed”: <number>}

} -Constant value can be 280º, 5KT.