ATMACA DLIC Application Design Document

Deliverable ID:	**D4.3**
Project Acronym:	**ATMACA**
Grant:	**101167070**
Call:	**HORIZON-SESAR-2023-DES-ER-02**
Topic:	**HORIZON-SESAR-2023-DES-ER2-WA2-2**
Consortium Coordinator:	**Eskişehir Teknik Üniversitesi**
Edition date:	**31 October 2025**
Edition:	**01.04**
Status:	**Official**
Classification:	**PU**

Abstract

This document defines the Data Link Context Management (DLCM) application of the ATMACA protocol. DLCM extends the legacy ATN/OSI Data Link Initiation Capability (DLIC) by introducing an enhanced framework that includes context, session, connection, and mobility management modules. It provides functions and message sets for logon and registration, identity and session binding, and role-based access (controlling, mirroring, monitoring). It also supports seamless handover between Air Traffic Services Units (ATSUs) and multilink operation across heterogeneous communication paths. Architecturally, DLCM serves as an intermediary between client applications and the transport layer, maintaining communication session continuity as roles or network paths change. This document specifies the data structures, workflows, and technical specifications for the context, session, connection, and mobility management modules.

EUROPEAN PARTNERSHIP

Co-funded by the European Union

Authoring & Approval

Author(s) of the document

Organisation name	Date
Sergun Özmen / THY	31/10/2025
Hassna Louadah / DMU	31/10/2025
Raouf Hamzaoui / DMU	31/10/2025
Feng Chen / DMU	31/10/2025

Reviewed by

Organisation name	Date
Guillermo Martin Vicente / UPM	20/10/2025
Michael Regnault / ENAC	22/10/2025

Approved for submission to the SESAR 3 JU by[1]

Organisation name	Date
Fulya Aybek Çetek / ESTU	31/10/2025
Stefano Bonelli and Tommaso Vendruscolo / DBL	31/10/2025
Raouf Hamzaoui / DMU	31/10/2025
Georges Mykoniatis / ENAC	31/10/2025
Ángel Ernesto Carmona Fernández / SAERCO	31/10/2025
Sergun Özmen / THY	31/10/2025
Rosa María Arnaldo / UPM	31/10/2025

Rejected by[2]

Organisation Name	Date

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

Document History

Edition	Date	Status	Company Author	Justification
01.01	30/09/2025	Draft	THY/DMU	First draft
01.02	14/10/2025	Draft	THY/DMU	Submitted for internal review
01.03	28/10/2025	Draft	THY/DMU	Revised after internal review
01.04	31/10/2025	Final	ESTU	Approved by the Consortium.

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.

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

Co-funded by the European Union

List of Figures

List of Tables

Acronym	Name
ACC	Area Control Centre
AeroMACS	Aeronautical Mobile Airport Communications System
ATC	Air Traffic Control
ATCo	Air Traffic Controller
ATM	Air Traffic Management
ATMACA	Air Traffic Management and Communication over ATN/IPS
ATN	Aeronautical Telecommunication Network
ATSU	Air Traffic Services Unit
CM	Context Management
CMA	Context Management Application
CPDLC	Controller-Pilot Data Link Communications
DAC	Dynamic Airspace Configuration
DFIS	Digital Flight Information Service
DLCM	Data Link Context Management
DLIC	Data Link Initiation Capability
EFB	Electronic Flight Bag
EUROCAE	European Organisation for Civil Aviation Equipment
EUROCONTROL	European Organisation for the Safety of Air Navigation
FIR	Flight Information Region
LISP	Locator/ID Separation Protocol
GRO	Green Route Operations
HMI	Human Machine Interface
ICAO	International Civil Aviation Organisation
IETF	Internet Engineering Task Force
IP	Internet Protocol
IPS	Internet Protocol Suite
IPv6	Internet Protocol version 6
LDACS	L-band Digital Aeronautical Communications System
LISP	Locator/ID Separation Protocol
OSI	Open Systems Interconnection
SATCOM	Satellite Communications
SCTP	Stream Control Transmission Protocol
SESAR	Single European Sky ATM Research Programme

SWIM	System-Wide Information Management
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
VDLm2	VHF Data Link Mode 2

1 INTRODUCTION

1.1 Background

The Aeronautical Telecommunication Network (ATN), originally based on the OSI protocol stack, has used the Data Link Initiation Capability (DLIC) application to initiate air-ground data link sessions between aircraft and Air Traffic Services Units (ATSUs). As civil aviation transitions to the ATN/Internet Protocol Suite (IPS), ICAO Doc 9896 defines the use of IP-based standards while maintaining interoperability for ATN/IPS applications [1]. Within this evolution, the Context Management (CM) function has become a key element, providing logon, flight context association, and session transfer between ATSUs. EUROCONTROL SPEC-192 further specifies CM (Logon Service/DLIC) as a common, reusable service supporting European Air Traffic Management (ATM) data link operations [2].

Today's operational environment is increasingly mobile and heterogeneous. Aircraft routinely traverse many Flight Information Regions (FIRs) and service domains, transitioning between VDLm2, AeroMACS, and SATCOM links [3]. Consequently, communication sessions, identities, and services availability must be maintained seamlessly across handovers.

Legacy ATN/OSI solutions centred on DLIC provide only initial connectivity and lack robust mechanisms for session continuity during ATSU transitions or role-aware, multi-endpoint operation (e.g., controlling, mirroring, and monitoring) [4].

More recent technologies also face practical constraints. L-band Digital Aeronautical Communications System (LDACS), though standardised by ICAO and EUROCAE, relies on IPv6, mobility, and multihoming capabilities that are not yet widely supported in avionics or Air Traffic Services (ATS) systems. As a result, deployment remains limited, constraining near-term interoperability in mixed environments [3]. Similarly, the Internet Engineering Task Force (IETF) network mobility framework [5] and the Locator/ID Separation Protocol (LISP) architecture [6] offer scalable mechanisms for network-level mobility management. However, these solutions alone do not ensure end-to-end session continuity across heterogeneous links and ATSU boundaries without broader system-level adoption and integration.

Against this backdrop, the ATMACA project proposes a comprehensive Data Link Context Management (DLCM) application to integrate and extend existing context management capabilities with advanced mobility services. By combining logon and registration, identity and session binding, and multi-dimensional mobility management (covering user, session, service, and terminal aspects), DLCM is proposed as a core enabler within the emerging ATN/IPS environment. It maintains interoperability with legacy ATN/OSI systems while ensuring session continuity across network transitions, supporting multilink operations and enabling persistent service delivery. In doing so, DLCM aligns with the objectives of ICAO, EUROCAE, and the Single European Sky ATM Research (SESAR) programme for a harmonised, digital, and service-oriented aeronautical communication infrastructure.

1.2 Purpose

DLCM is a foundational application that builds on the ATMACA communication protocol described in [7][8]. This report describes DLCM's role, the services it offers, and its interfaces. In addition, it specifies the message sets, data structures and workflows that provide context, session, connection, and mobility management. This ensures that applications using it, such as Controller-Pilot Data Link Communications (CPDLC), Digital Flight Information Service (DFIS) and Green Route Operations (GRO), remain operational during handovers and link transitions.

1.3 Methodology

Like the ATMACA protocol, DLCM was developed using a SESAR-aligned, iterative, and requirements-driven methodology with an explicit focus on mobility and end-to-end session continuity [9], [10].

Handover and connectivity requirements, which include link switching, network reselection, ATSU or FIR handovers, and failure recovery, are addressed through four management modules: context, session, connection, and mobility. A standard service interface and transport/session mechanisms provide continuous state monitoring, seamless reconnection, and verified end-to-end context handover and role failover across heterogeneous links throughout the system lifecycle.

1.4 Structure of the Document

The remainder of this document is organised as follows. Section 2 defines the core service capabilities of DLCM, describing the functions it provides, including context, session, connection, mobility, presence, and service delivery management. The section also outlines the data structures, message sets, and processes that enable reliable, role-based, and fault-tolerant operation within the ATMACA framework. Section **3** explains how these capabilities are implemented within the software architecture, detailing the internal structure of the DLCM layer, its modular components, and the main workflows and message exchanges that support context, session, connection, and mobility management. Section 4 presents the conclusions. Appendix A contains the public DLCM service interface catalogue to support application integration.

2 DATA LINK CONTEXT MANAGEMENT

DLCM is a combined DLIC and comprehensive Context Management Application (CMA). It serves as a runtime middleware environment connecting aeronautical applications such as CPDLC, DFIS, and GRO to the supporting communication and management components of the ATMACA system. DLCM enables context-aware, role-driven, and mobility-resilient operation across both stationary and mobile nodes. It provides an integrated set of management functions essential to air traffic communications, including

- Connection Management: Establishes and maintains transport layer connections, manages transitions between ATSUs, and enables seamless handovers across network domains.
- Session Management: Supports the dynamic creation, maintenance, and termination of application-level sessions, ensuring communication continuity throughout the operational lifecycle.
- Context Management: Maintains logical associations between clients and their operational metadata, including IP addresses, ATSU roles, session bindings, and user identities. It also supports context updates, transfers, and disassociations as required during operational transitions.
- Mobility Management: Enables transparent mobility across FIR boundaries through automated reassignment of agents, ATSUs, and transport paths. It supports user, session, terminal, and service mobility within the ATMACA mobility framework.
- Presence Management: Tracks and disseminates the online/offline status and role-specific availability of all participants, including Air Traffic Control (ATC) units and aircraft, to support real-time awareness and coordinated communication.
- Service Delivery Management: Orchestrates the provisioning, invocation, and lifecycle control of aeronautical services based on predefined policies, priorities, and operational triggers, ensuring reliable and context-aware service continuity across network domains.

The DLCM architecture consists of four software modules: context, session, connection, and mobility management (**Figure 2.1**). Note that the presence and service delivery functions are implemented across these four modules.

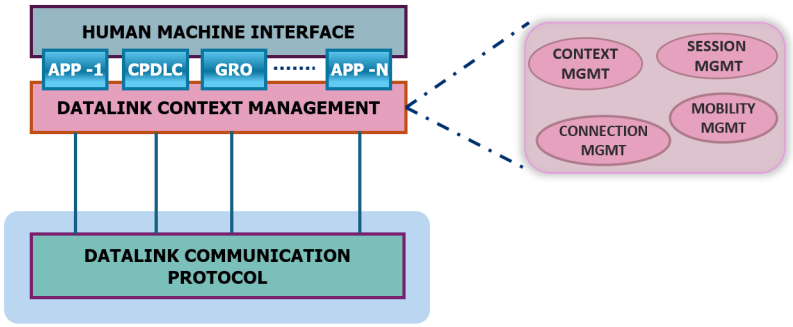

Figure 2.1: ATMACA Solution Conceptual Architecture.

The context manager handles user mobility through context association and role tracking; the session manager ensures session continuity via dynamic session establishment and transfer; and the connection manager manages terminal and service mobility, including handovers across ATSUs. Service mobility may also be supported by the session manager depending on the service design (transaction-based or session-based).

These modules can be deployed individually or in combination. When the session manager and connection manager are co-located, the platform functions as an ATC Agent; a platform hosting only the context manager operates as a CM Agent; and when all three modules are integrated, the platform is referred to as a CM/ATC Agent. This modular design supports both centralised and distributed deployments, enhancing scalability and fault tolerance.

DLCM manages application-to-network interactions in a modular, role-, and context-aware manner. It operates as middleware between client-facing aeronautical applications and the lower communication layers (Section **3**). DLCM does not host or implement the business logic of services like CPDLC or DFIS. Instead, it provides standardised functions that these applications use to establish, maintain, and recover their operational state.

Architecturally, DLCM interacts with

- Client applications, on client nodes or application servers, via a standardised service interface that abstracts low-level session and transport logic.
- ATC and CM Agents to coordinate session routing, role tracking, and context handovers.
- Transport and session layers to monitor network state, manage reconnections, and maintain service continuity.

Its primary scope includes:

- Initial contact and identification: Managing the DLIC process, which creates mutual awareness between the aircraft and the ground system through the exchange of identifiers such as Flight ID and ATSU information before session setup. DLCM extends this process by coordinating agent-to-agent transitions between regions, triggering proactive re-logon when crossing FIR or sector boundaries, and maintaining updated bindings so that sessions and presence continue seamlessly.
- Context association management**:** Linking terminal context (e.g., IP address, ATSU assignment, session metadata) with the appropriate ATSU or ATC agent to ensure accurate routing and coordination.
- Session preparation and control: Aligning session parameters (supported services, timeouts, encryption) and dynamically managing session activation, continuity, and termination across ATSUs or network transitions.
- Peer relationship establishment: Creating routing relationships between aircraft, ATSUs, and agents by exchanging control and routing metadata to support efficient handover and distributed communication.
- Presence and status monitoring: Continuously monitoring client and terminal status, updating or clearing records during disconnections or mobility events, and triggering recovery or continuity mechanisms as needed.
- Service Delivery: Managing discovery, registration, admission, and policy-aware routing for application layer services, ensuring reliable delivery without interpreting payloads.
- Mobility Support: Managing IP reassignment, context handover, and automatic reassociation of ATSUs and agents to maintain persistent sessions and roles as clients move across FIRs or change communication links (e.g., VHF ↔ SATCOM).

The following subsections provide the details of each management function.

2.1 Context Management

Context management is the first core capability of DLCM, responsible for organising, maintaining, and mirroring communication contexts to ensure stable and efficient service delivery. This function is implemented by the context management module described in Section 3.1.

A context is a logical container grouping related communication activities such as sessions, transactions, and operational metadata under a unified structure. Each context represents a distinct communication entity (e.g., an aircraft or ATC workstation), supporting tracking of ownership, operational state, and history across distributed nodes. This structure enables coordinated communication, continuity during handovers, and redundancy across stationary and mobile clients.

The context main structure (**Table 2.1**) defines the essential operational data for communication management, session control, and adjacency awareness. It consolidates static and dynamic information (active sessions, roles, routing, metadata, and history) to support efficient data handling in both normal and recovery scenarios.

Table 2.1: Context Main Structure

Field Name	Type	Description
context_id	string	Unique identifier for the context (e.g., delivery@saw.tr.atm, THY1AB@thy.tr.atm).
context_name	string	Human-readable name for the context (e.g., "Delivery Control").
context_type	string (Enum)	Identifies the type of context (e.g., ATC, Flight).
context_status	string (Enum)	Identifies the current operational status (**Table 2.2**).
context_connectivity	object	Contains node role assignments, Controlling, Mirroring, and Monitoring (**Table 2.3**).
context_routing	object	Contains routing information such as atc_agent_addr, gateways, etc (**Table 2.4**).
context_data	object	Contains session data and adjacent contexts (**Table 2.5**).
context_metadata	object	Describes context creation, updates, and additional metadata (**Table 2.6**).
context_history	array of objects	Tracks key context changes for audit, troubleshooting, and recovery (**Table 2.7**).

The context status is shown in **Table 2.2**.

Table 2.2: Context Status

Context Status	Description
REGISTERED	Context has been created and accepted by the ATM Server but not yet online
ONLINE	At least one peer is connected and operational; context is active
OFFLINE	All peers have disconnected; context is temporarily inactive but still defined
UNREGISTERED	Context has been explicitly terminated; removed from active records

Each context includes controlling, mirroring, and monitoring roles, which define how participating nodes share control and visibility (**Table 2.3**).

Table 2.3: Context Connectivity structure

Field Name	Type	Description
controlling_node_addr	string (IP Address)	IP address of the Controlling Node.
mirroring_nodes_addr	array of string	List of IP addresses assigned as Mirroring Nodes.
monitoring_nodes_addr	array of string	List of IP addresses assigned as Monitoring Nodes.

The controlling node is the primary workstation responsible for managing a communication context. It holds exclusive authority for directing message flows, executing operational commands, and maintaining overall control of the context. The controlling node executes all critical actions such as issuing instructions, handling clearances, and maintaining authoritative control over communication. Its main characteristics are:

- Serves as the main decision-maker, actively managing message transmission, data exchanges, and context updates.
- Only one controlling node is permitted per context at any given time.
- Has full read/write control at the Human-Machine Interface (HMI), enabling both data observation and operational input.
- All workstation requests and critical communication actions are routed exclusively through the controlling node.
- The CM Agent assigns and tracks the controlling node, while the ATC Agent ensures that messages directed to the context are delivered accordingly.
- During a planned handover or unexpected failure, a mirroring or monitoring node may request to assume the controlling role to maintain operational continuity.

The mirroring node provides multi-machine flexibility, allowing the same ATC user (or any authorised user) to operate across multiple workstations within a shared context. Unlike traditional standby configurations, each mirroring node functions as an active participant capable of full operational control, ensuring redundancy and seamless continuity. Its main characteristics are:

- Each mirroring node maintains read/write capabilities at the HMI, enabling control from any active workstation.
- No explicit handover or takeover is required as all mirroring nodes can handle communication and control actions.
- The CM Agent keeps all mirroring nodes synchronised, while the ATC Agent manages message routing and distributes data to all active mirroring nodes.
- Mirroring nodes operate independently but maintain synchronised data flow, ensuring continuous operation and redundancy.

The monitoring node operates as a passive observer within a communication context. Unlike controlling or mirroring nodes, it has read-only access to operational data and does not participate in communication control. Its primary function is to provide visibility and situational awareness to the operator, allowing observation of all relevant data without influencing communication flows. Its main characteristics are:

- The monitoring node receives a replicated data stream from the ATC Agent, providing full visibility into the context's operational state.
- Although it does not transmit messages or control context functions, it maintains an active connection to the ATC Agent to receive continuous updates.
- The CM Agent assigns the monitoring role to workstations joining a context that already has a controlling node or in situations requiring passive observation.
- Monitoring nodes are commonly used for backup, training, or supervisory purposes, enabling real-time observation without risk of interference.
- Each monitoring node remains synchronised with the active session state to ensure accurate and current data presentation.

The controlling role has full communication authority within a context, while mirroring and monitoring roles provide redundancy and situational awareness to ensure continuous and reliable operation. Routing information for each context is defined in **Table 2.4**.

Table 2.4: Context Routing Structure

Field Name	Type	Description
atc_agent_addr	string (IP Address)	IP address of the ATC Agent responsible for routing communications.
gateway_nodes (Optional)	array of string	List of gateway nodes for additional routing paths.
backup_routing_nodes (Optional)	array of string	Redundant routing nodes for failover handling.

Table 2.5: Context Data Structure

Field Name	Type	Description
associated_sessions	array of objects	List of active communication sessions within the context.
adjacent_contexts	array of string	List of related contexts for improved routing.

The context_data field (**Table 2.5**) maintains the most current operational state of a context, including all active sessions and related contexts used for optimised routing. This structure supports scalability, improves synchronisation, and ensures seamless integration with other components of the ATMACA system.

When an HMI is initialised, only the context_data field is retrieved. This ensures that controllers immediately access the most relevant and dynamic information without unnecessary overhead, optimising bandwidth usage and providing real-time visibility into active sessions and adjacent contexts. Metadata describing the origin and modification of each context is summarised in **Table 2.6**. These fields provide traceability by recording who created or updated a context, when it occurred, and any additional operational annotations.

Table 2.6: Context Metadata Structure

Field Name	Type	Description
created_by	string	Identifier of the client or user who created the context.
last_updated_by	string	Identifier of the client or user who last modified the context.
creation_timestamp	timestamp	Exact time the context was created.
last_updated_timestamp	timestamp	Time at which the context was last updated.
additional_info	object	Flexible metadata for comments, notes, or priority information.

Context history is used to track context changes and updates for audit, troubleshooting, and recovery activities (**Table 2.7**).

Table 2.7: Context History Structure

Field Name	Type	Description
event_id	string	Unique identifier for the recorded event.
event_type	string (Enum)	Describes the type of change (Node Change, Routing Update, etc.).
previous_value	string or object	The state before the change.
new_value	string or object	The state after the change.
change_reason	string (Optional)	Explains the reason for the change.
changed_by	string	Identifier of the user/system that triggered the change.
change_timestamp	timestamp	Time at which the change was recorded.
impact_scope (Optional)	string	Describes the scope of the change (e.g., Primary Only, All Nodes).

The context management message set defines the standardised set of messages used for managing context-related operations. These messages facilitate essential functions such as context creation, association, role management, metadata updates, and status queries. By establishing a structured communication framework between clients and CM Agents, these messages ensure seamless coordination of context activities, including role transitions, data synchronisation, and session continuity. Each message is designed with clear parameters and defined behaviours to maintain operational efficiency, scalability, and fault tolerance.

The context creation and association (**Table 2.8**) process involves the establishment and linking of contexts within the system, ensuring that participants (such as ATC stations or aircraft) are properly assigned to and integrated into specific operational environments. Context creation is the first step in initialising a new context, such as a new sector, area, or region, allowing it to be registered within the system. Once created, clients can associate with the context, assigning roles such as controlling, mirroring, or monitoring to participate in the operations. This ensures that the right participants are aligned with the correct operational contexts and are equipped with the appropriate permissions.

The format for the ATC context ID is sector@facility.icao.country.atm.timestamp, where:

- sector: name of the context (e.g., the sector's name like "Sector1").
- facility: facility or region where the sector is located (e.g., "ATCFacility").
- icao: The ICAO code for facility or operating region (e.g., "USA" for the United States, "CAN" for Canada).
- country: country code or name (e.g., "US" or "Canada").
- atm: constant to represent ATM (Air Traffic Management) system.
- timestamp: timestamp or date-time value, ensuring the uniqueness of the Context_ID over time.

The format for the flight context ID is: callsign@aircraft.icao.country.atm.timestamp, where:

- callsign: unique callsign of the aircraft (e.g., "AA123" for American Airlines flight 123).
- aircraft: static identifier for aircraft contexts.
- icao: ICAO code for the aircraft's registration or operating region (e.g., "USA" for the United States, "CAN" for Canada).
- country: country code for the aircraft's registration or the region it is operating in.
- atm: represents the ATM system.
- timestamp: timestamp to uniquely identify the aircraft's context at a given point in time.

Table 2.8: Context Creation and Association

Message Name	Direction	Purpose	Key Parameters	Behaviour
CONTEXT_CREATE_REQUEST	Client → CM Agent	Sent when a context does not exist and needs to be created.	Context_Name, Context_Type, Context_Owner, Initial_Metadata, ATC_Agent_Addr	CM AGENT registers the new context and instructs the client to log on to the ATC AGENT.
CONTEXT_CREATE_RESPONSE	CM Agent → Client	Confirms successful or failed context creation.	Status, Context_ID, Assigned_Role, Controlling_Address, ATC_Agent_Address	On success, the client proceeds with logon to the ATC AGENT; on failure, the client may retry or escalate.
CONTEXT_ASSOCIATION_REQUEST	Client → CM Agent	Request to associate with a context and assign a role.	Context_Name, Context_Owner, Context_Contact_Addr	Assigns a role (Controlling, Mirroring, or Monitoring); fails if the context doesn't exist.
CONTEXT_ASSOCIATION_RESPONSE	CM Agent → Client	Response to a CONTEXT_ASSOCIATION_REQUEST.	Status, Role, Controlling_Address, ATC_Agent_Address	On success, the client sends an ATTACH request to the ATC AGENT; otherwise, retry or create context.

Context maintenance ensures the ongoing health, security, and consistency of contexts within the operational system. It includes context consistency and synchronisation, which ensures that context data remains aligned across all participants and systems, and that any updates are properly synchronised. Context status monitoring is responsible for continuously tracking the health and activity of contexts, verifying that they remain operational and meet the required standards. Context security and integrity check protect the context data by ensuring its authenticity, security, and integrity, preventing unauthorised access or corruption. Should errors arise, context error handling and recovery is in place to detect issues and restore the context to a valid state, ensuring minimal disruption. Finally, context handover/takeover facilitates smooth transitions of responsibility for a context between different entities, ensuring operational flexibility and continuity. Collectively, these processes ensure that contexts remain stable, secure, and synchronised, supporting seamless and reliable operations.

Context Status Monitoring

Context status monitoring (**Table 2.9**) ensures the continuous health and performance of contexts within an operational environment. It involves tracking and managing the state of various contexts, such as sectors, facilities, or aircraft, to ensure that they are active, properly configured, and aligned with operational requirements. This process includes monitoring for context changes, detecting failures or anomalies, and ensuring that contexts remain synchronised across all participating entities. By continuously evaluating context status, operators can quickly identify issues, trigger recovery actions, and ensure smooth and uninterrupted operations within the system. Effective context status monitoring is essential for maintaining system integrity, optimising performance, and ensuring reliable and secure operations.

Table 2.9: Context Status Monitoring

Message Name	Direction	Purpose	Key Parameters	Behaviour
CONTEXT_STATUS_REQUEST	Client → CM AGENT	Requests the current status of a context.	Context_Name	Queries context details like Controlling, Mirroring, or Monitoring nodes.
CONTEXT_STATUS_RESPONSE	CM AGENT → Client	Provides the requested context's current status.	Current Controlling, Mirroring Nodes, Monitoring Nodes, Metadata	Details the context's active nodes and key information.

Context Consistency and Synchronisation

Effective synchronisation maintains consistent and accurate context data across multiple connected nodes. The CM Agent acts as the central authority responsible for ensuring that all active nodes, including controlling, mirroring, and monitoring nodes, remain updated with the latest context information. This synchronisation process is essential to support seamless communication, minimise data inconsistencies, and ensure operational continuity during node transitions, role changes, or recovery scenarios. The synchronisation workflow uses a combination of event-driven updates, periodic synchronisation, and on-demand refreshes to address various operational needs, ensuring that each node maintains the most recent context state while preventing conflicts and data loss (**Table 2.10**).

Event-Driven Synchronisation

When a client needs to modify context data such as changing the controlling_node_addr, it sends a CONTEXT_UPDATE_REQUEST to the system.

Upon receiving the request, the CM Agent verifies its validity and applies the necessary updates. It then broadcasts a CONTEXT_UPDATE_NOTIFY_REQUEST message to all relevant nodes.

Each of these nodes updates its local context cache accordingly and responds with a CONTEXT_UPDATE_NOTIFY_RESPONSE to confirm successful synchronisation of the updated context.

Periodic Synchronisation (Automated CM Agent Trigger)

The CM Agent periodically checks for nodes that may have fallen out of sync with the latest context data. It does this by comparing each node's last synchronisation timestamp against the most recent context update.

To confirm the synchronisation status, the CM Agent sends a CONTEXT_STATUS_REQUEST to the nodes. If any nodes report outdated information, they receive a CONTEXT_STATUS_RESPONSE prompting them to refresh their data.

To complete the update process, the CM Agent sends a CONTEXT_RETRIEVAL_REQUEST to the client, which then replies with a CONTEXT_RETRIEVAL_RESPONSE containing the latest context information.

Periodic Synchronisation (Automated Client Trigger)

Monitoring and mirroring nodes periodically pull context data to stay updated. This process begins when a client sends a CONTEXT_PULL_REQUEST to the CM Agent.

In response, the CM Agent provides the most recent context information by replying with a CONTEXT_PULL_RESPONSE.

On-Demand Synchronisation (Client-Initiated Sync)

When a client detects data inconsistencies, it can manually initiate context synchronisation by sending a CONTEXT_PULL_REQUEST to the CM Agent.

In response, the CM Agent returns the most up-to-date context data in a CONTEXT_PULL_RESPONSE, allowing the client to resolve the discrepancies.

Recovery Synchronisation (Node Restart/Recovery)

When a mirroring or monitoring node goes offline and later reconnects, it initiates a context update by sending a CONTEXT_PULL_REQUEST to the CM Agent. The CM Agent then responds with a CONTEXT_PULL_RESPONSE, providing the latest context data to ensure the node is fully synchronised upon rejoining the network.

Table 2.10: Context Consistency and Synchronisation

Message Name	Direction	Purpose	Key Parameters	Behaviour
CONTEXT_PUSH_REQUEST	Client → CM Agent	Request to upload or update context metadata.	Context_Name, Metadata	If successful, the workstation is automatically associated with the context.
CONTEXT_PUSH_RESPONSE	CM Agent → Client	Confirms successful context metadata upload.	Status	Confirms success or failure of the metadata update.
CONTEXT_PULL_REQUEST	Client → CM Agent	Request to download context metadata.	Context_Name	Requests metadata about a specified context.
CONTEXT_PULL_RESPONSE	CM Agent → Client	Provides the requested context metadata.	Metadata	Returns context details such as frequency, roles, and timestamps.
CONTEXT_UPDATE_REQUEST	Client → CM Agent	Request to update specific context fields.	Context_ID, Updated_Fields, Update_Reason, Requester_ID, Version, Force_Override	Updates selected fields; critical updates are logged in the Context_History table.
CONTEXT_UPDATE_RESPONSE	CM AGENT → Client	Confirms successful or failed update requests.	Status, Updated_Fields, Error_Details, Change_Timestamp	Provides details on the update result or conflicts.
CONTEXT_UPDATE_NOTIFY_REQUEST	CM Agent → Client	Request to update specific context fields.	Context_ID, Updated_Fields, Update_Reason, Requester_ID, Version, Force_Override	Updates selected fields; critical updates are logged in the Context_History table.
CONTEXT_UPDATE_NOTIFY_RESPONSE	Client → CM Agent	Confirms successful or failed update requests.	Status, Updated_Fields, Error_Details, Change_Timestamp	Provides details on the update result or conflicts.
CONTEXT_RETRIEVAL_REQUEST	CM Agent → Client	Requests retrieval of an existing context's details.	Context_ID, Requester_ID	CM AGENT retrieves the context details and responds with CONTEXT_RETRIEVAL_RESPONSE.
CONTEXT_RETRIEVAL_RESPONSE	Client → CM Agent	Provides the requested context details.	Context_ID, Metadata, Roles, Status, Last_Updated_Timestamp	Returns detailed information about the requested context.

Context Security and Integrity Check

Context security and integrity check (**Table 2.11**) maintains the reliability and trustworthiness of contexts within the operational environment. It involves verifying the security and authenticity of context data to ensure that unauthorised access or malicious activities are prevented. In addition, it focuses on checking the integrity of context data, ensuring that no corruption, tampering, or inconsistencies have occurred over time. Through these checks, operators can confirm that the context remains secure and accurate, preventing data breaches and ensuring the consistency of operations. This process helps safeguard sensitive operational data, supports compliance with security standards, and enhances the overall robustness of the system, making it essential for maintaining safe and efficient operational workflows.

Table 2.11: Context Security and Integrity Check

Message Name	Direction	Purpose	Key Parameters	Behaviour
CONTEXT_AUTHENTICATION_CHECK_REQUEST	CM Agent → Client	Request to verify the authentication and security of a context.	Context_ID, Requester_ID, Timestamp	CM Agent verifies context authenticity and security parameters.
CONTEXT_SECURITY_CHECK_RESPONSE	CM Agent → Client	Responds with the security status of the context.	Context_ID, Security_Status, Error_Code?, Last_Integrity_Check	Returns security status or failure details, including any security breaches.
CONTEXT_INTEGRITY_CHECK_REQUEST	CM Agent → Client	Request to check the integrity of the context data (i.e., corruption or alteration).	Context_ID, Requester_ID, Timestamp	CM Agent checks the context data for integrity violations, ensuring no corruption.
CONTEXT_INTEGRITY_CHECK_RESPONSE	CM Agent → Client	Responds with the integrity status of the context.	Context_ID, Integrity_Status, Error_Code?, Last_Integrity_Check	Returns integrity status and indicates if the context data is intact.
CONTEXT_OWNER_VALIDATION_CHECK_REQUEST	CM Agent → Client	Request to validate the owner of a context and their authorisation.	Context_ID, Owner_ID, Requester_ID, Timestamp	CM Agent validates if the owner is authorised to manage the context.
CONTEXT_OWNER_VALIDATION_CHECK_RESPONSE	CM Agent → Client	Responds with the validation status of the context owner.	Context_ID, Owner_ID, Validation_Status, Error_Code?	Returns the validation result or failure details for the context owner.

Context Error Handling Recovery

Context error handling and recovery (**Table 2.12**) maintains the stability and reliability of contexts within the operational system. It involves detecting errors, failures, or inconsistencies within a context and initiating recovery procedures to restore the context to a valid and functional state. This process ensures that when errors occur (whether due to data corruption, system faults, or unexpected disruptions), the system can automatically or manually detect and resolve issues to prevent operational downtime or performance degradation. By managing errors and triggering recovery actions, this process ensures the continuity of operations and minimises the impact of failures. Effective context error handling and recovery helps maintain the integrity of the system, ensures data consistency, and provides operators with the tools necessary to recover from faults efficiently and reliably.

Table 2.12: Context Error Handling Recovery

Message Name	Direction	Purpose	Key Parameters	Behaviour
CONTEXT_RECOVERY_REQUEST	Client → CM Agent	Request to initiate recovery for a faulty or corrupted context.	Context_ID, Recovery_Type, Recovery_Reason, Timestamp	Client requests CM Agent to initiate context recovery to restore the context to a valid state.
CONTEXT_RECOVERY_RESPONSE	CM Agent → Client	Confirms recovery success or failure for the context.	Context_ID, Recovery_Status, Recovery_Time, Error_Code?	CM Agent confirms the success of the recovery process or provides error details if recovery fails.

Context Handover/Takeover

Context handover/takeover (**Table 2.13**) maintains operational continuity and ensures smooth transitions within dynamic environments. It involves transferring the responsibility of a context, such as a control sector or operational task, from one entity to another. Context handover typically occurs when the current responsible party (e.g., an ATC operator or system) needs to pass control to another due to shifts in roles, geographical boundaries, or operational requirements. Conversely, context takeover occurs when a new entity assumes control, especially in cases of system failures, operator changeovers, or when the original entity becomes unavailable. This process ensures that responsibilities are seamlessly transferred without disrupting the ongoing operations, maintaining consistency and security within the system. By facilitating context handover/takeover, the system can adapt to changes, prevent service interruptions, and provide operational flexibility and resilience.

Table 2.13: Context Handover/Takeover

Message Name	Direction	Purpose	Key Parameters	Behaviour
CONTEXT_HANDOVER_REQUEST	Controlling Client → CM Agent	Initiates a planned Controlling-to-Mirroring handover.	Context_Name, Target_Addr	CM Agent assigns the new Controlling node and sends CONTEXT_ROLE_CHANGE_NOTIFY.
CONTEXT_HANDOVER_RESPONSE	CM Agent → Client	Confirms the successful role handover.	Status	Confirms the success or failure of the handover process.
CONTEXT_TAKEOVER_REQUEST	Mirroring/Monitoring Client → CM Agent	Request for a Mirroring/Monitoring node to assume Controlling role.	Context_Name	CM Agent assigns the requesting node as Controlling if no active Controlling node exists.
CONTEXT_TAKEOVER_RESPONSE	CM Agent → Client	Confirms the successful takeover.	Status	Confirms the success or failure of the takeover.
CONTEXT_ROLE_CHANGE_NOTIFY_REQUEST	CM Agent → Client(s)	Requests role change notification for Controlling, Mirroring, or Monitoring nodes.	New_Role	Informs clients about a new Controlling, Mirroring, or Monitoring node.
CONTEXT_ROLE_CHANGE_NOTIFY_RESPONSE	CM Agent → Client(s)	Confirms successful role change notification.	Context_Name, New_Role, Timestamp	Confirms that all clients are aware of the new role and updated context details.

Context Termination and Disassociation

Context termination and disassociation are essential processes in managing the lifecycle of contexts within the operational environment. Context termination refers to the process of formally ending a context's active status, rendering it inactive while preserving the data and structure for future reference (cleanup by policy or admin when a context is no longer needed). This is often necessary when a context is no longer needed or when operations related to the context have been completed. On the other hand, context disassociation involves the removal of a client or entity from a context, effectively

disconnecting it from the operational environment associated with that context. This action may be required when a participant no longer needs to interact with a particular context or when roles or responsibilities shift. Context termination and disassociation ensure that resources are effectively managed, outdated contexts are cleared, and the system remains flexible, efficient, and secure by removing unnecessary connections while maintaining operational integrity. **Table 2.14** lists client-initiated context disassociation and termination requests (Client ↔ CM Agent). Admin/policy cleanup and agent-initiated actions (e.g., marking a context OFFLINE when no peers remain, or agent-initiated session termination) are described in the presence/session management sections.

Table 2.14: Context Termination and Disassociation

Message Name	Direction	Purpose	Key Parameters	Behaviour
CONTEXT_DISASSOCIATE_REQUEST	Client → CM Agent	Request to leave a context.	Context_Name, Workstation IP	Removes the workstation from the context; assigns a new Controlling node if required.
CONTEXT_DISASSOCIATE_RESPONSE	CM Agent → Client	Confirms the disassociation request.	Status	Confirms removal of the workstation or informs about the new Controlling node.
CONTEXT_TERMINATE_REQUEST	Client → CM Agent	Request to terminate (inactivate) a context, making it inactive	Context_ID, Context_Owner, Termination_Reason, timestamp	Request for context termination
CONTEXT_TERMINATE_RESPONSE	CM Agent → Client	Confirm the context termination (inactivation) or provide failure details	Context_ID, Status, Termination_Time, error_code?	Response on context termination

2.2 Session Management

Session management ensures stable, secure, and efficient communication between network entities by maintaining detailed session information for both mobile nodes (e.g., aircraft) and stationary nodes (e.g., ATC clients). This function is implemented by the session management module (see Section 3.4.2).

Sessions are critical for managing communication states, ensuring continuity during mobility events, and preserving data integrity. The session ID is a unique identifier assigned to each session to differentiate it from other sessions within the same context. The session owner ID identifies the entity responsible for managing the session, ensuring accountability and tracking. A session is typically owned by a context. When a session is transferred from one session owner to another, only that specific session is moved while other sessions under the same context remain unaffected.

A session represents an active communication instance established between two endpoints such as an aircraft and an ATSU or between different ATC clients. Each session is uniquely identified and maintained to ensure seamless data exchange. It can be seen as an active logical communication link between peer node applications residing on distinct nodes while the physical connection required for it is established through the ATC Agent. Sessions are established within a context and include details like the Remote Context ID, Session Owner ID, and timestamp information. A context can contain multiple sessions.

Session Data Structure

The session data structure (**Table 2.15**) manages, organises, and maintains critical information related to communication sessions in data link environments. Sessions are created and managed inside a context,

which operates at the application layer and is transport-agnostic. Although this deliverable focuses on data link communications, the same model can be applied to other environments that involve endpoints, roles, and sessions. The session data structure ensures reliable, secure, and efficient data exchange between connected endpoints, such as aircraft systems, ATC workstations, and supporting network nodes. By consolidating key session-related parameters, this structure streamlines the management of session initiation, maintenance, handover, and termination.

The session data structure maintains crucial information required for session establishment, maintenance, and termination. It includes fields for session identification, ensuring each session is uniquely referenced throughout its lifecycle. The session owner and connected endpoint fields define the involved entities, while application context specifies the communication protocol in use, ensuring compatibility across different data link systems.

The structure also emphasises timing control, tracking the session's start and expiry times to manage connectivity lifecycles effectively. Session status management ensures real-time updates on session states such as active, inactive, or handoff, enabling precise monitoring and control. To improve resilience and reliability, the structure supports replication and backup by recording the status of primary and backup systems.

Table 2.15: Session Data Structure

Category	Field	Description	Example
Session Identification	Session ID	Unique identifier for the communication session	CPDLC-987654321
	Session Type	Behaviour, and operational context of a communication session distinguish between different types of communication flows to ensure effective management, resource allocation, and priority handling.	CPDLC Session, GRO session, FDIS session
	Session Owner	Identifies the initiating system (e.g., ATC/ACFT)	Aircraft: DAL101
	Connected Endpoint	Identifies the remote endpoint of the session	ATC: Maastricht UAC
	Application Context	Identifies the application associated with the session	CPDLC, DLIC, GRO
Timing Information	Session Start Time	The session’s creation/start timestamp	2025-02-20T12:30Z
	Session Expiry Time	The expiration timestamp of the session	2025-02-20T12:50Z
Session Status Management	Session Status	Tracks the current session state	Active, Inactive, Handoff, Terminated
Replication & Backup	Session Replication Status	Tracks if the session data is forked to a backup system	Primary: Active, Backup: Synchronising
Application Context Data Block	DLIC	manage and track the core communication details required for establishing and maintaining a reliable data link session between airborne and ground systems.	
	CPDLC	communication details are systematically stored, transmitted, and monitored to maintain flight safety, reduce voice communication workload, and improve data integrity.	

Application Context Data Block Structure

By integrating the CPDLC and DLIC data blocks, the session structure expands to manage CPDLC message exchange, aircraft flight information, DLIC connectivity details, and handover management through Current Data Authority (CDA) and Next Data Authority (NDA) fields. These additions ensure seamless communication handover between ATC units during long-haul flights or transitions across controlled airspace boundaries.

The DLIC data block is crucial in initiating data link sessions, enabling seamless information exchange between airborne and ground systems. It stores key details such as logon status, connection state, and active communication links, providing real-time visibility into session connectivity (**Table 2.16**). To enhance network reliability, the DLIC block also tracks multiple data links (e.g., SATCOM, VDL, LDACS) and defines link priorities for improved failover control.

Table 2.16: DLIC data block

Category	Field	Description	Example
Logon Information	Logon Status	Tracks the current DLIC logon state	Online, Offline
	Connection Status	Status of the data link connection	Connected, Disconnected
Communication Details	Communication Links	List of available data links	SATCOM, VDL, LDACS
	Active Link	Identifies the current active communication link	Active Link: SATCOM
	Link Performance Metrics	Tracks latency, bandwidth, and packet loss	Latency: 200ms, Bandwidth: 64 kbps
	Link Priority	Preferred link order for redundancy and failover	Primary: SATCOM, Backup: VDL
Network & Intermediate Nodes	Intermediate Node List	List of network nodes involved in communication	[Node 1: Router_A, Node 2: Relay_X]
	Active Intermediate Node	Current active node handling the data path	Node 1: Router_A
	Intermediate Node Info	IP addresses, ports, or IDs for each intermediate node	[2001:db8:85a3::1, Port 4500]
Link Health Information	Link Quality Metrics	Provides real-time link health and quality details	Latency: 200ms, Packet Loss: 0.1%
	Connection Failover Status	Indicates failover behaviour during link interruptions	Failover: Auto Switched to VDL
Security & Authentication	Encryption & Authentication Tokens	Ensures secure data exchange	TLS Certificate ID: X.509_ABC123
	Access Control Information	Defines permissions for session data management	Permission: ATC, Airline Ops
	Message Integrity Checks	Ensures data integrity and authenticity	HMAC-SHA256: 3f9a9a5c...

The CPDLC data block is essential for transmitting critical flight instructions, clearances, requests, and information updates in both domestic and oceanic airspace. It maintains key communication elements such as pending messages, message history, and message sequencing to ensure accurate tracking of interactions between pilots and controllers (**Table 2.17**). By supporting structured messaging, the CPDLC data block enhances safety by minimising miscommunication risks and improving situational awareness. More details about the CPDLC application and messages are available in D4.2 [11].

In addition to message management, the CPDLC data block also tracks essential flight information, such as the flight number, aircraft registration, and route details, ensuring all relevant operational data is linked to the session. It further supports ATC handovers by maintaining records of the CDA and NDA, enabling seamless transitions as aircraft move across regional airspace boundaries.

Table 2.17: CPDLC Data Block

Category	Field	Description	Example
Flight Information	Flight Number	Unique identifier for the flight	DAL101
	Aircraft Registration	Aircraft's registration code	N123AB
	Aircraft Type	Model and variant of the aircraft	Boeing 777-300ER
	ICAO 24-bit Address	Aircraft's unique transponder code	A123BC
	Departure Airport	Origin airport ICAO/IATA code	JFK (KJFK)
	Destination Airport	Destination airport ICAO/IATA code	LHR (EGLL)
	Alternate Airport	Alternate airport for emergencies	LGW (EGKK)
	Route Information	Planned flight path with waypoints	DCT GREKI NATR2 MALOT DCT
	Current Position	Real-time aircraft position in latitude/longitude	N43°15' W050°24'
	Next Waypoint	The next planned waypoint	MALOT
	Estimated Time of Arrival (ETA)	Projected arrival time at the destination	2025-02-20T16:45Z
	Departure Time	Actual departure time from the origin	2025-02-20T12:30Z
	Flight Level	Current altitude in flight levels	FL380
	FIR Boundaries	List of FIRs crossed during the flight	KZNY → CZXQ → EGGX → EGTT
	Squawk Code	Assigned transponder code	7500 (Emergency)
	Flight Status	Current operational status	Active
	Airline Operator	Airline operating the flight	Delta Airlines
Message Management	Pending Messages	CPDLC messages awaiting pilot/ATC response	DESCEND TO FL320
	Message History	Tracks exchanged CPDLC messages for audit and traceability	CLIMB TO FL380
ATC Authority Management	ATC Unit	Current ATC unit managing the session	ATC Unit
	FIR Code	ICAO FIR code for the CDA's/NDA's jurisdiction	FIR Code
	Role	Defines the CDA's role in operations	Role
	VHF Address	VHF frequency for CDA/NDA communication	VHF Address
	Contact Info	Emergency contact details for the CDA/NDA	Contact Info
	Control Boundary	Defined geographical boundary for the CDA/NDA's jurisdiction	Control Boundary
	Handoff Conditions	Conditions required to trigger CDA/NDA handover	Handoff Conditions

Session Creation/Initialisation

Session creation and initialisation ensures secure, efficient, and structured data exchange between aircraft and ground systems. Each session acts as a dedicated communication channel, uniquely identified to track and manage interactions throughout its lifecycle. During session creation, key parameters such as the Session ID, Session Owner, and Remote Endpoint are established, enabling message routing, resource allocation, and continuity management (**Table 2.19**). The session initialisation process also incorporates essential security mechanisms, ensuring data integrity, authenticity, and resilience against disruptions. By adopting a well-defined session creation framework, ATMACA facilitates reliable air-ground communication across diverse applications, including CPDLC, DLIC, and GRO (see example in **Table 2.18**), while supporting seamless transitions between ATSUs and maintaining compliance with ICAO and SESAR guidelines.

Table 2.18: Session Format

Application	Session ID Format	Example
CPDLC	CPDLC-LFBB-ATSU01-LECB-ATSU02-DAL101-20250319120030-a3d9b8f6	Identifies a CPDLC session for flight DAL101.
GRO	GRO-LFBB-ATSU01-LECB-ATSU02-DAL303-20250319123030-c7a4f2b5	Identifies a GRO session for flight DAL303.

The session format and creation process can be summarised as follows:

[App ID] - [Session Owner ID] - [Session Remote ID] - [Flight ID] - [Timestamp] - [Random Hash]

Step 1: Identify application type

- Extract the App ID from the initial message request (e.g., CPDLC, GRO, etc.).

Step 2: Assign session ownership

- The ATSU that receives the initial request, or initiates the request, assigns itself as the Session Owner.
- Assign the opposing node as the Session Remote Endpoint.

Step 3: Generate unique identifiers

- Capture the Flight ID (e.g., DAL101).
- Append the timestamp for chronological uniqueness.
- Generate a random hash to guarantee uniqueness in congested environments.

Table 2.19: Session Creation and Initialisation

Message Type	Direction	Purpose	Key Contents
SESSION_CREATE_REQUEST	Client → CM Agent	Initiate a new session between two entities	context_id, remote_id, app_id, flight_id, requested_role, token, timestamp
SESSION_CREATE_RESPONSE	CM Agent → Client	Confirm session creation and return session metadata	session_id, status, assigned_role, routing_info, heartbeat_interval
SESSION_START_REQUEST	Client → Remote Client (peer)	Inform the remote peer that session is now starting	session_id, app_id, start_time, initiator_info, token
SESSION_START_RESPONSE	Remote Client → Client (peer)	Acknowledge that the session is accepted and started	session_id, status, accept_time, capabilities

Session Maintenance

Session maintenance ensures that established sessions remain stable, secure, and functional throughout their lifecycle. During this phase, the system actively monitors session states, handles message exchanges, and dynamically updates session parameters in response to network changes or operational events. Key activities in session maintenance include periodic heartbeat signals to verify connectivity, data synchronisation between the session owner and remote endpoint, and adaptive resource management to maintain performance. Additionally, session maintenance incorporates mechanisms to support seamless transitions during FIR boundary crossings, ensuring communication continuity with minimal disruption. By implementing robust maintenance strategies, ATMACA enhances the reliability, resilience, and efficiency of air-ground communication systems, even in high-mobility aviation environments.

Status monitoring and connectivity messages track session health, confirm connectivity status, and detect potential disruptions to ensure reliable communication between entities (**Table 2.20**).

Table 2.20: Session Monitoring and Connectivity

Message Type	Direction	Purpose	Key Contents
SESSION_STATUS_REQUEST	Client → Remote Client (peer)	Request peer's view of session state	session_id, requester_id, app_id, status_scope, timestamp
SESSION_STATUS_RESPONSE	Remote Client → Client (peer)	Provide session status as seen locally	session_id, status, last_active, capabilities, errors[]
SESSION_CONTINUITY_REQUEST	Client → Remote Client (peer)	Re-establish sync after reconnection, handover, or role switch	session_id, origin_id, last_seen_state, recovery_token, timestamp
SESSION_CONTINUITY_RESPONSE	Remote Client → Client (peer)	Acknowledge sync status, resend deltas if needed	session_id, continuity_result, missing_messages[], sync_ok, resync_required

Synchronisation and session management messages ensure seamless coordination, data consistency, and session continuity between communicating entities by managing context updates, parameter changes, and message recovery (**Table 2.21**). A forked context refers to a session that has been duplicated into another context. After forking, the forked context periodically sends SESSION_REFRESH_REQUEST to the CM Agent to keep the fork alive. In this table, "Controlling Context" or "Forked Context" denotes the client representing that context's role (controlling, mirroring, or monitoring) when exchanging messages with the CM Agent.

Table 2.21: Synchronisation and Session Management

Message Type	Direction	Purpose	Key Contents
SESSION_SYNC_REQUEST	Client → CM Agent	Request full or delta sync of session state	session_id, context_id, sync_type, last_known_state, timestamp
SESSION_SYNC_RESPONSE	CM Agent → Client	Return full session state snapshot	session_id, owner, participants, state_hash, sync_payload[]
SESSION_DATA_PULL_REQUEST	Client → CM Agent	Request app-level session data	session_id, request_scope, filter_by_app, since_timestamp
SESSION_DATA_PULL_RESPONSE	CM Agent → Client	Return requested session data content	session_id, data_payload, last_updated, content_size
SESSION_DATA_PUSH_REQUEST	Client → CM Agent	Submit new application-specific data	session_id, app_id, data_block, sequence_no, source_id, timestamp
SESSION_DATA_PUSH_RESPONSE	CM Agent → Client	Acknowledge or reject the pushed data	session_id, status, ack_timestamp, error_code?
SESSION_PARAMETER_UPDATE_REQUEST	Client → CM Agent	Modify session-level parameters	session_id, updated_params[]
SESSION_PARAMETER_UPDATE_RESPONSE	CM Agent → Client	Confirm parameter update or return error	session_id, status, applied_params, error_code?
SESSION_DATA_UPDATE_REQUEST	Client → CM Agent	Request update to previously submitted session data	session_id, data_block_id, new_data, reason, timestamp
SESSION_DATA_UPDATE_RESPONSE	CM Agent → Client	Acknowledge or reject session data update	session_id, status, updated_block_id, version, error_code?
SESSION_REFRESH_REQUEST	Forked Context → CM Agent	Periodically reassert presence in forked session	session_id, context_id, role, timestamp, heartbeat_counter, token
SESSION_REFRESH_RESPONSE	CM Agent → Forked Context	Acknowledge refresh or signal expiration/detachment	session_id, status, next_refresh_due, session_state, error_code?
SESSION_FORK_REQUEST	Controlling Context → CM Agent	Request to fork session to another context	session_id, target_contexts[], requested_roles[], sync_mode, initiator_id, reason, timestamp

SESSION_FORK_RESPONSE	CM Agent → Controlling Context	Confirm session fork and deliver state or error	session_id, fork_status, assigned_contexts[], sync_payload, errors[]?
SESSION_MERGE_REQUEST	Controlling Context → CM Agent	Merge multiple sessions into one unified session	session_ids[], merge_mode, initiator_context, reason, timestamp
SESSION_MERGE_RESPONSE	CM Agent → Controlling Context	Confirm merged session or reject with reasons	merged_session_id, status, errors[], new_participants[], apps[]
SESSION_AGGREGATE_REQUEST	Controlling Context → CM Agent	Request logical grouping of sessions	aggregate_by, filters, requester_context, timestamp
SESSION_AGGREGATE_RESPONSE	CM Agent → Controlling Context	Return logical aggregation of matching sessions	aggregation_id, session_ids[], contexts[], apps[], status
SESSION_LOST_MSG_RETRIEVAL_REQUEST	Client → Remote Client (peer)	Request retransmission of missing messages	session_id, missing_sequence_ids[], app_id, timestamp, sender_context_id
SESSION_LOST_MSG_RETRIEVAL_RESPONSE	Remote Client → Client (peer)	Deliver requested messages or notify which ones couldn't be recovered	session_id, recovered_data[], not_found_ids[], status, timestamp

Security and integrity control messages ensure session authenticity, data integrity, and secure communication by managing token updates, encryption key refreshes, and identity validation procedures (**Table 2.22**).

Table 2.22: Security and Integrity Control Messages

Message Type	Direction	Purpose	Key Contents
SESSION_AUTH_REQUEST	Client → CM Agent	Authenticate the session based on token, role, and context	session_id, token, context_id, role, timestamp, auth_type
SESSION_AUTH_RESPONSE	CM Agent → Client	Confirm or reject session authentication	session_id, auth_status, reason_code, granted_role, valid_until
SESSION_TOKEN_REFRESH_REQUEST	Client → CM Agent	Request new session token after timeout or renewal window	session_id, current_token, context_id, requester_id, timestamp
SESSION_TOKEN_REFRESH_RESPONSE	CM AGENT → Client	Return updated token and expiry metadata	session_id, new_token, valid_until, error_code?
SESSION_INTEGRITY_CHECK_REQUEST	Client → CM Agent	Verify session data integrity or consistency (for audit/debug)	session_id, context_id, hash_method, data_block_refs, timestamp
SESSION_INTEGRITY_CHECK_RESPONSE	CM Agent → Client	Report integrity status (OK / mismatch / expired)	session_id, integrity_status, mismatched_blocks[], last_verified, hash_signature
SESSION_ENCRYPTION_REFRESH_REQUEST	Client → CM Agent	Request to refresh encryption settings (e.g., new keys, algorithm change)	session_id, current_encryption_key, requested_encryption_method, timestamp
SESSION_ENCRYPTION_REFRESH_RESPONSE	CM Agent → Client	Return updated encryption settings or error	session_id, new_encryption_key, valid_until, status, error_code?

Error handling and recovery messages detect, report, and resolve communication failures, session disruptions, and data inconsistencies, ensuring session stability and continuity in challenging operational conditions (**Table 2.23**). Recovery is for CM Agent-mediated repair after disruption and Resync corrects

peer-to-peer data mismatches. Timeout Warning notifies peers that expiry is approaching and SESSION_SYNC_ provides the CM Agent's authoritative state snapshot during routine operation.

Table 2.23: Error Handling and Recovery Messages

Message Type	Direction	Purpose	Key Contents
SESSION_TIMEOUT_WARNING	Client → Remote Client (peer)	Notify the peer that the session is nearing timeout or expiration	session_id, timeout_warning_time, session_state, timestamp
SESSION_RECOVERY_REQUEST	Client → CM Agent	Request recovery of a session after disruption (e.g., network failure)	session_id, context_id, failure_reason, timestamp
SESSION_RECOVERY_RESPONSE	CM Agent → Client	Confirm recovery process or provide error details	session_id, recovery_status, error_code?, new_session_state
SESSION_RESYNC_REQUEST	Client → Remote Client (peer)	Request synchronisation after a mismatch or data inconsistency	session_id, context_id, mismatch_info, timestamp
SESSION_RESYNC_RESPONSE	Remote Client → Client (peer)	Confirm synchronisation or provide errors	session_id, sync_status, error_code?, updated_data

Session Handover/Takeover

Session transfer, also known as session handover, ensures seamless continuity of established communication sessions as aircraft transition between different ATSUs or network domains. During a handover, the active session state, including the Session ID, context information, and communication parameters, is securely transferred from the current Session Owner to the new ATSU, which assumes the role of the Remote Endpoint (**Table 2.24**). This process is designed to maintain data integrity, minimise latency, and ensure ongoing CPDLC or other applications' communication without interruption. The ATMACA protocol uses proactive synchronisation, session mirroring, and failover mechanisms to ensure the handover process is smooth and resilient to network delays or failures. "Target Context" denotes the client representing that context role.

Table 2.24: Session Handover/Takeover

Message Type	Direction	Purpose	Key Contents
SESSION_HANDOVER_REQUEST	Session Owner → Target Context	Request to hand over session control to another client (target context)	session_id, target_context_id, handover_reason, timestamp
SESSION_HANDOVER_RESPONSE	Target Context → Session Owner	Confirm the handover request or provide failure reason	session_id, status, target_context_id, handover_time, error_code?
SESSION_TRANSFER_COMPLETED_REQUEST	Session Owner → Target Context	Request confirmation that session transfer has been successfully completed	session_id, target_context_id, transfer_time, status
SESSION_TRANSFER_COMPLETED_RESPONSE	Target Context → Session Owner	Confirm that session transfer is complete and the target context is now in control	session_id, status, transfer_complete_time, new_owner_id
SESSION_TAKEOVER_REQUEST	Session Owner → Target Context	Request takeover of session control from a failed or inactive context	session_id, takeover_reason, timestamp
SESSION_TAKEOVER_RESPONSE	Target Context → Session Owner	Confirm session takeover or provide error details	session_id, status, takeover_time, error_code?
SESSION_OWNER_CHANGE_REQUEST	Client → CM Agent	Request to change or identify the session owner	session_id, owner_id, context_id, timestamp
SESSION_OWNER_CHANGE_RESPONSE	CM Agent → Client	Confirm the session owner change or provide error information	session_id, owner_id, status, timestamp
SESSION_OWNER_CHANGE_NOTIFY_REQUEST	CM Agent → Client	Notify participants of the change in session ownership	session_id, new_owner_id, status, notification_time

SESSION_OWNER_CHANGE_NOTIFY_RESPONSE	Client → CM Agent	Acknowledge the notification of the session owner change	session_id, status, acknowledgment_time, error_code?
SESSION_OWNER_VALIDATION_REQUEST	Client → Remote Client (peer)	Request to validate the current session owner	session_id, requester_id, timestamp
SESSION_OWNER_VALIDATION_RESPONSE	Remote Client → Client (peer)	Respond with the validation status of the current session owner	session_id, is_valid_owner, owner_id, validation_time, error_code?

Session Termination

Session termination (**Table 2.25**) formally ends an established communication session between participating entities, such as ATSUs, aircraft, or CMAs. Termination ensures that session resources are released, security keys are invalidated and pending data exchanges are either completed or discarded. The termination process may be initiated by the Session Owner, the Remote Endpoint, or automatically triggered due to timeout, or network failure. Proper session termination procedures are essential to maintain communication integrity, prevent resource leaks, and ensure accurate system state management. By implementing structured termination protocols, ATMACA effectively manages session lifecycle control, improves security, and ensures reliable air-ground communication.

Graceful session shutdowns use the SESSION_END_REQUEST / RESPONSE message pair, initiated by the client when session operations are successfully completed. This flow ensures orderly deallocation of session resources and proper acknowledgment from the remote participant.

In contrast to client-initiated shutdown, the SESSION_TERMINATE_REQUEST is reserved for unexpected session terminations and is exclusively initiated by the ATC Agent. This message is used when a session ends abnormally, most commonly when the originating client becomes unreachable due to events such as a context-wide disconnection, watchdog timeout, or system failure. In such cases, the ATC Agent acts as a proxy for the lost peer and transmits the SESSION_TERMINATE_REQUEST to the remote session participant to formally indicate that the session is no longer valid. The termination_reason field is explicitly set (e.g., CONTEXT_UNREACHABLE, PEER_DISCONNECTED) to enable the remote peer to carry out appropriate cleanup and avoid relying on stale session state. This mechanism ensures session-level consistency, prevents orphaned or dangling sessions, and enables clean recovery behaviour across the distributed ATMACA network.

Table 2.25: Session Termination

Message Type	Direction	Purpose	Key Contents
SESSION_END_REQUEST	Client → Remote Client (peer)	Request to end the session, typically after operations are completed	session_id, end_reason, timestamp
SESSION_END_RESPONSE	Remote Client → Client (peer)	Confirm the session end request, providing status and completion time	session_id, status, end_time, error_code?
SESSION_TERMINATE_REQUEST	CM Agent → Client	Request to terminate an active session, typically after operation or on error	session_id, termination_reason, timestamp
SESSION_TERMINATE_RESPONSE	Client → CM Agent	Confirm session termination, provide status and completion info	session_id, status, termination_time, error_code?

2.3 Connection Management

The connection management capability of DLCM is responsible for establishing, supervising, and maintaining communication links between clients, agents, and the ATM Server integrating ATC Clients, FlightDeck Clients, and Application Servers into the ATMACA network via their designated agents. Through robust signalling flows, presence monitoring, and structured state management, it provides the critical

backbone for resilient, secure, and context-aware communication across all entities. This function is implemented by the connection management module (Section 3.3).

A key aspect of this function is the capability exchange process, which allows clients and agents to advertise and negotiate their supported applications, roles, and protocol capabilities with the ATM Server. This initial negotiation ensures compatibility and readiness before session establishment.

To maintain connection integrity and liveness, ATMACA implements a watchdog mechanism based on periodic heartbeat exchanges. These device-watchdog messages are exchanged between agents and the ATM Server to detect connection loss and trigger recovery actions when needed.

Additionally, the protocol includes a structured disconnection process using disconnect-peer messages, allowing any node to gracefully terminate its connection with the ATM Server or its associated agent, ensuring a clean state without orphaned sessions or stale peer data.

These mechanisms define a robust set of procedures for node registration, logon, attachment, updates, contact transitions, ensuring seamless integration and operational continuity across the ATMACA network.

Peer Connection and Peer Structure

Peer connection management enables nodes (whether clients, agents, or servers) to establish and maintain secure, structured communication pathways across the network. Each connection represents a trusted relationship between two entities and is governed by initial capability exchange, ongoing watchdog monitoring, and graceful disconnection handling. To effectively manage these interactions, each node maintains a structured view of its peers, capturing essential attributes such as identity, IP address, session bindings, operational role, and state (**Table 2.27**). This peer structure (**Table 2.26**) enables dynamic routing decisions, presence awareness, and protocol-level coordination, ensuring robust and fault-tolerant connectivity within highly dynamic air traffic environments.

Table 2.26: Peer Structure

Field	Description
Peer-ID	NodeHost+NodeRealm
Peer-Handle	Unique peer handle
Peer-Name	NodeHost
Peer-State	**Table 2.27**
Peer-Type	"None", "Dynamic", "Static"
Peer-Name-Assigned	bool
Peer-Group-Assigned	bool
Peer-Group_Name	A logical name assigned
Peer-ConnAttempCounter	Value from config file
Peer-KeepAliveCounter	Value from config file
RemoteRealm	NodeRealm
RemoteAddress	Latest known IP address
RemotePort	Remote peer port
LocalAddress	Local Address
LocalPort	Local Port
Remote-NodeRole	Functional role: ATM SERVER, APPLICATION SERVER, ATC AGENT, CM AGENT, ATC CLIENT, FD CLIENT
Remote-NodeType	SERVER, AGENT, CLIENT
User-Role	Contextual role: Controlling, Mirroring, Monitoring, or Peer
TransportProt	Transport layer details (e.g., Transmission Control Protocol (TCP):5910, Stream Control Transmission Protocol (SCTP):3868)
Session-ID(s)	Active session identifiers with this peer

Assigned-Context(s)	Contexts this peer is part of (e.g., flight or sector identifiers)
Supported-Apps	List of supported applications (e.g., CPDLC, DLIC, SWIM)
Capabilities	Features, protocol options, vendor-specific extensions
Watchdog-Timestamp	Time of last successful DWR/DWA exchange
MaintenanceState	"None", "Created", "Connected", "Locally Disconnected", "Remotely Disconnected", "Deleted", "Cancelled"

Table 2.27: Peer State

State	Description
PEER_NONE	Default/uninitialised state — no peer session or object instantiated yet
PEER_LOCALLY_DISCONNECTED	Peer disconnected by the local node (e.g., agent shut down, user-initiated detach)
PEER_REMOTELY_DISCONNECTED	Peer connection lost due to remote party failure (timeout, transport loss)
PEER_CONNECTED	Peer is actively connected (e.g., after Attach or Logon)
PEER_CREATED	Peer object created and initialised, but not yet attached or connected
PEER_CANCELLED	Peer operation aborted before completion (e.g., registration failed)
PEER_DELETED	Peer object is removed entirely (e.g., after termination or cleanup)

Forwarding and Routing Process

The forwarding and routing process in the ATMACA protocol governs how messages are delivered between nodes across the distributed ATM communication network. It is designed to support dynamic peer discovery, efficient message delivery, and reliable fallback mechanisms in highly mobile and fault-tolerant environments. At its core, the system uses a two-tiered resolution model: the forwarding table, which caches active peer connections for low-latency delivery, and the routing table, which provides a broader view of reachable realms, sectors, and adjacent agents. When a message is received, the system first attempts to resolve the destination through the forwarding table (peer table), as shown in **Table 2.26**, using a known Peer-ID or Host@Realm. If no direct match is found, it queries the routing table, as shown in **Table 2.28**, to identify the appropriate adjacent agent, then re-evaluates the forwarding table using the resolved next hop. This layered approach enables ATMACA to adapt dynamically to changing network conditions, maintain session continuity, and ensure message delivery even across complex and evolving operational topologies.

Table 2.28: Routing Table Structure

Field	Description
Destination-Realm	Domain of the target peer (e.g., ltfm.atc.tr)
Context-Name	(Optional) Target context or sector name
NextHop-Peer-ID	Primary peer ID for message delivery
NextHop-Peer-Name	Fully qualified hostname of the next hop peer
Alternate-Peer-ID	Backup peer ID
Alternate-Peer-Name	Backup peer hostname
Route-Origin	Static, Dynamic, etc.
Route-Priority	Preference score
Last-Validated	Timestamp of last successful check

Connection Management Messages

The ATMACA connection management message defines the structured set of messages used to establish, maintain, and manage network connectivity across clients, agents, and servers within the ATMACA environment. These messages govern the full connection lifecycle, including node registration, logon, attachment, disconnection, and updates.

Authorisation is handled as an integrated part of the registration process: when a node submits a REGISTRATION_REQUEST to the ATM Server, it includes its role, realm, identity, and supported capabilities. The ATM Server validates these claims against predefined operational policies and access rules, and responds via REGISTRATION_RESPONSE with either approval, adjusted role assignment, or a rejection accompanied by a reason. This embedded authorisation model ensures that only valid and trusted ATMACA nodes participate in the system, reducing complexity while strengthening security and domain control.

The connection messages are grouped into three categories as shown in **Table 2.29**:

Table 2.29: Connection Management Message Set

Name	Scope
Registration Messages	Between ATM Server and all ATMACA nodes
Operational Connection Messages	Between Clients, Application Servers, and ATC Agents
Protocol-Level Infrastructure Messages	Between all nodes (Agents, Servers, Clients) using base protocol mechanisms like CER, DWR, DPR

Registration is the initial procedure through which all ATMACA nodes formally introduce themselves to the ATM Server. This process establishes the identity, declared role, realm, and supported capabilities of each node (**Table 2.30**). As part of this exchange, the ATM Server conducts built-in authorisation checks to ensure the requesting node is permitted to operate within the declared domain and role. Authorisation is based on predefined contexts, not individual user identities. Each context, such as an ATC sector or a flight, is validated by the ATM Server during the registration process to ensure it is recognised and permitted for operational use.

For ATC Clients, the declared sector is checked against the ATM Server's internal sector table. If the sector exists and is valid for operational assignment, the ATM Server returns the appropriate ATC Agent address, allowing the client to continue with the logon procedure. For aircraft, the flight context is validated against authoritative filed flight plan data. The ATM Server checks the registration request using parameters such as call sign, departure and destination airports, off-block time, aircraft registration number, and flight date. If the flight matches an authorised flight plan filed in advance, the context is considered valid, and the ATM Server provides the assigned ATC Agent address for initial logon and communication setup. This context-centric authorisation model ensures that only valid, operationally authorised entities participate in ATMACA communication, preserving consistency and trust across the network.

The registration function in the ATMACA protocol integrates nodes into the operational communication network. It ensures that both mobile clients (such as aircraft and mobile ATC systems) and stationary clients (such as ATC workstations, ATC Agents, CM Agents, and Application Servers) are properly onboarded and aligned with the system's architecture. Rather than authenticating individual users, ATMACA authorises predefined contexts by validating them against authoritative internal data.

Upon successful registration, the ATM Server responds with provisioning data tailored to the node type. This includes the address of the assigned ATC or CM Agent, adjacent agent relationships, and any necessary configuration parameters required to begin operational communication. The registration function also ensures proper assignment of each client to its responsible agent, which is critical for managing context ownership, session initiation, routing, and mobility support. This centralised registration and provisioning mechanism establishes a trusted framework that enables secure participation, coordinated configuration, and seamless integration of all nodes before any active data exchange occurs.

Table 2.30: Registration Message

Message Name	Direction	Purpose	Key Parameters
REGISTRATION_REQUEST	Node → ATM Server	Advertise node identity, capabilities, and roles	NodeRole, NodeHost, NodeRealm, SupportedApps, Capabilities
REGISTRATION_RESPONSE	ATM Server → Node	Assign agent info and operational provisioning	AssignedAgent, CMAddress, AdjacentAgentList

Agents are also identified, authenticated, and provisioned by the ATM server for further communication management. Upon successful registration, the server transfers all necessary provisioning data to the agent, enabling it to manage client connectivity, routing, and context data effectively.

A stationary node (i.e., an ATC Agent or CM Agent), in the ATMACA network, is responsible for managing physical connections and ensuring seamless data flow. The agent's responsibilities include:

- Session Handling: Agents establish the physical communication link that allows logical sessions between peer applications to operate successfully.
- Context Data Preservation: Since all transactions pass through the agent, the agent preserves vital context data and its associated session details.
- Session Transfer Support: When a session is transferred from one session owner to another, the agent tracks this movement and marks the transferred session with a timestamp for historical tracking.
- Update Management: If a context's data changes, the client must send an UPDATE request message to the agent with the updated information. The agent ensures the data is correctly updated and preserved.

By maintaining these responsibilities, the agent plays a pivotal role in ensuring stable, secure, and organised communication across the ATMACA network.

To ensure consistency across the ATMACA network, every configuration update managed by the ATM Server (e.g., changes to sector ownership, peer relationships, or routing logic) is tracked using a centralised provisioning data version. This versioning mechanism allows agents and clients to verify whether they are operating with the most current provisioning data and to initiate synchronisation procedures when discrepancies are detected. The ATM Server acts as the authoritative source for version control, incrementing the version number whenever relevant configuration data is modified.

Upon registration or during subsequent provisioning exchanges, the ATM Server provides the ProvisionDataVersion along with all provisioning data (**Table 2.31**) to the requesting node. Each ATC Agent or CM Agent caches this version locally as part of its provisioning metadata.

The version is also included in messages such as PROVISION_UPDATE_REQUEST, PROVISION_QUERY_RESPONSE, and PROVISION_STATUS_RESPONSE, enabling nodes to compare the server's active version against their local copy. If the versions differ, agents can initiate a PROVISION_QUERY_REQUEST to retrieve the latest data or trigger a PROVISION_UPDATE_RESPONSE to confirm update receipt.

The provisioning data (**Table 2.31**) is the authoritative, versioned operational dataset that the ATM Server distributes (e.g., sectors, peer/adjacency, assigned agent, facility lists). It may include fields such as

the ProvisionDataVersion identifier, EffectiveTime, update Scope, and an optional checksum for data integrity validation. It may also carry the UpdatedBy field to identify the origin of the change and a ChangeLogID to correlate the update with an audit trail or configuration log. For example, a provisioning record might specify version "3.2.1" with an effective time of "2025-03-30T08:00:00Z", covering scope areas like "Sectors" and "PeerMap", and referencing a change log such as "cfg-20250330-001".

This versioning model plays a critical role during synchronisation and recovery. If an agent restarts or detects stale configuration during a peer exchange, it can compare its local ProvisionDataVersion against the ATM Server's current version and take corrective action accordingly. By maintaining a shared versioning system across all nodes, the ATMACA protocol ensures robust consistency, fault-tolerant synchronisation, and traceable configuration lifecycle management. Provisioning data is the centrally versioned part of configuration (distributed by the ATM Server); configuration also covers local static settings not in the server's provisioning bundle.

Table 2.31: Provisioning Data Structure

Field	Purpose
ProvisionDataVersion	Version identifier (semantic versioning or timestamp-based).
EffectiveTime	When this version became active.
Scope	Which components were affected (Sectors, Peers, Facilities, etc.).
Checksum	Optional hash of payload for integrity validation.
UpdatedBy	Identifier of who or what initiated the update.
ChangeLogID	Reference to a human-readable change log or audit entry.

ATC Agent Registration

Upon initialisation, the ATC Agent initiates a REGISTRATION_REQUEST to the ATM Server to announce its presence and begin participating in the ATMACA network. In response, the ATM Server returns a comprehensive provisioning package containing all necessary configuration data required for the agent to operate within its assigned domain.

This provisioning package includes several key elements: a unique facility identifier that designates the ATC unit or centre; a facility list enumerating all operational nodes within the same facility; and an associated sector list specifying the airspace sectors under the agent's control (e.g., *LFPG_DEL*, *LFPG_GND*). Additionally, it contains an adjacent ATC Agent list, detailing neighbouring agents responsible for adjacent airspace sectors, and an adjacent facility list, which describes external sub-networks or services managed by nearby agents used for inter-agent routing and delegation.

Based on this provisioning data, the ATC Agent builds a peer connection table, which maps direct physical and logical links to other agents and servers. This table captures essential information such as the type of transport connection (e.g., TCP, User Datagram Protocol (UDP), SCTP), current link state, and reconnection policies. This structure enables fast and reliable handshakes and message exchanges across agent boundaries.

In parallel, the ATC Agent also constructs a routing table optimised for multiple layers of communication. The routing logic supports sector-based routing, mapping each airspace sector to its controlling or adjacent agent; facility-based routing, which enables internal message forwarding to services or application servers; and fallback logic, which defines alternate paths for failover or load balancing. This routing intelligence ensures that client requests, context messages, and service traffic are reliably delivered to their intended destinations including when network condition changes due to node transitions or failures.

CM Agent Registration

Upon initialisation, the CM Agent sends a REGISTRATION_REQUEST to the ATM Server, declaring the facility it is responsible for managing. In response, the ATM Server returns a sector assignment list, which defines all airspace sectors delegated to that CM Agent based on its facility scope. Alongside this, the CM Agent receives a provisioning package that includes a unique facility identifier representing the ATC unit or centre, as well as the complete list of associated sectors under its jurisdiction (e.g., LFPG_DEL, LFPG_GND).

For each assigned sector, the CM Agent automatically instantiates a corresponding set of context management structures. These structures serve as state holders that track context ownership, associated metadata, and dynamic role assignments such as controlling, mirroring, and monitoring. They also maintain session-to-context bindings, allowing the agent to track live communication sessions, and include historical references to support traceability and continuity during handovers or failover scenarios.

Following provisioning, the CM Agent constructs its internal peer table, which maps both its logical and physical connectivity with the ATM Server and all associated user workstations. This mapping enables robust interaction, peer status tracking, and message routing coordination across the network.

In parallel, the CM Agent also initialises a context lifecycle registry. This registry enables the agent to monitor context status in real time, track ownership changes, and manage context transitions efficiently. It also supports the handling of role assignment requests from clients or other agents and plays a critical role in enabling failover, takeover, or context restoration during degraded operational conditions. These components enable the CM Agent to manage airspace contexts consistently, reliably, and with fault tolerance within its designated facility scope.

Client Registration Process

ATC clients, whether stationary or mobile, must register with the ATM Server before they can participate in any air-ground or ground-ground communication. Registration ensures that each client is properly identified, authorised, and provisioned with the necessary routing and agent information based on its declared sector role and domain affiliation.

When an ATC workstation or controller interface initialises, it sends a REGISTRATION_REQUEST to the ATM Server containing key data such as the targeted sector or role (e.g., *LTFM_GND*), the client type (stationary or mobile), and optionally, user credentials, HMI type, or workstation location. The domain portion of the context ID (e.g., ltfm.atc.tr) specifies the airport or control region being served. The ATM Server parses this information to identify the corresponding ATC Agent, verify that the declared sector exists and is available, and confirm that the user or organisation is authorised to operate within it.

Upon successful validation, the ATM Server marks the client as REGISTERED and returns a REGISTRATION_RESPONSE containing the ATC Agent's address (e.g., 10.0.2.15:5910). Additional configuration data (e.g., sector lists or related context information) may also be included to support operational readiness. Once registered, the ATC Agent manages the client's context data, facilitates real-time communication, and maintains session continuity as roles or network paths change.

Aircraft and other flight systems act as mobile clients, following a similar registration process. When initialising the ATMACA stack, the aircraft system (e.g., avionics unit or Electronic Flight Bag) sends a REGISTRATION_REQUEST containing essential flight metadata, including the callsign (e.g., *THY6AB*), aircraft registration (e.g., *TC-JHK*), departure and destination airports (e.g., *LTFM–EDDF*), and, optionally,

aircraft type, operator code, and scheduled departure time. The ATM Server validates this information, checks for unique callsigns within the active flight window, and determines the appropriate ATC Agent based on airspace mapping.

After validation, the server responds with the address of the designated ATC Agent (e.g., 10.0.5.4:5910), marking the aircraft as REGISTERED. The aircraft then initiates a LOGON procedure to the assigned ATC Agent to establish active communication sessions. From this point, the ATC Agent maintains session continuity, manages context updates, and supports mobility as the aircraft transitions across FIR boundaries or communication domains.

Application Server Registration

Upon initialisation, an Application Server sends a REGISTRATION_REQUEST to the ATM Server, identifying itself as a functional service node operating within a specific facility or domain. The request includes its host identity, realm, node role (i.e., APPLICATION_SERVER), and a list of supported capabilities. After validating the server's identity and confirming authorisation, the ATM Server responds with a provisioning package that includes the address of the ATC Agent the server should communicate with. This ATC Agent becomes the Application Server's point of integration for exchanging operational data and participating in context-related workflows.

Once registered, the Application Server establishes a connection to the assigned ATC Agent, updates its peer table with the agent's information, and transitions to the REGISTERED state. This status enables the server to begin supporting ATMACA services such as data sharing, advisory generation, or other domain-specific coordination functions. This lightweight and centralised registration process ensures secure onboarding, efficient routing, and controlled service integration of Application Servers into the ATMACA network.

While registration establishes the initial configuration for each node, it is equally important to ensure that this provisioning data remains current throughout the system's lifecycle. To support this, DLCM includes dedicated messages that allow the ATM Server to push updates or allow nodes to request provisioning refreshes when necessary (**Table 2.32**). These messages help maintain alignment between the ATM Server and participating agents or clients, ensuring consistency in sector assignments, peer awareness, and routing accuracy.

Table 2.32: Provisioning Synchronisation Messages

Message Name	Direction	Purpose	Key Parameters
PROVISION_UPDATE_REQUEST	ATM Server → Agent/Client	Pushes updated provisioning data, such as new sector assignments or peer changes.	UpdatedDataSet, UpdateType, EffectiveTime, ProvisionDataVersion
PROVISION_UPDATE_RESPONSE	Agent/Client → ATM Server	Acknowledges update receipt or reports issues.	Status, ErrorCode?, AcknowledgedVersion, NodeHost, NodeRealm
PROVISION_STATUS_REQUEST	Agent/Client → ATM Server	Requests current provisioning version or status without pulling full data.	NodeHost, NodeRealm, CurrentVersion, Scope
PROVISION_STATUS_RESPONSE	ATM Server → Agent/Client	Confirms version validity or indicates that an update is required.	ProvisionDataVersion, Status, IsUpdateRequired, UpdateHint (e.g., "sector-update")
PROVISION_PULL_REQUEST	Agent/Client → ATM Server	Requests full or scoped provisioning data for sync or recovery.	NodeHost, NodeRealm, QueryScope(e.g., sector, agent, all), LastKnownVersion?
PROVISION_PULL_RESPONSE	ATM Server → Agent/Client	Responds with provisioning data matching the requested scope.	ProvisioningData, ProvisionDataVersion, Scope, ResponseTimestamp

Operational connection messages (**Table 2.33**) are exchanged between ATMACA clients (e.g., FlightDeck and ATC Clients) as well as Application Servers and ATC Agents to establish and maintain active, context-aware communication links. These messages are used after the registration process is complete, when a node transitions into its operational state by joining a specific context (e.g., a flight or ATC sector) and assuming a role such as controlling, mirroring, or monitoring. Through logon, attachment, updates, and graceful disconnection, these messages ensure that nodes remain synchronised with the network, properly integrated into ongoing sessions, and capable of real-time service exchange. This set of procedures is essential to maintain operational continuity, support role transitions, and manage presence awareness throughout the lifecycle of an active ATMACA participant.

Table 2.33: Operational Connection Messages

Message Name	Direction	Purpose	Key Parameters
LOGON_REQUEST	Client/App → ATC Agent	Initiate operational presence	ContextID, Role, SessionToken
LOGON_RESPONSE	ATC Agent → Client/App	Confirm logon and session readiness	Status, SessionToken
UPDATE_REQUEST	ATC Agent → Client/App	Notify Agent Address	ContextData, NewIP, Capabilities
UPDATE_RESPONSE	Client/App → ATC Agent	Confirm update success	Status, Timestamp
CONTACT_REQUEST	ATC Agent → FD Client	Notify FD Client of next ATC Agent for handover	NextAgentHost, Realm, Context
CONTACT_RESPONSE	FD Client → ATC Agent	Confirm initiation of logon to next agent	Status, TargetAgentInfo
GROUND_FORWARDING_REQUEST	AT Client → Adj ATC Agent	Forward message (context/session related) to ground peer	OriginalContextID, TargetAgent, Payload
GROUND_FORWARDING_RESPONSE	ATC Agent → Adj ATC Agent	Acknowledge or reply from the forwarded message	Status, ResultData

The logon function enables clients to establish communication sessions with designated agents. Following the initial registration process, each client identifies its assigned agent's physical address and submits a logon request containing key context information. This context data, along with session details, is managed and stored by the agent to facilitate efficient communication and routing. The agent plays a major role in building and maintaining routing tables for effective packet forwarding, especially when destination nodes are hosted by other agents. Additionally, agents actively manage peer connections with both clients and adjacent agents to ensure seamless data exchange. Upon processing the logon request, a logon response is issued to the client, transitioning their status from registered to online. The ATMACA protocol's robust presence monitoring ensures network stability by promptly notifying relevant parties if a connection is lost or a node becomes unreachable. By combining efficient database management, dynamic routing logic, and proactive presence monitoring, the logon function ensures secure, reliable, and continuous communication in high-mobility aeronautical environments.

From a design perspective, the logon process relies on three key elements: efficient database management for storing context and session data, robust routing logic to enable dynamic peer discovery and message delivery, and proactive presence monitoring to ensure reliable status awareness for all participating nodes. These mechanisms enable ATMACA to provide stable, secure, and synchronised communication throughout the system.

The update function is used by ATC Agents to notify clients of operational changes that may affect their current connectivity, session alignment, or context participation. It supports real-time synchronisation between the agent and its assigned peers, ensuring that all connected nodes remain aware of updated

parameters, such as routing adjustments, role changes, or context metadata modifications, without requiring full re-registration or session termination.

This function builds on the registration and logon phases, acting as a continuity-preserving service that ensures clients remain aligned with the agent's operational state as mobility events, transitions, or configuration updates occur within the network.

The ground forwarding function enables the relay of messages between ATMACA nodes when a direct communication path is not available. Typically used when the destination node (such as a client or service) is not reachable through the local ATC Agent, this function allows the agent to forward the message to an adjacent agent based on routing information. Ground forwarding ensures that critical operational messages continue to flow seamlessly across distributed regions, even in the presence of mobility, handovers, or segmented infrastructure. It relies on the agent's forwarding table and routing logic to determine the optimal next hop, maintaining reliable communication across the ground network. This function supports continuity, resilience, and interoperability within ATMACA's multi-agent architecture.

The contact function in the ATMACA protocol ensures seamless communication continuity during mobility events, particularly when a mobile node such as a FlightDeck Client transitions between regions managed by different ATC Agents. When such a transition is detected, the currently serving ATC Agent initiates a contact procedure by providing the FlightDeck Client with the address of the adjacent ATC Agent responsible for the next region. This enables the client to proactively establish a new logon with the upcoming agent, maintaining uninterrupted connectivity and session awareness. The contact function is essential for supporting FIR boundary crossings, sector transitions, and dynamic reattachment scenarios, forming a critical part of ATMACA's mobility management strategy.

Binding messages (**Table 2.34**) are used to establish and manage the physical communication linkage between clients (FlightDeck or ATC) and their assigned ATC Agents. These messages operate at the network transport layer, enabling the system to accurately associate each client with its current IP address and transport parameters. The ATTACH_REQUEST message is sent by a client to notify the agent of its presence and communication path. Upon successful processing, the agent updates its peer table with the binding details, allowing subsequent messages to be routed correctly. This binding also enables message forking to multiple active endpoints such as mirroring or monitoring nodes by associating a single context or session with multiple IP addresses for redundancy or parallel visibility.
The DETACH_REQUEST message, conversely, is used to gracefully remove the client's binding, signalling disconnection and allowing the agent to release any related routing or session state. These messages provide a robust mechanism for maintaining accurate, synchronised communication across distributed operational environments.

Table 2.34: Binding Messages

Message Name	Direction	Purpose	Key Parameters
ATTACH_REQUEST	Client/App → ATC Agent	Announce IP and connection path	IPAddress, TransportInfo
ATTACH_RESPONSE	ATC Agent → Client/App	Confirm successful attach	Status, Timestamp
DETACH_REQUEST	Client/App → ATC Agent	Gracefully disconnect from current agent	Reason, Timestamp
DETACH_RESPONSE	ATC Agent → Client/App	Confirm disconnection	Status, CleanupStatus

The ATTACH_REQUEST message is used by a client to notify the assigned ATC Agent of its current IP address and transport parameters. Upon successful processing, this information is added to the agent's peer table, completing the binding process. This enables correct routing, message forking, and

presence awareness for operational communication. The client's connection status transitions to ONLINE following a successful attach, or upon receiving the associated context data from the ATC Agent, which confirms the client's operational readiness.

The DETACH_REQUEST message is used by the client to inform the agent that it is intentionally disconnecting. This unbinds the IP address and associated transport details, removing the peer entry and allowing the agent to release resources and avoid stale message delivery. Importantly, the client's context association and operational role remain intact (the detach operation only affects transport-level connectivity). The client is still considered part of the context, but temporarily unavailable for communication until a new ATTACH_REQUEST is issued.

2.4 Mobility Management

Mobility management is the fourth core capability of DLCM. It ensures continuity of service and communication as nodes or users change their point of attachment or role within the network. This capability, implemented by the mobility management module (Section 3.4), supports dynamic, distributed air traffic environments by managing transitions across physical, logical, and service contexts while maintaining session and data integrity throughout.

Mobility management supports four types of mobility (**Table 2.35**):

- User Mobility: Managed by the context management module. It enables users (e.g., controllers or pilots) to dynamically associate themselves with different contexts without losing operational state or role.
- Session Mobility: Handled by the Session Management module. It allows communication sessions to persist and transfer between different controllers or agents, ensuring uninterrupted service during handovers.
- Terminal (Device) Mobility: Supported by the connection management module. It tracks and maintains connectivity as clients (e.g., flight deck or ATC terminals) physically move between network zones or ATC Agents.
- Service Mobility: Enabled by the infrastructure's routing and registry systems. It ensures that services can follow the user, or in some cases, originate from mobile nodes (e.g., aircraft-based servers), by dynamically resolving and rerouting service endpoints as mobility events occur.

By modularising these mobility aspects across the stack, ATMACA provides resilient, scalable mobility support for a wide range of deployment scenarios. It ensures that critical services, context roles, and communication sessions remain intact and synchronised, even as users and nodes traverse different regions, facilities, or control domains.

Table 2.35: Types of Mobility and Their Management Responsibilities

Mobility Type	Managed By	Purpose
User Mobility	Context Management Module	Allows users to dynamically change contexts while retaining roles
Session Mobility	Session Management Module	Enables session continuity across control or sector handovers
Terminal Mobility	Connection Management Module	Maintains connectivity for mobile devices and clients
Service Mobility	Service Registry & Routing Logic	Ensures services are reachable regardless of user or server location

All four mobility types (user, session, terminal, and service) ultimately converge on a single architectural goal: ensuring service continuity. Regardless of whether a controller changes context, an aircraft shifts between ATC sectors, a terminal moves across network zones, or a service endpoint relocates, the ATMACA system is designed to preserve the operational flow. By coordinating context ownership, session routing, connectivity, and service availability, the mobility management framework guarantees that ongoing communication and functionality are never interrupted. This principle of uninterrupted service underpins the resilience and scalability of ATMACA's mobility-aware architecture.

ATMACA's mobility management relies on a set of well-defined protocol messages that ensure seamless continuity as users, sessions, terminals, and services move across the network. For most mobility types, the system builds on existing messages already defined under context, session, and connection management. These messages are simply reused to support mobility without adding protocol overhead. However, terminal mobility requires more active monitoring of network conditions, so it introduces a few new message types designed specifically for detecting disconnections and updating transport details in real time.

All these new messages such as MOBILITY_UPDATE_REQUEST, MOBILITY_UPDATE_RESPONSE, NODE_REACHABILITY_CHECK_REQUEST and NODE_REACHABILITY_CHECK_RESPONSE are defined and described in Section 2.4.3. To support terminal mobility scenarios, the ATMACA framework uses several connection management procedures, particularly those responsible for binding and re-binding client terminals to ATC agents. These binding-level operations rely on the ATTACH_REQUEST, DETACH_REQUEST, and their corresponding responses to manage the underlying transport state.

While these messages are defined and detailed within the connection management module, they are also reused in terminal mobility scenarios to dynamically update the client's connectivity as it moves across zones or reestablishes presence after a disconnection. Their reuse exemplifies the layered interaction between connection state and mobility continuity.

2.4.1 User Mobility

User mobility is enabled through the context management module. It allows users to bind themselves to a context and transition to another one without disrupting the service state. For example, a controller in an Area Control Centre (ACC) may switch to another sector, or a flight crew may assume control under a new FIR. This mobility is supported through context association, ownership updates, and validation mechanisms ensuring integrity and authorisation. The context management system tracks context ownership handles conflict resolution during transitions and synchronises with peer agents to reflect user handovers. This makes it possible to dynamically adjust user roles while maintaining consistent communication flows.

User mobility in the DLCM context management module supports dynamic and flexible participation of air traffic control operators across multiple workstations and roles. The system is designed to allow the same user to associate with the same context from different machines (enabling scenarios such as hot-standby terminals or dual-screen mirroring) while preserving session continuity and operational authority.

Additionally, multiple users can associate with the same context simultaneously, provided they assume compatible roles such as monitoring or mirroring without conflicting with the active controller. These capabilities are managed through strict role validation, context state tracking, and notification mechanisms achieved using ATTACH and DETACH messages, which dynamically update the ATC Agent's

routing table with valid IP endpoints for each context. When a user logs in from a secondary machine for the same context, an ATTACH message adds a second address to the ATC Agent's list, enabling mirroring behaviour. Similarly, when a different user logs into the same context in monitoring mode, their machine also sends an ATTACH, registering an additional read-only presence. This allows the ATC Agent to fork context messages to all attached endpoints, ensuring real-time visibility across mirrored and monitored workstations, while maintaining controlled access via validated roles.

User mobility involves both context lifecycle operations and dynamic reattachment (**Table 2.36**), and includes the following messaging flows:

Table 2.36: Context and Terminal Association Message Definitions

Message Name	Direction	Purpose
CONTEXT_CREATE_REQUEST	Client → CM Agent	Create a new context instance
CONTEXT_CREATE_RESPONSE	CM Agent → Client	Confirm creation, assign context ID
CONTEXT_ASSOCIATION_REQUEST	Client → CM Agent	Associate client with existing context
CONTEXT_ASSOCIATION_RESPONSE	CM Agent → Client	Confirm association and update routing
CONTEXT_DISASSOCIATION_REQUEST	Client → CM Agent	Detach client from the associated context
CONTEXT_DISASSOCIATION_RESPONSE	CM Agent → Client	Acknowledge disassociation
ATTACH_REQUEST	Client → ATC Agent	Notify physical connection details for terminal awareness
ATTACH_RESPONSE	ATC Agent → Client	Confirm and update routing to context address table
DETACH_REQUEST	Client → ATC Agent	Notify disconnection
DETACH_RESPONSE	ATC Agent → Client	Confirm routing removal

These messages work together to enable:

- Initial context creation or association via the CM Agent.
- Terminal-level address registration using ATC Agents (for routing and mirroring).
- Clean detachment during handovers, logouts, or mobility.

A detailed message sequence diagram for these flows is provided in Section 3.4.

2.4.2 Session Mobility

Session mobility is enabled through the session management module of the DLCM. This module maintains stateful session representations that can be reassigned from one context to another as needed. Each session is uniquely identified and linked to both local and remote entities such as an ATC Client and an airborne unit. When a handover is triggered, the session can be securely transferred between authorised contexts.

This is accomplished using structured message flows including SESSION_HANDOVER_REQUEST, SESSION_OWNER_CHANGE, and SESSION_TRANSFER_COMPLETED, routed via ATC Agents. These messages ensure that session metadata, ownership, and transfer status are accurately updated and reflected across all involved parties.

The module also logs lifecycle changes and verifies handover validity to prevent conflicts or unauthorised reassignments. This ensures that communication continuity and data integrity are preserved even during high-mobility transitions across airspace or organisational boundaries.

Table 2.37 presents the core message types used to support session mobility in the ATMACA architecture. These messages are categorised across session control, context coordination, and connection continuity to ensure that a session can be gracefully transferred between operational entities without

service disruption. Each message plays a distinct role in negotiating, authorising, and completing a session handover while keeping the context and peer associations synchronised across ATC Clients, ATC Agents, and CM Agents. Message sequence diagrams can be found in Section 3.4.

Table 2.37: Session Transfer and Binding Management Message Definitions

Message Name	Direction	Type	Purpose
SESSION_HANDOVER_REQUEST	ATC Client → ATC Agent	Session Control	Request to transfer an existing session to a new controller/context
SESSION_OWNER_CHANGE	ATC Agent → CM Agent	Session / Context	Notify CM Agent about the change in session ownership
SESSION_TRANSFER_COMPLETED	ATC Agent → ATC Client	Session Acknowledgment	Confirm to the new controller that session transfer is successful
UPDATE_REQUEST	ATC Agent → Client	Connection Mgmt	Update IP or routing details for the session or context
UPDATE_RESPONSE	Client → ATC Agent	Connection Mgmt	Acknowledge routing/IP update
ATTACH_REQUEST	Client → ATC Agent	Binding Mgmt	(If needed) Rebind client connection during handover
ATTACH_RESPONSE	ATC Agent → Client	Binding Mgmt	Confirm client reattachment
DETACH_REQUEST	Client → ATC Agent	Binding Mgmt	Optionally used if previous client unbinds from session
DETACH_RESPONSE	ATC Agent → Client	Binding Mgmt	Confirm detachment and session release

2.4.3 Terminal Mobility

Terminal mobility in the ATMACA architecture refers to maintaining seamless communication as a terminal (e.g., ATC workstation or flight deck client) moves across network zones, which may lead to changes in IP or transport-level identifiers. The DLCM connection management module ensures continuity by tracking and updating these identifiers, allowing the terminal to remain associated with its original session and context even after network transitions.

When a terminal detects a change in its network attachment (e.g., due to a Layer 2 or Layer 3 handover), it must initiate a MOBILITY_UPDATE_REQUEST to the ATC Agent. This message announces the new transport parameters while preserving session continuity. The ATC Agent replies with a MOBILITY_UPDATE_RESPONSE to confirm the update and maintain binding consistency.

In scenarios where the ATC Agent detects the loss of communication with a known client (e.g., context becomes unreachable), it triggers a NODE_REACHABILITY_CHECK_REQUEST to probe the client's availability. The client replies with a NODE_REACHABILITY_CHECK_RESPONSE, confirming its operational presence and optionally including updated transport information. This proactive probing mechanism allows the system to detect failures early and initiate recovery flows, preserving service continuity even when clients move unpredictably.

This combination of reactive (client-initiated) and proactive (agent-initiated) messaging ensures robust support for terminal mobility. It allows mobile clients to remain reachable, facilitates real-time context rebinding, and guarantees uninterrupted communication across dynamic networking conditions.

These new messages allow the ATMACA system to support terminal mobility both proactively (client-initiated updates) and reactively (agent-initiated checks), enhancing fault resilience and minimising service disruption during mobility events.

To support the diverse forms of mobility described across user, session, terminal, and service levels, the ATMACA architecture defines a dedicated set of mobility-aware protocol messages (**Table 2.38**). These messages orchestrate communication across agents, clients, and servers to ensure that context, session, connection, and service bindings are maintained or gracefully transferred during mobility events. The tables below categorise and summarise the key message exchanges involved in each type of mobility management. Message sequence diagrams can be found in Section **3.4**.

Table 2.38: Mobility Management Message Definitions

Message Name	Direction	Type	Purpose
MOBILITY_UPDATE_REQUEST	Client → ATC Agent	Mobility Mgmt	Notify the agent of newly established peer connection with updated IP
MOBILITY_UPDATE_RESPONSE	ATC Agent → Client	Mobility Mgmt	Confirm the update and adjust context/session routing accordingly
NODE_REACHABILITY_CHECK_REQUEST	ATC Agent → Client	Mobility Mgmt	Sent when the agent loses connectivity to a known context's endpoint; asks client to confirm presence or trigger recovery.
NODE_REACHABILITY_CHECK_RESPONSE	Client → ATC Agent	Mobility Mngmt	Sent by client to confirm availability and optionally provide updated transport state

2.4.4 Service Mobility

Application Servers can be deployed on fixed or mobile stations. This dual-mode approach to service mobility ensures that both ground-based and airborne services can move freely while maintaining availability, discoverability, and operational continuity.

In the first case, a fixed service node (e.g., a ground-based application server) may migrate from one host or data centre to another for reasons like failover, scaling, or maintenance. The system ensures continued accessibility by dynamically updating routing and notifying clients of the new service location.

In the second case, a service node may be mobile (e.g., an aircraft providing weather updates or flight coordination services to nearby aircraft). These mobile services periodically advertise their capabilities and availability through registration or announcement messages. ATC Agents or distributed registries track their position and service state, enabling peer discovery and routing. Clients use logical identifiers and session tokens to interact with mobile services, decoupling identity from physical location.

To support dynamic service delivery in distributed environments, service mobility in ATMACA ensures that services remain discoverable and accessible, even as their hosting nodes change location or availability. Whether the service node is fixed (e.g., ground-based Application Server) or mobile (e.g., aircraft-based service provider), ATC Agents maintain up-to-date service registries to route client requests to the most appropriate service node.

All service-related communication is strictly routed through the ATC Agent, acting as a broker between clients and Application Servers. This ensures centralised control, secure routing, and service abstraction, regardless of the service node's location or mobility state. Messages shown in **Table 2.39** form the communication chain enabling reliable service mobility and continuity. Message sequence diagrams can be found in Section 3.4.

Table 2.39: Service Interaction and Delivery Message Definitions

Name	Direction	Type	Purpose
SERVICE_REQUEST	Client → ATC Agent	Service Discovery	Client requests access to a specific service
SERVICE_PROCESSING	ATC Agent → Application Server	Service Control	Notify the service node to start processing the request
SERVICE_DELIVERY	Application Server → ATC Agent	Service Delivery	Deliver service content to the ATC Agent

SERVICE_DELIVERY	ATC Agent → Client	Service Delivery	Forward service response to the client
SERVICE_REJECT	Application Server → ATC Agent	Service Control	Reject service request with reason
SERVICE_REJECT	ATC Agent → Client	Service Control	Forward rejection notice to the client
SERVICE_ERROR	Application Server → ATC Agent	Service Control	Notify ATC Agent of internal error
SERVICE_ERROR	ATC Agent → Client	Service Control	Forward error report to the client
SERVICE_CANCEL	Client → ATC Agent	Service Control	Request cancellation of ongoing service
SERVICE_CANCEL	ATC Agent → Application Server	Service Control	Forward cancellation to the service provider
SERVICE_ABORT	ATC Agent → Application Server	Service Control	Forceful termination of a service session (e.g., failover)
UPDATE_REQUEST	ATC Agent → Client	Connection Mgmt	Notify client of new or rerouted service endpoint
UPDATE_RESPONSE	Client → ATC Agent	Connection Mgmt	Confirm update to newly assigned service provider

2.5 Presence Management

Presence management checks if peers are alive and sets the CONNECTED or DISCONNECTED state. Connection management maintains the transport binding and routes. ATMACA uses a watchdog-based presence monitoring mechanism, where each connected peer periodically exchanges Device-Watchdog-Request (DWR) and Device-Watchdog-Answer (DWA) messages with its assigned agent. These heartbeats detect connection drops or silent failures. If a peer fails to respond within a specified watchdog timeout, the agent updates the peer's state to DISCONNECTED and triggers internal failover or recovery mechanisms depending on the peer's role. Presence can trigger failover or cleanup, and connection management then performs attach, update, and detach actions.

The ATC Agent and the CM Agent serve distinct but complementary purposes in maintaining context continuity and operational resilience. This decoupled yet coordinated design enables robust, scalable presence management without requiring direct links between CM and ATC Agents. Instead, each agent reacts locally and informs its designated subsystems to maintain operational continuity and context consistency. Peer state acts as the common truth layer, interpreted differently by the ATC Agent (transport-level) and the CM Agent (logical role-level) to support autonomous failover, context resilience, and seamless recovery.

Presence is evaluated per peer but aggregated per context. As long as at least one peer in a context remains active, the context remains ONLINE. If all peers are unreachable, the context transitions to OFFLINE. In parallel, user role reassignment (such as promoting a mirroring peer to controlling) is coordinated by the CM Agent, which maintains context role state and monitors user presence across peer devices.

The ATC Agent is responsible for managing peer transport connections, routing sessions, and maintaining visibility into the context's structural composition and active session bindings. When a peer within a context becomes unreachable, such as due to a watchdog timeout or transport-level failure, the ATC Agent updates its internal peer table to reflect the peer's disconnected status. However, as long as other peers within the same context remain reachable, the context is still considered active, and no remote notification or session-level disruption occurs. The ATC Agent does not participate in user role management or peer promotion.

If, however, all peers associated with a context become unreachable, the context is marked as unavailable. In this case, the ATC Agent initiates a cleanup procedure for all active sessions associated with the now-defunct context. Acting as a proxy for the unreachable clients, the ATC Agent sends a SESSION TERMINATE REQUEST to the remote participants of each affected session, indicating that the session has ended unexpectedly due to CONTEXT_UNREACHABLE or PEER_DISCONNECTED. This ensures that the remote side is informed of the session's invalidation and can take appropriate action to clean up

and release session resources. This mechanism avoids orphaned sessions and maintains consistent session state across the distributed ATMACA environment.

In contrast, the CM Agent operates within each client node and is responsible for managing logical user roles within a context such as controlling, mirroring, or monitoring. It tracks the status of all peers within the context and evaluates whether a role transition is needed. If the controlling peer becomes unreachable, and the context still has active mirroring nodes, the CM Agent autonomously triggers a local role switch, promoting one of the eligible mirroring nodes to take over as the new controlling peer. This transition occurs internally within the context and requires no coordination with the ATC Agent or external systems.

When a peer becomes disconnected, regardless of whether it was operating in controlling, mirroring, or monitoring mode, the agent responsible for managing the context detects the disconnection through heartbeat failure or timeout mechanisms. Upon detection, the agent immediately sends a CONTEXT UPDATE NOTIFY REQ message to all other active peers in the same context to inform them about the removal of the disconnected node. If the disconnected peer held the controlling role, the response strategy depends on the available peers.

If there are Mirroring nodes still active, no takeover request is needed, as they belong to the same ATC user and are eligible to directly assume the Controlling role. In this case, the Mirroring node promotes itself to Controlling and sends a CONTEXT UPDATE REQUEST to the relevant CM Agents. Once the CM Agents validate and update the context state, they propagate the change to all other participants by issuing a CONTEXT UPDATE NOTIFY REQ, ensuring consistency across the distributed system.

Alternatively, if only monitoring nodes remain after the controlling peer disconnects, one of them submits a takeover request to its CM Agent. Since the original controlling node is unreachable, the CM Agent processes the request locally without attempting to forward it. Upon approval, the requesting monitoring node is promoted to controlling, and a CONTEXT UPDATE REQUEST is sent to the CM Agents. As before, the CM Agents then issue CONTEXT UPDATE NOTIFY REQ messages to inform all remaining peers of the updated role configuration. This coordinated update process ensures resilience, seamless role transitions, and synchronisation across all nodes in the ATMACA network.

2.6 Service Delivery Management

The service delivery management component in ATMACA orchestrates the complete lifecycle of service interactions between clients and distributed application servers, ensuring continuity, reliability, and scalability across both fixed and mobile environments. It operates under the supervision of the ATC Agent, which acts as an intermediary for all messaging, providing abstraction, session mediation, and dynamic routing.

Service delivery management provides a robust framework for handling diverse types of service interactions between clients and application servers (**Figure 2.2**). These interactions are entirely brokered by ATC Agents, which maintain a service registry table that maps registered services to specific application server instances. Application servers must first register with the ATM Server for authorisation and provisioning and then log on to an ATC Agent. Upon logon, the ATC Agent creates or updates its service registry (**Table 2.40**) with the server's available services, enabling efficient routing of service requests from clients.

Table 2.40: Service Registry Data Structure

Field Name	Type	Description
ServiceID	Integer	Unique ID for the service
ServiceName	String	Human-readable name of the service
ServiceMode	Enum (00/01)	Whether the service is ON DEMAND or CONTRACT
ProviderURI	String	URI of the Application Server providing the service
SourceAddress	String	Physical or logical address of the application server
ApplicationServerID	String	Internal ID or reference for the app server
Capabilities	JSON/String	Metadata about supported features or payload types
Status	Enum	ACTIVE / INACTIVE / ERROR
LastHeartbeat	Timestamp	Last time the app server was reachable
RegistrationTime	Timestamp	Time of logon or update
ATCZone	String	Zone, FIR, or sector the service is associated with (if relevant)

Three primary service delivery modes (**Table 2.41**) are supported: on-demand, contract-based (periodic), and continuous. These services can further be classified as transaction-based (short-lived) or session-based (persistent). Session-based services use the same session lifecycle mechanisms as other ATMACA communication, including session creation, transfer, and termination, and are tightly bound to context identities maintained across agents.

Table 2.41: Service Type and Operational Model Table

Service Mode	Operational Model	Use Case Example
On-Demand	Transaction-Based (single request-response)	Request for flight plan information
Contract-Based	Session-Based (periodic updates upon contract)	Weather updates every 10 minutes
Continuous	Session-Based (persistent, real-time data flow)	Continuous surveillance or telemetry streaming

All communication flows through the ATC Agent, which maps client requests to the appropriate application server using the service registry table. **Figure 2.2** illustrates the service request lifecycle, showing how clients initiate, cancel, or receive service results via the ATC Agent, which coordinates request processing and responses with the Application Server.

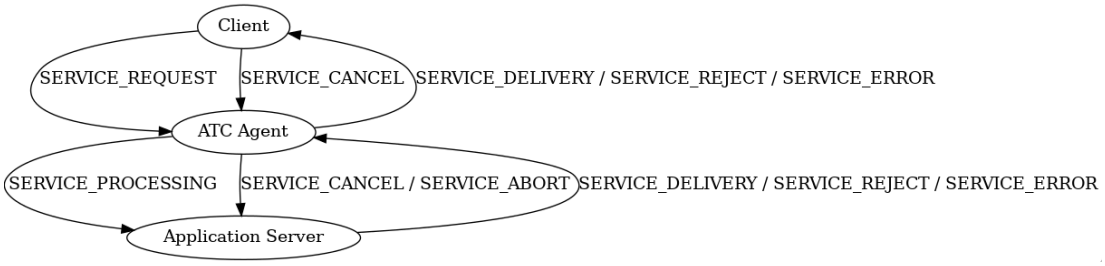

Figure 2.2: Service Interaction Flow Among Client, ATC Agent, and Application Server

To ensure operational scalability and fault tolerance, service mobility is tightly integrated into this framework. As aircraft or mobile clients change their ATC Agent affiliation (e.g., due to airspace transitions) the system ensures that they continue receiving services from the nearest registered application server. This is accomplished by updating routing tables and service associations in real time at the new ATC Agent, which either holds or dynamically fetches the latest service registry entries. The result is a seamless, location-transparent service experience that maintains quality and reduces latency. This delivery model ensures continuity, security, and proximity-aware routing of services, regardless of underlying mobility events.

3 SOFTWARE ARCHITECTURE

The ATMACA protocol stack is implemented through a modular and layered software architecture designed for high portability, runtime flexibility, and clean separation of concerns [8]. Each module in the system encapsulates a specific functional responsibility ranging from low-level transport and connection handling to protocol parsing, session management, context coordination, and application integration. These components interact through clearly defined interfaces, event-based messaging, and shared service contracts, enabling seamless integration across diverse deployment scenarios such as CM Agents, ATC Agents, and Application Servers.

The modular structure promotes separation of concerns, parallel development, and easier testing and deployment. Components interact through clearly defined interfaces and message buses, allowing flexibility in deployment, whether components are co-located (e.g., in a CM/ATC Agent) or distributed across separate agents.

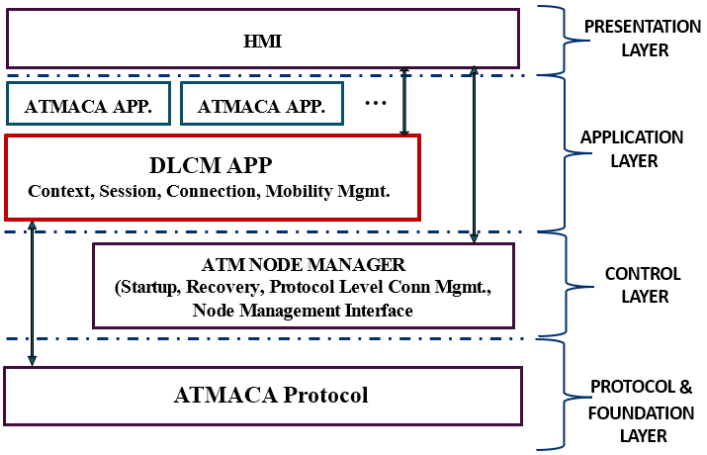

Figure 3.1: DLCM within ATMACA Software Architecture

Figure 3.1 presents a simplified ATMACA layered software architecture, illustrating the position of the DLCM application. The ATMACA protocol layers along with the ATM node manager (control layer), which orchestrates the system's lifecycle are used by the application layer. The node manager is responsible for initialising all application-level components, handling node startup and recovery, and exposing management interfaces for operational visibility. The application layer represents the system's topmost functional tier, responsible for delivering domain-specific services and runtime intelligence. At the centre of this layer is DLCM, which is initialised and managed by the control layer (the mode manager is detailed in D3.2 [8]).

DLCM's context management module keeps track of users' roles, system state, and operational assignments, helping coordinate services across different nodes. Its session management module is responsible for starting, maintaining, and ending communication sessions, ensuring that data flows smoothly even during transitions. The connection management module handles the network links, checking their health, recovering from failures, and keeping everything stable. Finally, the mobility management module ensures that communication continues without interruption when users or systems move across different devices, regions or networks. These modules enable ATMACA to deliver reliable, flexible, and context-aware communication in complex, changing environments. **Figure 3.2** illustrates the internal interactions between the DLCM modules, detailing how mobility, context, session, and connection management components coordinate to provide the mobility capabilities.

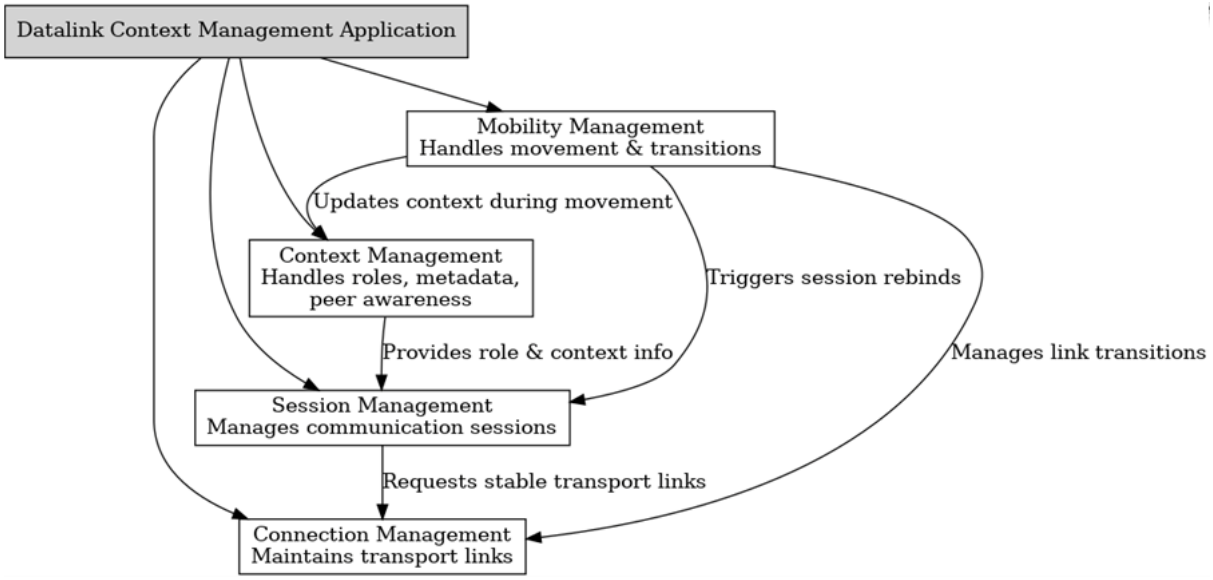

Figure 3.2: DLCM Module Interaction

3.1 Context Management Module

The context management module is responsible for establishing and maintaining operational context for each node in the network. Each context defines a communication scope for a mobile or stationary client (e.g., an aircraft or ATC workstation) and enables seamless session and role continuity across mobility events and network transitions.

As part of the node manager's application initialisation, the module ensures each client is accurately tracked with respect to its session, role, facility association, and sector presence. This module is particularly crucial for supporting mobility scenarios, such as handovers across sectors, facilities, and areas, enabling seamless communication and service continuity for airborne and stationary clients. In addition, the module is responsible for managing the lifecycle of these contexts across startup, failover, and dynamic reconfiguration events. The context management module

- Maintains a logical container for grouping active sessions tied to a context.
- Keeps track of the role assignments (e.g., controlling, monitoring, mirroring).
- Supports real-time updates and synchronisation of context metadata (e.g., session bindings, node location, IP address).
- Enables reassignment of context roles (controlling, mirroring, monitoring) via planned handover or failover-driven takeover between ATC clients within the same context.
- Provides interface endpoints for peer agents, ATM Servers, and applications to query or modify context state.

The context management module is typically run by CM Agents and participates in distributed synchronisation with other DLCM components to ensure fault tolerance and system-wide coherence. It supports both functional and runtime responsibilities related to client context, session lifecycle, and mobility operations.

The context management module includes several key components that work together to support accurate role assignment, operational awareness, and seamless coordination across the network. **Figure**

3.3 depicts the context manager components showing the context lifecycle from creation to registration, presence tracking, data synchronisation, and eventual role association via the role binding table.

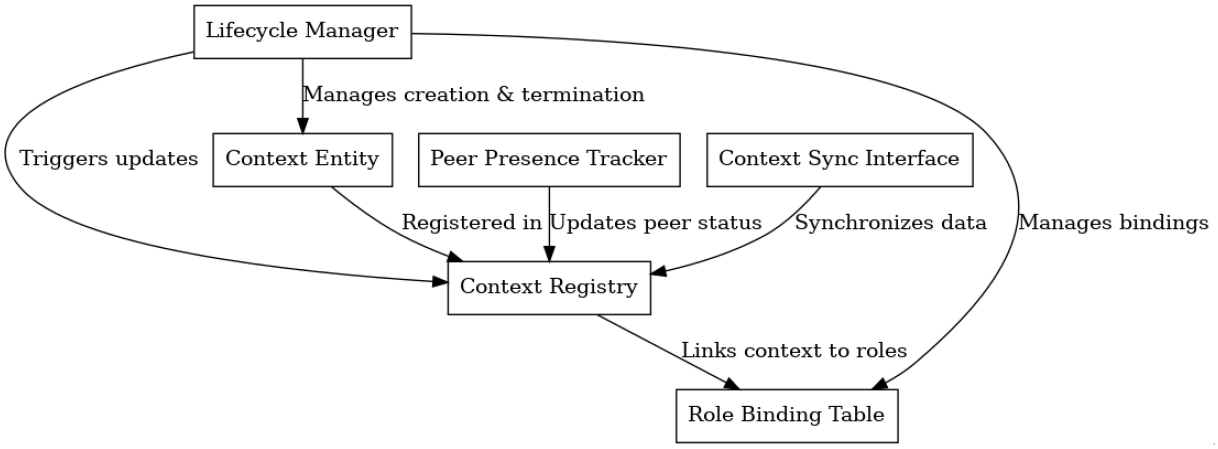

Figure 3.3: Context Management Architectural Elements diagram

The context entity represents a logical unit of communication, such as a user session or operational role, uniquely identified and tracked across the system. Context entities are registered in a context registry that maintains an authoritative list of active contexts, including associated metadata like node ID, realm, and assigned role. The Context Sync interface provides context updates, replication, and failover handling between distributed nodes. A role binding table is used to map each context entity to one or more operational roles (e.g., controlling, monitoring) while a peer presence tracker keeps real-time awareness of peer availability and status. The lifecycle manager covers the context creation, updates, termination, and cleanup processes, ensuring consistency throughout dynamic events.

The context management module (**Figure 3.4**) supports robust, dynamic, and standards-compliant handling of operational contexts across distributed air traffic control environments. It comprises five specialised functional subsystems, each addressing a critical aspect of context lifecycle and coordination. The context lifecycle manager is responsible for creating, updating, and retiring logical contexts, ensuring proper initialisation and teardown aligned with real-world operational scenarios. The role manager governs the assignment and transition of user roles while enforcing consistency, authority rules, and conflict resolution across shared contexts.

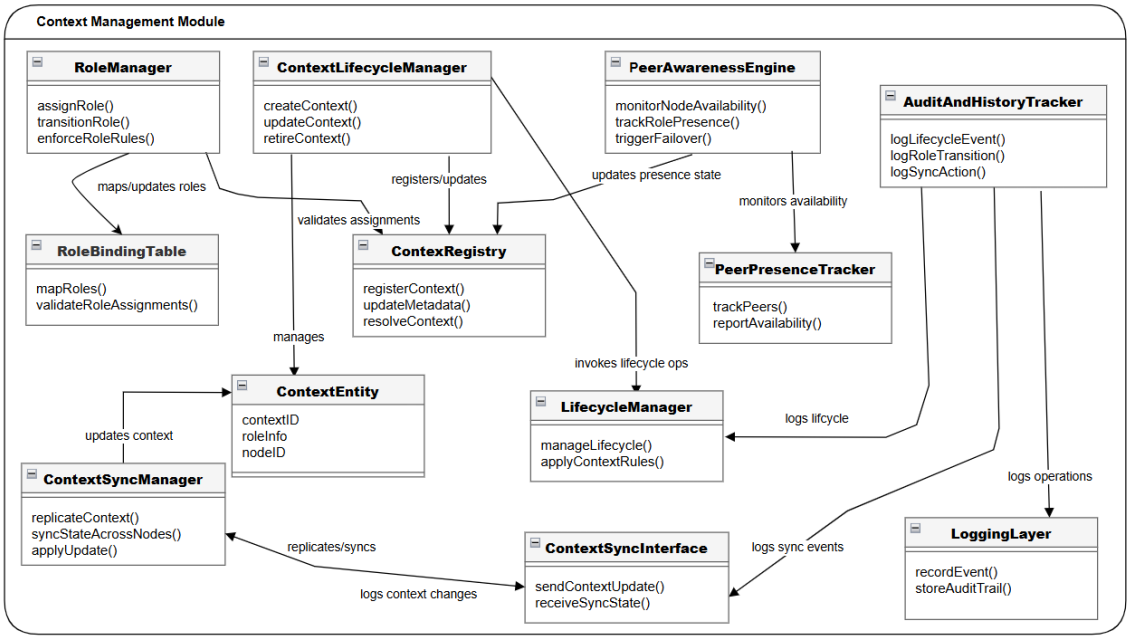

Figure 3.4: Context Management Module – Class-Level Functional Architecture

The Context Sync manager handles real-time state replication across CM Agents and ATC workstations, enabling consistent mirrored or monitored roles across distributed nodes. The peer awareness engine continuously tracks node availability, role presence, and health of each context, issuing alerts or triggering failover when necessary. Finally, the audit and history tracker maintains a tamper-proof operational log of all lifecycle events, role transitions, and synchronisation actions, supporting traceability, diagnostics, and regulatory compliance.

Table 3.1 summarises how each functional subsystem of the context management module aligns with its supporting architectural components to ensure coordinated context handling.

Table 3.1: Context Management Module – Updated Linkage Table

Functional Subsystems	Related Architectural Elements	Interaction Summary
Context Lifecycle Manager	Context Entity, Context Registry, Lifecycle Manager	Creates, updates, and retires logical contexts; registers lifecycle changes in the registry.
Role Manager	Role Binding Table, Context Registry	Governs assignment of roles and updates role mappings within the context registry.
Context Sync Manager	Context Sync Interface, Context Registry	Replicates context state across distributed nodes and synchronises operational updates.
Peer Awareness Engine	Peer Presence Tracker, Context Registry	Monitors the availability and status of peer nodes, updating the registry in real time.
Audit & History Tracker	(Implied Logging Layer), Context Registry, Lifecycle Manager, Context Sync Interface	Records lifecycle transitions, role updates, and sync actions for traceability, compliance, and recovery.

The context management workflow (**Figure 3.5**) governs the lifecycle of operational contexts within the DLCM subsystem, enabling consistent association, activation, monitoring, and synchronisation across distributed ATC systems. Each context represents a logical boundary tied to a specific sector or facility and acts as the central point for user association and session coordination. The workflow begins with the initialisation phase, where the CM Agent registers with the ATM Server and provisions its assigned sectors as inactive contexts. Upon user interaction, such as a controller associating with a context, a series of validation and registration steps are executed to activate the context and update its operational state.

Throughout its lifecycle, a context may support multiple users in different roles (e.g., controlling, mirroring, monitoring) and can be synchronised or transitioned as needed through failover, mirroring, or mobility events. The context management workflow ensures that context creation, ownership, status, and user bindings are accurately maintained and propagated across all DLCM components and participating agents in real-time.

The DLCM context lifecycle and user association flow ensures that air traffic control users can dynamically associate with operational sectors in a controlled, state-aware manner. When a CM Agent starts, it registers with the ATM Server to obtain a list of sectors for its assigned facility. For each sector, a corresponding context is created locally with an initial status of INACTIVE. When a user requests to associate with a context, the system checks its current state. If the context is already ACTIVE, the association is immediately accepted, and an ATTACH message is sent to the ATC Agent to notify it of the new participant. If the context is still INACTIVE, a multi-step process is triggered where the context is registered with the ATM Server, the user logs on to the ATC Agent, and the user then sends an UpdateContext command to activate it. Upon successful activation, the context status is set to ACTIVE, audit records are logged, and the ATTACH message is sent to the ATC Agent. This layered and fault-aware flow enforces proper lifecycle management, prevents premature associations, and maintains real-time awareness across all agents in the network.

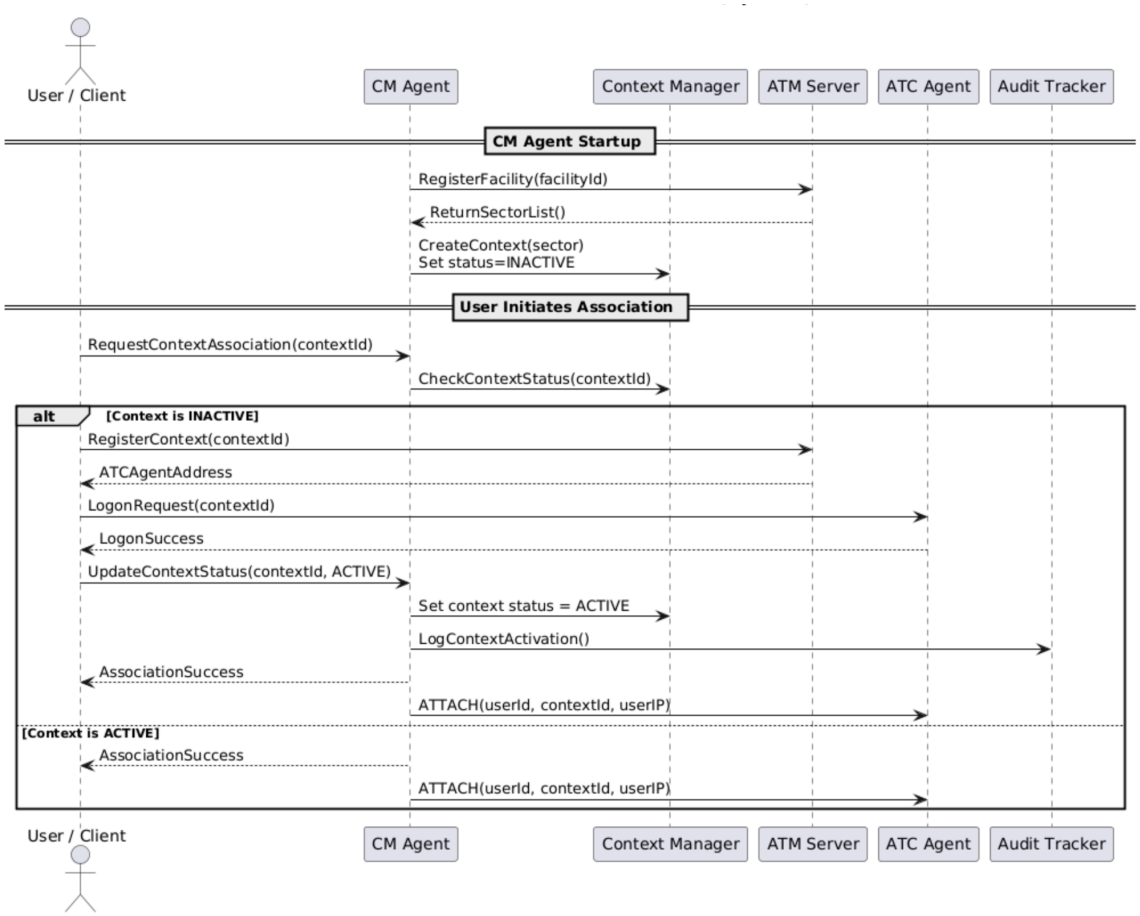

Figure 3.5: Context Initialisation and Association

The context synchronisation workflow (**Figure 3.6**) ensures consistent and real-time propagation of context metadata across CM Agents and ATC clients. It allows clients to upload new or updated context configurations using the CONTEXT_PUSH_REQUEST, which upon success associates the client with the

context. Clients may also initiate metadata synchronisation by issuing a CONTEXT_PULL_REQUEST to retrieve the latest context definition from the CM Agent. When partial updates to a context are needed, the CONTEXT_UPDATE_REQUEST is used, followed by a corresponding CONTEXT_UPDATE_RESPONSE indicating the outcome. The CM Agent can proactively propagate updates to other clients via the CONTEXT_UPDATE_NOTIFY_REQUEST, ensuring state alignment across mirrored and monitored contexts. Additionally, the CM Agent may invoke CONTEXT_RETRIEVAL_REQUEST to pull full context data directly from a client node. This set of interactions enables dynamic, distributed, and fail-safe management of context state.

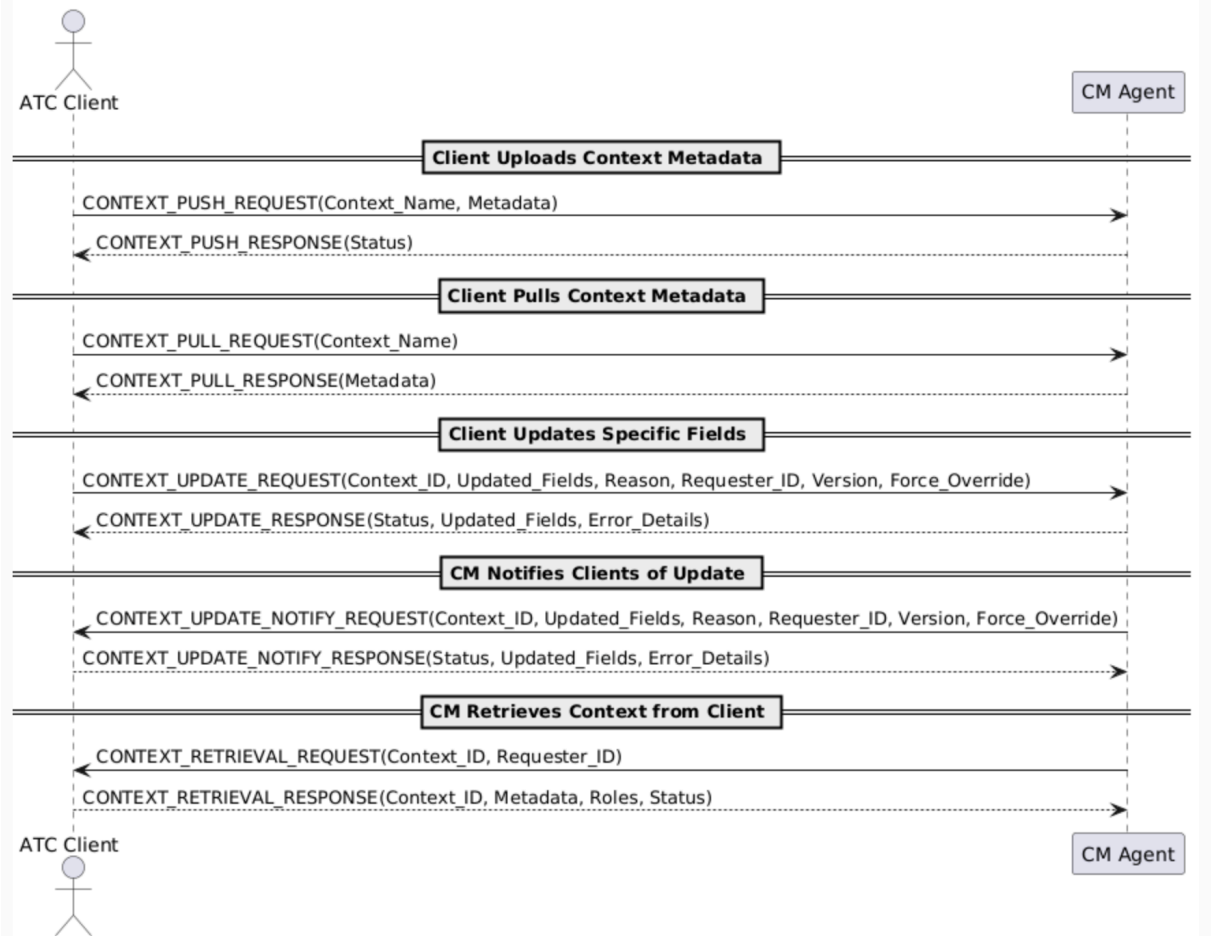

Figure 3.6: Context Consistency and Synchronisation Workflow

The context handover and takeover workflow (**Figure 3.7**) facilitate seamless transitions of the controlling role between ATC clients, supporting both planned delegation and unplanned failover scenarios. A planned handover is triggered by the active controlling client using the CONTEXT_HANDOVER_REQUEST, designating a mirroring or monitoring node to assume control. Upon successful validation, the CM Agent responds with a CONTEXT_HANDOVER_RESPONSE and notifies relevant participants through CONTEXT_ROLE_CHANGE_NOTIFY_REQUEST, ensuring that all nodes in the context are aware of the new controlling authority.

In unplanned scenarios such as controller workstation failure or loss of connectivity, a mirroring or monitoring client can initiate a takeover via CONTEXT_TAKEOVER_REQUEST. If the CM Agent detects that no active controlling node is present, it promotes the requesting client to the controlling role and confirms this with CONTEXT_TAKEOVER_RESPONSE, followed by role change notifications to synchronise all nodes.

This workflow also inherently supports user mobility, allowing controllers to switch workstations or relocate within a facility without disrupting operational context. Through coordinated role reassignment and real-time notifications, users can be elevated to or demoted from the controlling role as they move across devices, enabling features like mirrored terminals, hot-standby consoles, and dynamic role shifting. These capabilities enhance fault tolerance and operational flexibility, ensuring that ATC services remain continuous, distributed, and resilient to change.

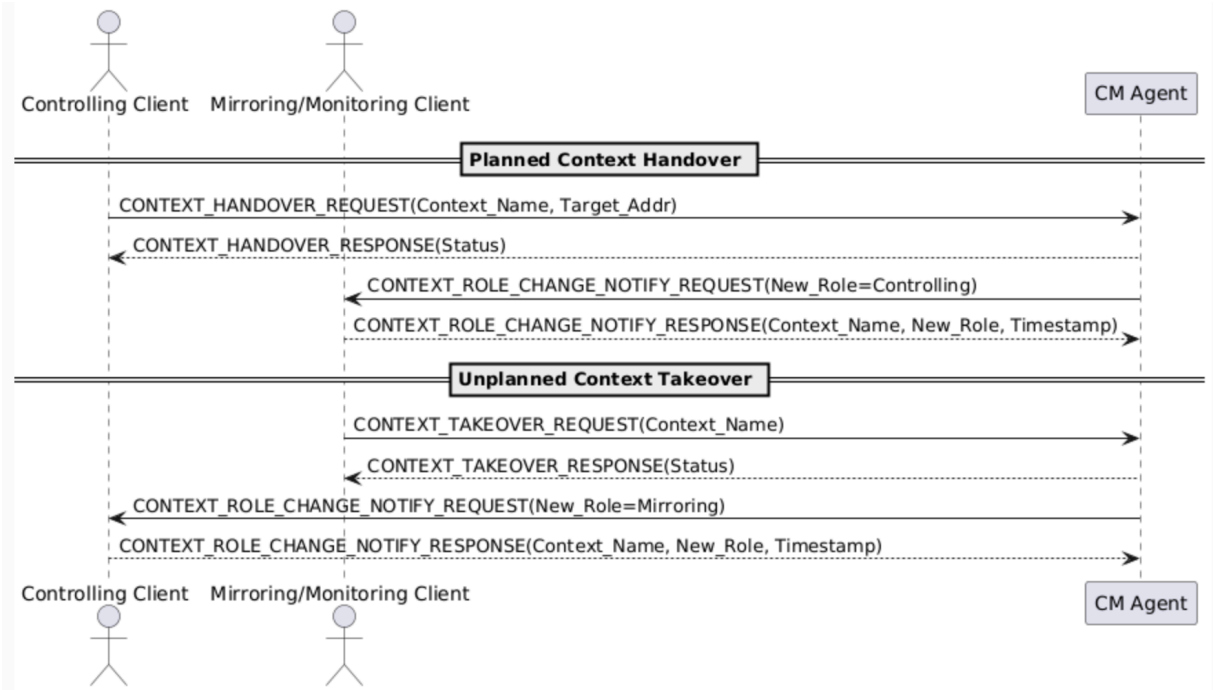

Figure 3.7: Context Handover and Takeover Workflow

The context termination and disassociation workflow (**Figure 3.8**) enables ATC clients to gracefully exit from a context, either as part of a shift handover, mobility event, or system shutdown. When a client no longer needs to participate in an active context, it sends a CONTEXT_DISASSOCIATE_REQUEST to the CM Agent, specifying the target context and its own IP address. Upon processing the request, the CM Agent responds with a CONTEXT_DISASSOCIATE_RESPONSE, confirming successful removal. If the disassociated client was acting in a controlling role, the CM Agent may trigger internal reassignment logic to elevate a mirroring or monitoring node to the controlling role, ensuring that the context remains operational. This workflow supports both user-driven exits and automation-driven disassociations, enabling mobility, workstation transitions, and safe context teardown procedures across the distributed ATMACA architecture.

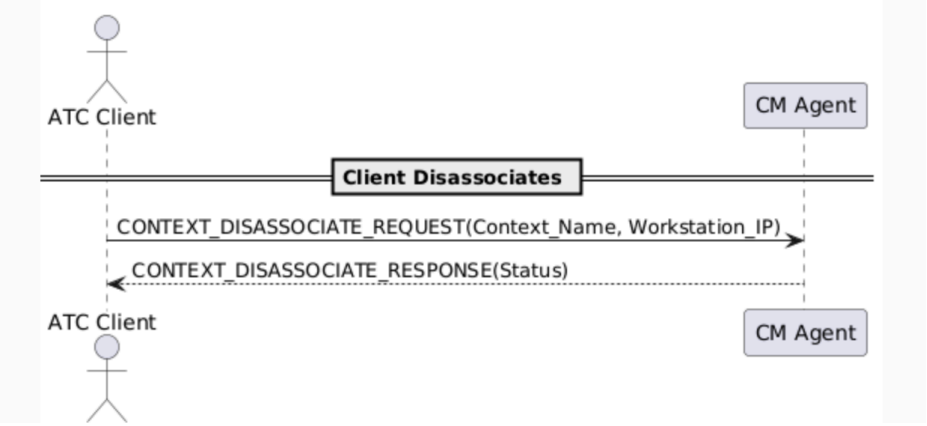

Figure 3.8: Context Termination and Disassociation Workflow

3.2 Session Management Module

The session management module is responsible for establishing, tracking, transferring, and terminating sessions between operational entities, such as ATC controllers and aircraft, across contexts. Sessions are dynamically managed and can be handed over from one context to another to support operational continuity across sector changes, controller shifts, or facility handovers.

The session management module works in coordination with the context management module and ATC Agents. While session-related messages originate from client workstations (e.g., ATC controllers), they are routed through ATC Agents, which act as intermediaries for reliable, secure session communication, especially between clients connected to different CM Agents. This decoupled communication model ensures that client-to-client session operations, such as session transfers, are possible even without direct physical connections. The session management module

- Maintains a registry of all active sessions and their current context ownership.
- Supports full session lifecycle: creation, binding, update, handover, and termination.
- Enables real-time session handover between contexts for sector or controller transitions.
- Provides validation and conflict checks during session handovers.
- Routes all session handover and transfer requests via ATC Agents for consistent access across distributed nodes.
- Tracks remote and local context associations tied to a session (e.g., controller and aircraft).
- Updates context ownership metadata and notifies relevant system components (e.g., CM Agent) upon changes.
- Logs session lifecycle events to ensure traceability and fault recovery.

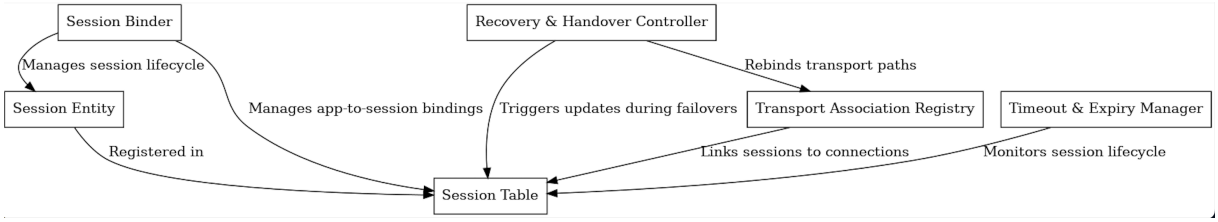

Figure 3.9: Session Management Architectural Elements diagram

The session management module supports logical, context-bound session tracking between controllers, aircraft, and remote nodes. It enables flexible session handovers using a structured message flow routed via ATC Agents, ensuring consistent connectivity even in a distributed, multi-facility ATMACA network. To maintain robust, persistent communication across dynamic network environments, the session management architecture (**Figure 3.9)** includes a set of core components that work together to manage session lifecycle, association, and recovery:

- Session Entity: Represents a unique, persistent communication instance between endpoints, identified by a session ID and used throughout the session lifecycle.
- Session Table: Serves as the active repository for tracking all live sessions, maintaining metadata such as session status, associated nodes, and timestamps.
- Session Binder: Handles the creation, lookup, and binding of application logic to session entities, ensuring that application flows are correctly tied to session context.
- Transport Association Registry: Maps sessions to one or more underlying transport connections, allowing the protocol to manage multiplexing and recover connections during failovers.
- Timeout & Expiry Manager: Monitors session activity and enforces timeout rules, handling expiration, cleanup, and idle session termination to maintain system efficiency.
- Recovery & Handover Controller: Manages the recovery of sessions during failovers or mobility events by re-binding session state and restoring communication continuity.

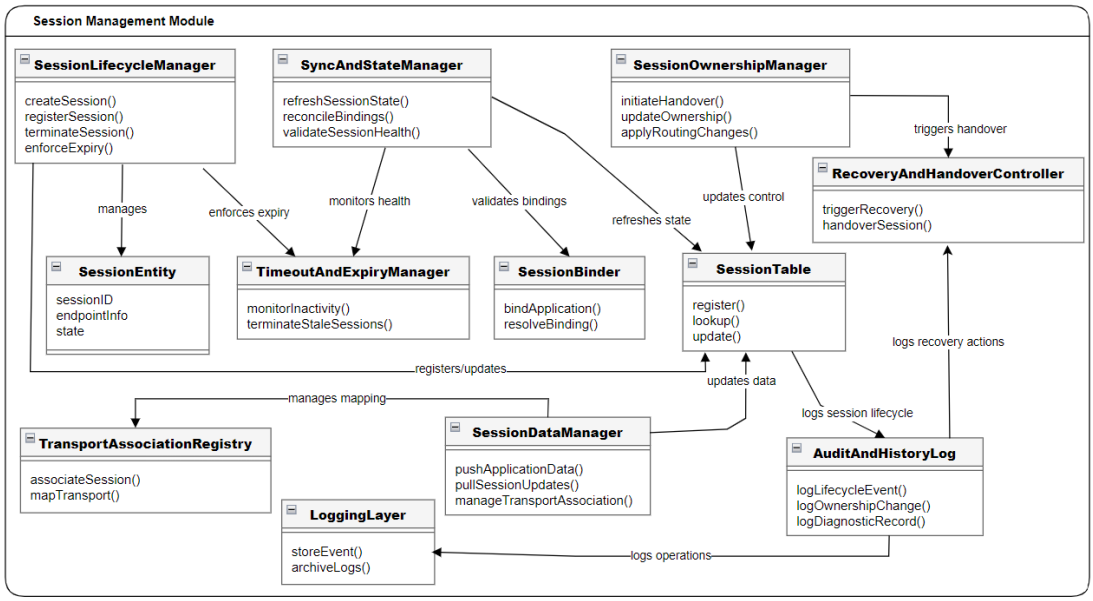

Figure 3.10: Session Management Module – Class-Level Functional Architecture

The session management module (**Figure 3.10**) handles the complete lifecycle, ownership, and data synchronisation of sessions across ATC clients and operational contexts. This functional architecture is

composed of modular managers that work in coordination to support session creation, handover, synchronisation, and termination, while ensuring consistent state and traceability. The session lifecycle manager oversees the initiation and graceful closure of sessions, while the session ownership manager enables ownership changes through handover or takeover operations. The SyncAndState manager handles session state reconciliation, periodic refreshes, and validation requests. Application-level data exchange and update flows are managed by the session data manager, supporting both push and pull mechanisms. All major operations are logged by the AuditAndHistory log to ensure traceability, fault analysis, and compliance. These components interact with both the CM Agent and remote ATC clients, forming a distributed yet coordinated system capable of managing complex multi-session air traffic control scenarios.

Table 3.2 outlines the connections between the session management subsystems and their underlying architecture, highlighting the flow of session control, ownership, and data synchronisation.

Table 3.2: Session Management Module – Updated Linkage Table

Functional Subsystem	Related Architectural Elements	Interaction Summary
Session Lifecycle Manager	Session Entity, Session Table, Timeout & Expiry Manager	Manages the full lifecycle of sessions including creation, registration, and termination, while enforcing expiry rules.
Session Ownership Manager	Session Table, Recovery & Handover Controller	Enables session handovers and ownership transitions by updating control and routing metadata.
SyncAndState Manager	Session Table, Session Binder, Timeout & Expiry Manager	Handles reconciliation, periodic state refresh, and validation of session integrity and health.
Session Data Manager	Transport Association Registry, Session Table	Facilitates push/pull exchange of application-level session data and manages transport mapping.
AuditAndHistory Log	Logging Layer (Implied), Session Table, Recovery & Handover Controller	Logs lifecycle events, ownership transfers, and system actions for compliance, traceability, and diagnostics.

The following outlines the key workflows and message exchanges that govern the lifecycle of sessions within the DLCM framework. Sessions represent stateful communication links between operational contexts, such as those owned by controllers and aircraft, and are tightly integrated with both context ownership and mobility operations. Session management workflows define how sessions are created, bound to contexts, transferred between users or sectors, and eventually terminated. These workflows ensure continuity, authority validation, and system-wide synchronisation in scenarios such as sector handovers, controller transitions, and dynamic network reconfigurations. All session-related messages are exchanged between ATC Clients (e.g., controller terminals) and are routed via the ATC Agent, which serves as a communication bridge, especially when clients are associated with different CM Agents. By standardising these interactions through well-defined protocol messages, such as SESSION_HANDOVER_REQUEST, SESSION_OWNER_CHANGE_REQUEST, and SESSION_TRANSFER_COMPLETED, the system enables resilient and flexible session coordination in a distributed air traffic management environment.

The session creation workflow (**Figure 3.11**) establishes the initial communication context between two participating ATC entities, typically a controller workstation and a remote peer such as another controller or an aircraft. This process begins when the initiating ATC Client sends a SESSION_CREATE_REQUEST to its locally connected CM Agent. This message includes the context ID, remote peer ID, application and flight identifiers, requested role, and security credentials. Upon validation, the CM Agent returns a SESSION_CREATE_RESPONSE, confirming the creation of the session and providing necessary metadata such as the assigned session ID, routing information, and heartbeat configuration. Following this, the initiating client informs the remote peer of the new session by sending a SESSION_START_REQUEST, which is acknowledged with a SESSION_START_RESPONSE. This sequence of messages ensures that both

endpoints have synchronised session context and are prepared for subsequent data exchange. The session is then registered locally, bound to the appropriate context, and marked as active within the initiating node's session manager.

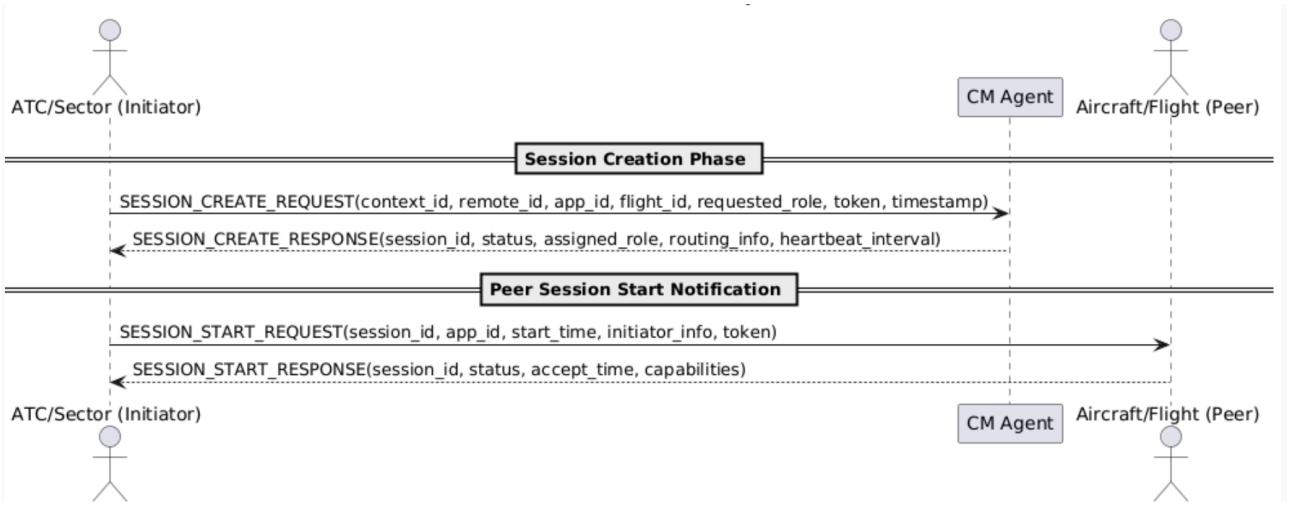

Figure 3.11: Session Creation Workflow (Protocol-Compliant, with ATC Controllers)

The session synchronisation and state management workflow (**Figure 3.12**) enables ATC clients and CM Agents to maintain a consistent and up-to-date view of session state across distributed nodes. Synchronisation is critical in ensuring operational continuity, especially in scenarios involving user mobility, mirroring, context forking, or failover recovery. This workflow is initiated by the client via the SESSION_SYNC_REQUEST, which can request either a full or delta update depending on the current known state. The CM Agent responds with a SESSION_SYNC_RESPONSE containing authoritative session metadata, ownership, participants, and application-specific payloads. Clients may further retrieve detailed application-level content using SESSION_DATA_PULL_REQUEST or submit updates via SESSION_DATA_PUSH_REQUEST. The CM Agent also supports in-band updates to session parameters through SESSION_PARAMETER_UPDATE_REQUEST, enabling clients to dynamically adjust behaviour or reconfigure session attributes without restarting the session. These message exchanges provide robust and scalable session state alignment across nodes, supporting real-time, collaborative, and fault-resilient ATC operations.

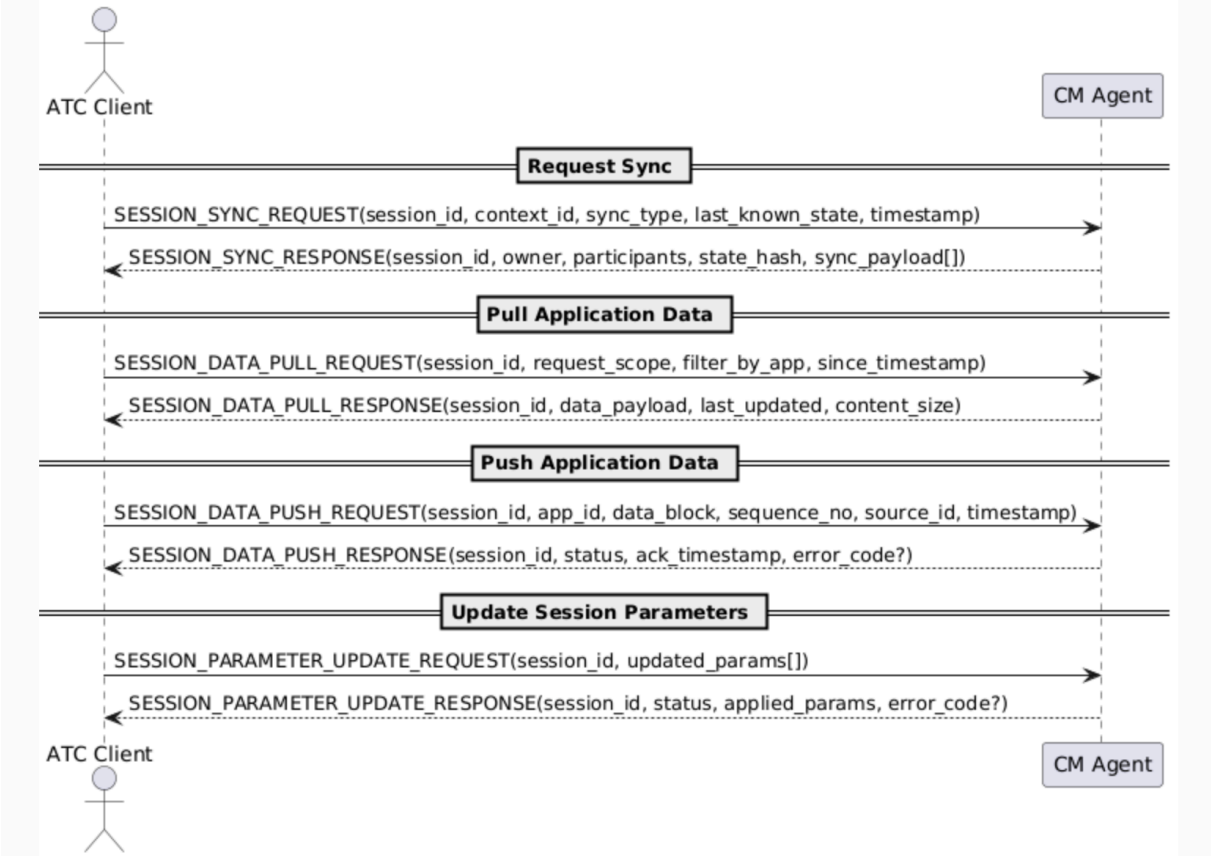

Figure 3.12: Session Synchronisation Workflow

The session handover and takeover workflow (**Figure 3.13**) enables controlled transfer of session ownership between ATC contexts to support operational transitions, controller handoffs, and fault recovery scenarios. A session handover is initiated by the current session owner via the SESSION_HANDOVER_REQUEST, targeting another active context. Upon receiving and accepting the request, the target context responds with a SESSION_HANDOVER_RESPONSE. After both parties complete the transition steps, the initiating context sends a SESSION_TRANSFER_COMPLETED_REQUEST, which is acknowledged by the new owner through a SESSION_TRANSFER_COMPLETED_RESPONSE. In cases where the original session owner becomes unresponsive or inactive, the system supports a fault-tolerant mechanism through the SESSION_TAKEOVER_REQUEST and SESSION_TAKEOVER_RESPONSE message pair, allowing a new context to assume ownership. Following a successful handover or takeover, the new session owner notifies the CM Agent using SESSION_OWNER_CHANGE_REQUEST, which in turn distributes ownership updates to participants via SESSION_OWNER_CHANGE_NOTIFY_REQUEST. This robust exchange ensures that all session participants and system components maintain a consistent understanding of session control and ownership in real time.

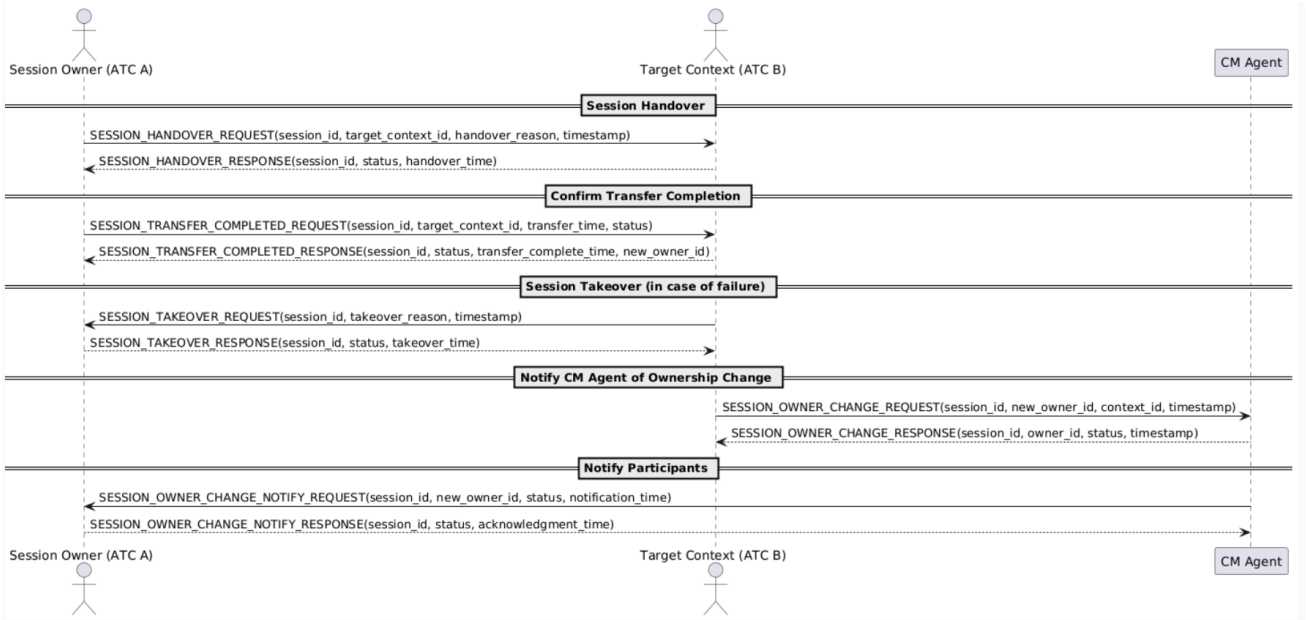

Figure 3.13: Session Handover and Takeover Workflow

The session termination workflow (**Figure 3.14**) defines the formal closure process for an active session, whether initiated voluntarily by clients or as a result of administrative or error-driven termination. In standard operational scenarios, an ATC client concludes a session by issuing a SESSION_END_REQUEST to its remote peer, signalling that all mission-related communication is complete. The peer responds with a SESSION_END_RESPONSE, confirming the session's successful closure and logging the completion status. Alternatively, the termination process may be initiated by the CM Agent, typically in response to timeouts, protocol violations, or system events via a SESSION_TERMINATE_REQUEST message directed at the owning client. The client must then confirm the termination using a SESSION_TERMINATE_RESPONSE. This dual-path approach ensures that sessions are gracefully decommissioned, state is synchronised, and resource cleanup is handled uniformly across the distributed architecture.

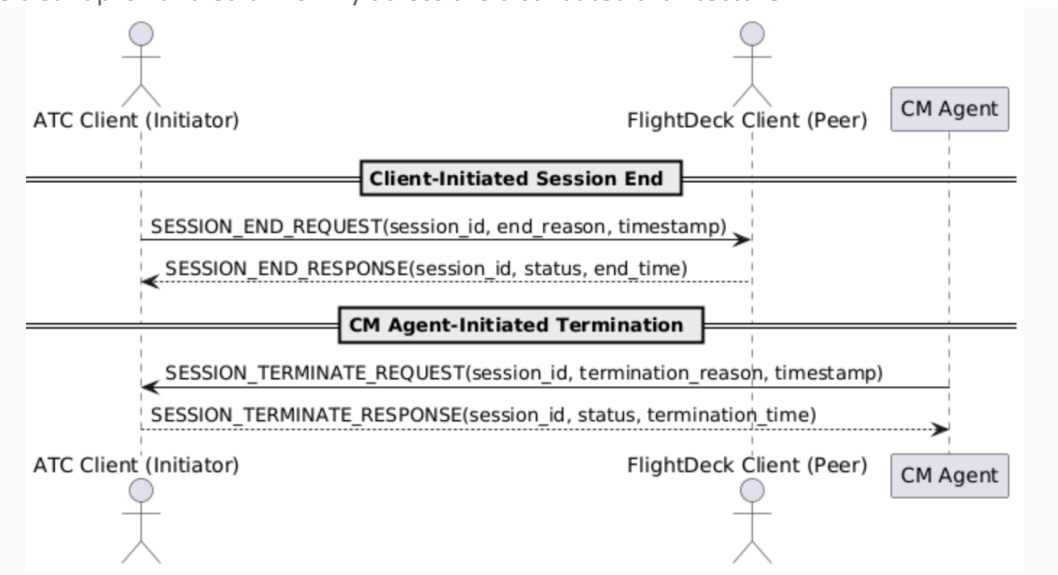

Figure 3.14: Session Termination Workflow

3.3 Connection Management Module

The connection management module is responsible for managing the operational state and lifecycle of communication links between mobile clients (such as flight deck applications) and ground-based agents (such as CM and ATC Agents). Each connection represents a logical attachment point, session capability, and service engagement instance between a mobile endpoint and a ground node.

This module forms the dynamic access and control layer enabling flight deck clients to engage with ATC services in a distributed, multi-agent network. It allows ground systems to authorise and monitor active clients, handle peer transitions (e.g., inter-agent handovers), and initiate control sequences required to maintain operational presence. Connections are message-driven and can be gracefully initiated, updated, or terminated using standardised ATMACA protocol messages.

The module provides mechanisms to coordinate multiple connection lifecycles in real time, ensuring unique bindings per role and resolving overlaps or conflicts. It uses ATC Agents as primary relays for all control-plane messages, ensuring clean demarcation between mobility-sensitive flight deck endpoints and fixed infrastructure such as CM Agents or the ATM Server. The context management module

- Manages operational communication links on top protocol level peer connection between mobile clients and ground agents.
- Supports DLIC lifecycle messages for initiating and terminating operational presence.
- Ensures unique active bindings per role (e.g., controlling/monitoring).
- Coordinates inter-agent handovers and reassignment logic.
- Validates connection parameters (e.g., context ID, IP address).
- Relays runtime changes such as updates to IP, host, or realm using UPDATE messages.
- Manages forwarding requests between ATC/CM Agents for mobility handling.
- Supports contact routing for informing clients of upcoming agent transitions.

The connection management module operates as a state-driven component within the DLCM layer, handling all runtime logic associated with connection establishment, maintenance, and lifecycle transitions. It separates responsibilities into several subsystem managers aligned with the message categories used in the ATMACA protocol. Each manager operates independently but interacts via a shared binding registry and message dispatcher.

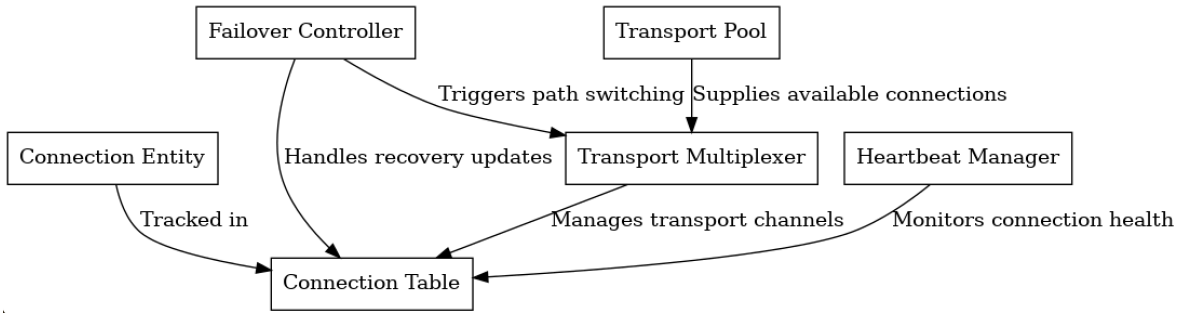

Figure 3.15: Connection Management Architectural Elements diagram

The connection management module ensures that reliable, low-level communication paths remain active and responsive across dynamic network environments. Its architecture (**Figure 3.15**) is composed of

several interrelated components that maintain transport layer continuity and enable seamless connection recovery:

- Connection Entity: Represents a single transport-level connection between nodes, carrying metadata such as protocol type, address, and connection state.
- Connection Table: Acts as the central registry for all active connections, tracking their state, associated nodes, and connection parameters in real time.
- Transport Multiplexer: Manages multiple parallel transport channels, enabling load balancing, prioritisation, and multi-homing across network interfaces.
- Heartbeat Manager: Periodically sends heartbeat messages and monitors acknowledgments to detect connection health and trigger alerts on failure.
- Failover Controller: Coordinates recovery actions in the event of link failure, including re-routing, session re-binding, or switching to alternative transport paths.
- Transport Pool: Maintains a catalogue of available transport resources (e.g., TCP, UDP, SCTP), allowing the system to establish new connections dynamically as needed.

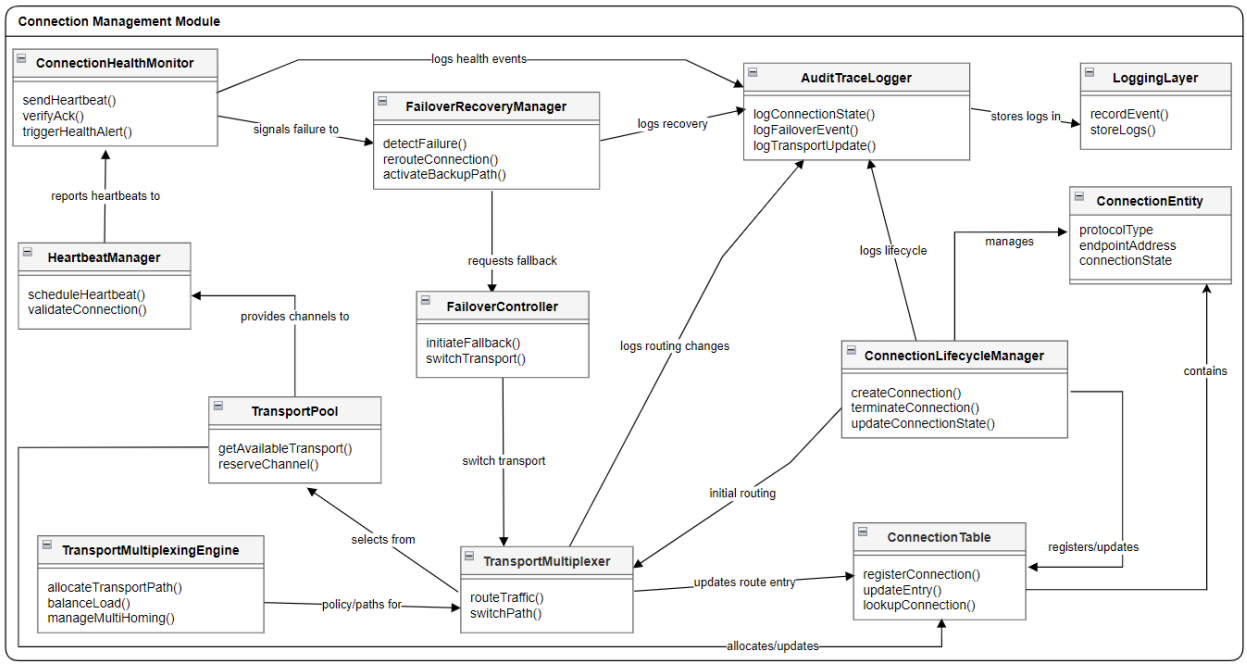

Figure 3.16: Connection Management Module – Class-Level Functional Architecture

The connection management module (**Figure 3.16**) ensures stable and reliable transport layer connectivity across dynamic and degraded network environments. It is composed of five specialised functional subsystems that coordinate the full lifecycle, health monitoring, multiplexing, failover handling, and traceability of communication links. The connection lifecycle manager oversees the creation, maintenance, and retirement of connection entities, maintaining accurate records in the connection table. The transport multiplexing engine manages multiple transport channels across available interfaces, distributing data flow and balancing performance. The connection health monitor uses heartbeat mechanisms to assess the status and liveliness of each connection, issuing alerts upon failures. When disruptions occur, the failover and recovery manager initiates fallback procedures and reroutes data paths using the transport multiplexer and failover controller. Finally, the audit and trace logger captures all critical events (e.g., link state changes, recovery actions, and transitions) into a dedicated logging layer,

supporting diagnostics, compliance, and operational transparency. These components enable resilient, service-aware connection management within ATMACA's distributed architecture.

The relationships detailed in **Table 3.3** show how the functional components of the connection management module map to architectural elements responsible for transport stability and recovery.

Table 3.3: Connection Management Module – Linkage Table

Functional Subsystem	Related Architectural Elements	Interaction Summary
Connection Lifecycle Manager	Connection Entity, Connection Table	Manages the creation, maintenance, and termination of transport layer connections.
Transport Multiplexing Engine	Transport Multiplexer, Transport Pool, Connection Table	Handles channel multiplexing, resource allocation, and connection-state binding.
Connection Health Monitor	Heartbeat Manager, Connection Table	Periodically checks connection status, detects failures, and updates system state accordingly.
Failover & Recovery Manager	Failover Controller, Transport Multiplexer, Connection Table	Initiates fallback logic, reroutes paths, and updates connection states during disruptions.
Audit & Trace Logger	Logging Layer (Implied), Connection Table, Failover Controller	Logs operational events, connection updates, and failover history for diagnostics and compliance.

To maintain stable and resilient communication across dynamic network conditions, ATMACA's connection management workflow governs how transport layer connections are initiated, maintained, and recovered. The following subsections describe the sequence of operations that ensure continuous link availability and fault-tolerant data exchange.

The registration and authorisation workflow (**Figure 3.17**) enables each ATMACA node to declare its identity, operational role, and capabilities to the ATM Server during startup. This interaction ensures that every node (whether it be an ATC Agent, CM Agent, mobile client, stationary client, or Application server) is correctly provisioned with the appropriate agent mappings, network roles, and adjacent entity relationships. Registration is initiated immediately after configuration is loaded from the management station and is completed through the exchange of standardised REGISTRATION_REQUEST and REGISTRATION_RESPONSE messages. The response enables the node to join the ATMACA runtime with validated identity and connectivity information, forming the trusted base for all higher-layer protocol interactions.

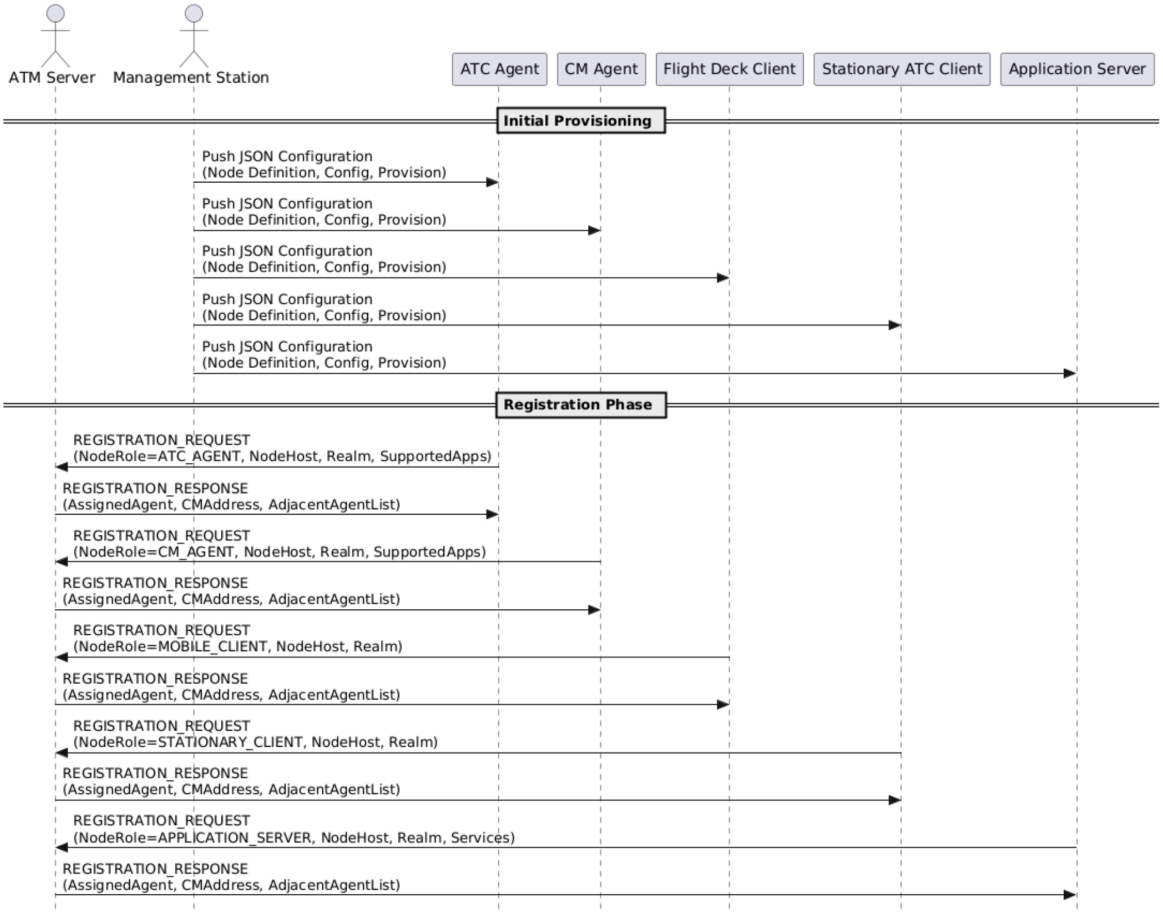

Figure 3.17: ATM Node Registration and Authorisation Flow

The DLIC function enables the establishment and maintenance of operational presence for ATC clients (e.g., controller systems, aircraft) within the ATMACA architecture. This module orchestrates the logical association between client applications and their corresponding operational context by facilitating connection initiation, context updates, inter-agent handovers, and peer forwarding. It ensures seamless identification, mobility support, and system-wide synchronisation, especially across distributed CM/ATC Agents and operational sectors.

DLIC uses structured message exchanges between ATC clients, ATC Agents, and FlightDeck clients to manage the lifecycle of operational presence and connectivity. The core capabilities include logon validation, session continuity, agent contact handovers, and support for ground message forwarding.

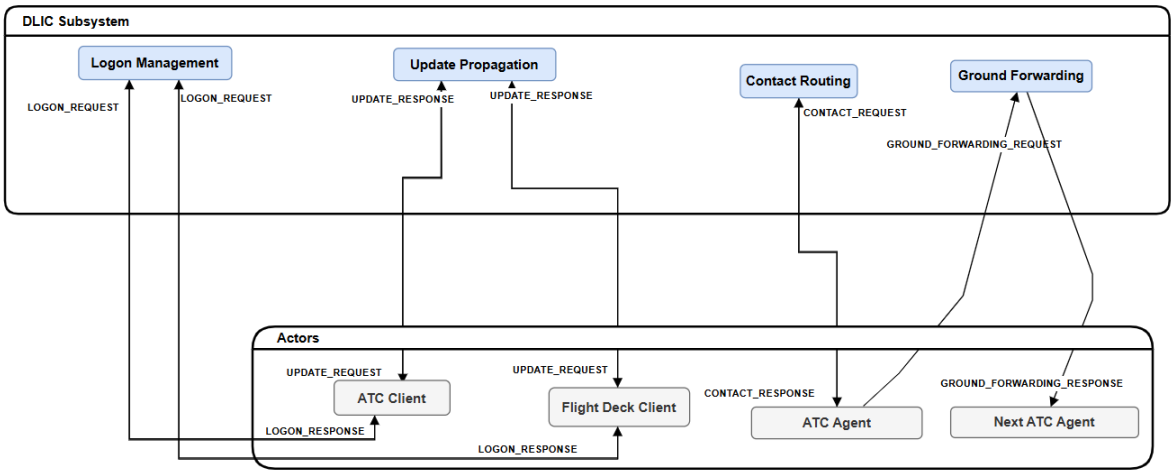

Figure 3.18: DLIC Functional Subsystem

Figure 3.18 outlines the core functional subsystems that compose the DLIC layer of the ATMACA protocol. It showcases the message-driven interactions between system actors (e.g., Flight Deck Clients, ATC Clients, and ATC Agents) and the four primary DLIC subsystems: logon management, update propagation, contact routing, and ground forwarding. Each subsystem is responsible for handling specific message types and operations that together ensure reliable operational presence, client mobility, and inter-agent coordination. The structure emphasises modularity and separation of concerns, supporting distributed runtime environments and flexible deployment.

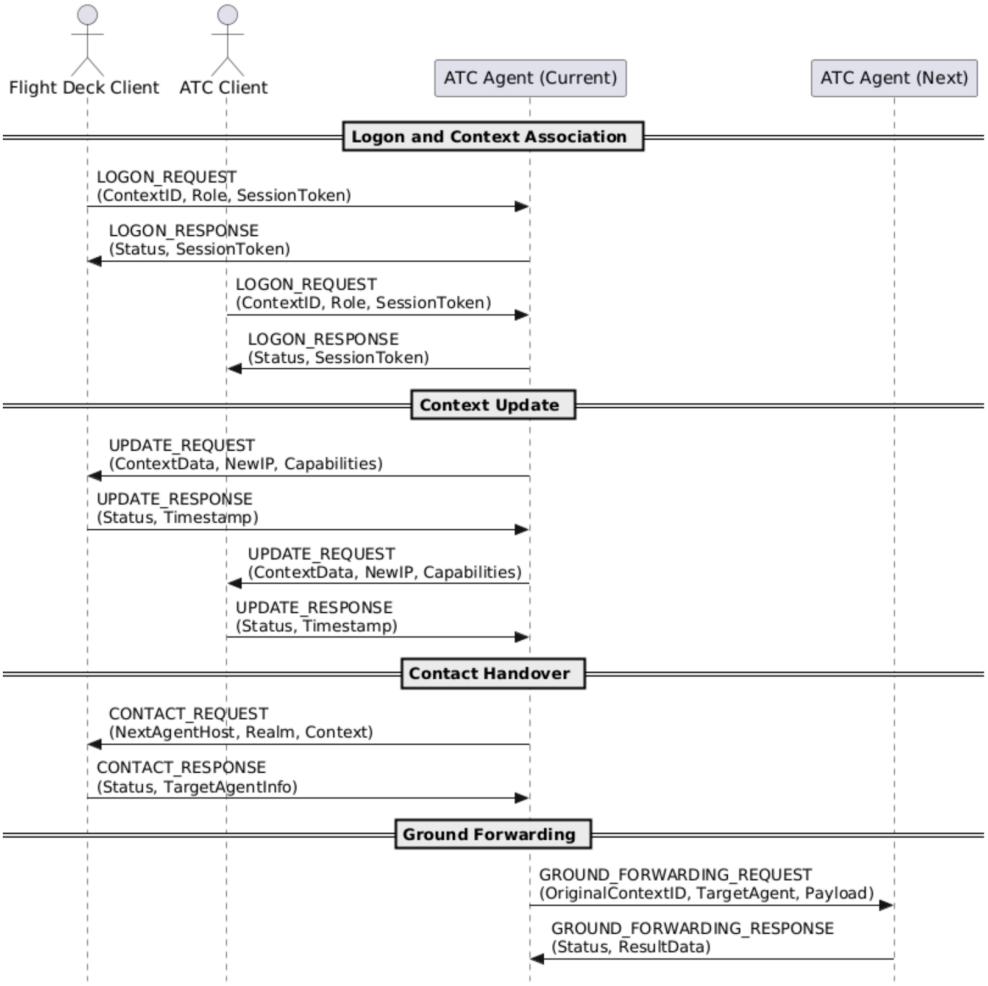

Figure 3.19: DLIC Message Flow

Figure 3.19 shows the full message exchange lifecycle of the DLIC system. It depicts how Flight Deck Clients and ATC Clients initiate their operational presence via LOGON messages and how ATC Agents respond. The diagram also illustrates how ATC Agents propagate context updates to clients, initiate handover contact procedures, and coordinate with other ATC Agents through ground forwarding messages. The clear separation between client-agent and agent-agent communication ensures scalability and isolation in distributed environments. **Figure 3.20** illustrates the logon flow for the Application Server.

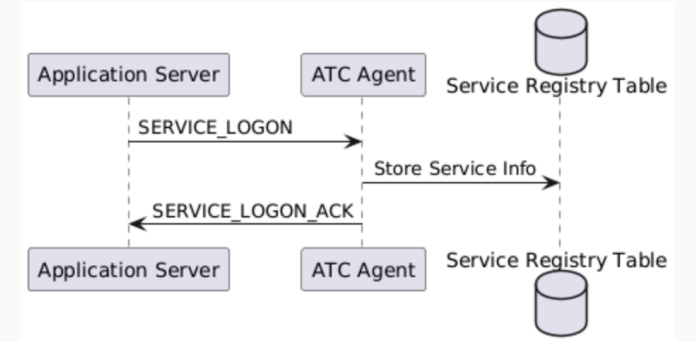

Figure 3.20: Application Server Logon Flow

3.4 Mobility Management Module

The mobility management module operates as a cross-cutting orchestration layer that binds together the capabilities of context, session, connection, and service management in response to mobility events. It does not directly manage resources or connections; instead, it coordinates and delegates tasks to lower-layer modules responsible for binding, session state, context ownership, and message routing. This architecture introduces additional messaging workflows, extended reachability logic, and continuity coordination across ATC Agents and CM Agents. Each mobility scenario is tied to a message flow and context/session lifecycle.

The mobility management module is responsible for ensuring service continuity, consistent session state, and resilient operational continuity as clients or services move across domains, agents, or physical nodes. This module abstracts various forms of mobility in air traffic control scenarios, mapping them to the appropriate DLCM subcomponents based on their functional responsibilities.

ATMACA recognises four distinct types of mobility, each mapped to a specific DLCM submodule (**Table 3.4**):

- User Mobility: supported by the context management module
- Session Mobility: supported by the session management module
- Terminal (Station) Mobility: supported by the connection management module
- Service Mobility: supported by the connection management module and session management module

Each type of mobility is handled with role-specific logic, protocol messaging, and routing coordination, ensuring that transitions are seamless and resilient across the distributed ATMACA network.

Table 3.4: Mobility Types and their managing modules

Mobility Type	Managed By	Purpose
User Mobility	Context Management Module	Allows users (e.g., controllers, pilots) to dynamically change contexts while retaining roles and operational continuity.
Session Mobility	Session Management Module	Enables active session transfer between controllers or agents without disruption during handovers.
Terminal Mobility	Connection Management Module	Maintains communication when devices change their physical or IP attachment points.
Service Mobility	Connection & Session Management Modules	Ensures services are reachable and consistent, even if they or their host platforms move.

All mobility types aim to preserve service continuity ensuring no disruption in operational data exchange, control, or availability, regardless of node movement or transitions. The mobility management module enables uninterrupted service, communication, and session persistence as clients, users, or services move across network domains or logical contexts within the ATMACA architecture. It coordinates dynamic transitions such as controller handovers, terminal IP changes, service node migrations, and airborne session persistence.

The module acts as a distributed coordination layer that monitors and reacts to mobility-triggering events, ensuring system consistency and transparent failover. Each mobility type is mapped to a dedicated DLCM submodule (e.g., context management for user mobility). Its primary role is to abstract mobility handling across all layers and maintain operational continuity without protocol interruption or service loss.

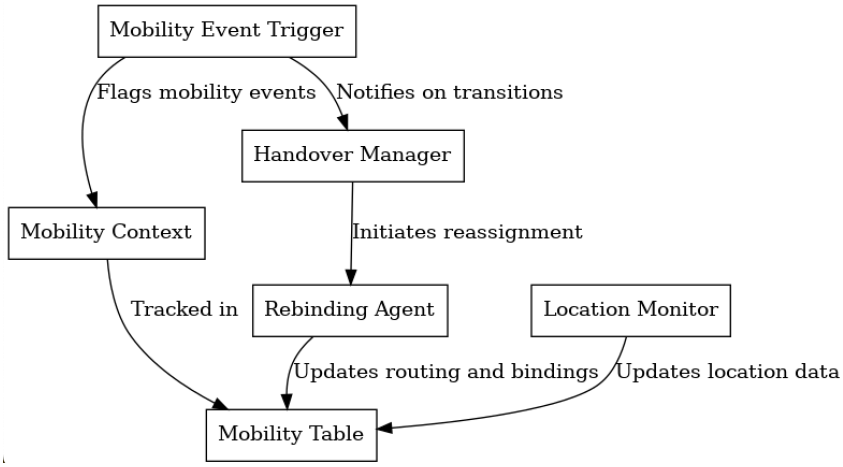

Figure 3.21: Mobility Management Architectural Elements diagram

Subsystems like MobilityUpdateHandler, SessionHandoverCoordinator, and ContextReassociationManager work with registries such as BindingRegistry, SessionTable, and ServiceRegistry to provide resilience across dynamic mobility transitions.

The mobility management module provides the logic and infrastructure required to maintain communication continuity as nodes move across operational domains, network boundaries, or airspace regions. The architecture (**Figure 3.21**) is composed of several key elements, each contributing to real-time mobility tracking, context migration, and seamless session reassignment:

- Mobility Context: Represents the current mobility state of a node, including its last known position, active role, and network association.
- Mobility Table: A dynamic record of all mobile entities in the system, used to track location updates, movement history, and active bindings.
- Handover Manager: Coordinates role transitions and communication handoffs when nodes move between agents, facilities, or control areas.
- Rebinding Agent: Manages the reassignment of session, context, and transport associations during mobility events to ensure service continuity.
- Mobility Event Trigger: Detects movement conditions that require system-level response, such as FIR crossing, agent boundary transitions, or connectivity changes.
- Location Monitor: Continuously tracks node location using identifiers or beacon updates, feeding data to mobility controllers for proactive handover decisions.

Figure 3.22 depicts how the mobility management subsystems interact with architectural components to track mobility state, manage handovers, and preserve service continuity.

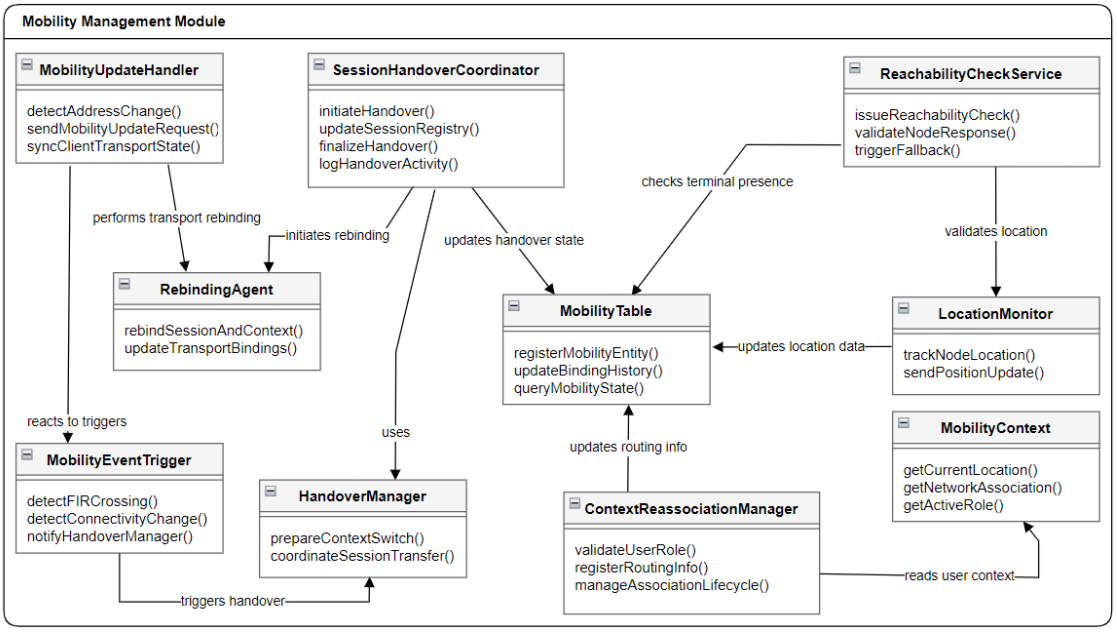

Figure 3.22: Mobility Management Module – Functional & Architectural Mapping

The mobility management module is structured around dedicated coordination components that interface with the core DLCM subsystems (context management, session management, connection management). These architectural elements ensure seamless mobility transitions while maintaining operational integrity and service continuity. The primary coordinators and interactions are

SessionHandoverCoordinator:
Acts as the orchestrator for transferring session ownership between operational contexts. It collaborates directly with the SessionManager to:

- Initiate session handover (SESSION_HANDOVER_REQUEST)
- Update ownership in the session registry
- Finalise handover (SESSION_TRANSFER_COMPLETED)
- Maintain session consistency and logging throughout the handover process

ContextReassociationManager:
Handles dynamic reassociation of users and terminals with existing or new contexts. It works with the ContextManager to:

- Validate user-role permissions for new context association
- Register updated routing information via ATTACH_REQUEST
- Manage association lifecycle including disassociation or mirroring modes

MobilityUpdateHandler:
Responsible for detecting client IP or transport changes during movement. It operates with the connection management layer to propagate MOBILITY_UPDATE_REQUEST/RESPONSE sequences and synchronise client address state with active bindings.

ReachabilityCheckService:
Monitors terminal availability in uncertain network conditions. It periodically

issues NODE_REACHABILITY_CHECK_REQUEST when a known endpoint becomes unreachable and expects a NODE_REACHABILITY_CHECK_RESPONSE to validate client liveness.

These components ensure a decoupled and service-resilient architecture that abstracts the complexity of node movement while preserving the ATMACA system's distributed design. **Table 3.5** presents the linkage between the mobility management subsystems and their architectural elements, illustrating how dynamic handovers and rebinding are supported in real time.

Table 3.5: Mobility Management Table – Linkage Table

Functional Subsystem	Related Architectural Elements	Interaction Summary
Session Handover Coordinator	Mobility Context, Mobility Table, Rebinding Agent	Initiates and manages session transfers by updating routing and session bindings stored in the table.
Context Reassociation Manager	Mobility Context, Mobility Table, Mobility Event Trigger	Validates and rebinds context metadata upon role or agent transitions, using updated mobility data.
Mobility Update Handler	Mobility Context, Transport Pool, Rebinding Agent	Detects transport-level mobility, and triggers rebinding operations through the pool and agent.
Reachability Check Service	Mobility Table, Location Monitor, Mobility Event Trigger	Continuously monitors node liveness and responds to failures or uncertainties by flagging transition needs.

The four mobility types are separate but coordinated to preserve service continuity. User mobility changes the active context or role via context management. When a user (for example, a controller) switches context, any ongoing sessions bound to that context can be transferred using session management handover messages rather than being dropped. Context changes may trigger session mobility. Terminal mobility keeps the device reachable. The client informs its ATC Agent of new transport parameters and answers reachability checks, and the agent rebinds routing so existing context and sessions continue without interruption. Finally, service mobility is brokered by the ATC Agent, which maintains a registry of available services. When a service provider migrates or the agent's routing decision changes, the agent updates the service association and notifies active clients. Clients acknowledge keeping delivery continuous across moves. If the client re-homes to a new ATC Agent, the new agent restores the client's service bindings from the registry, reevaluates the best provider, and if the endpoint changes issues UPDATE_REQUEST acknowledged with UPDATE_RESPONSE to redirect delivery. Overall, the context (user), session, connection (terminal), and service modules each manage their tier but interlock.

3.4.1 User Mobility

The user mobility subsystem ensures dynamic context reassignment and real-time role transitions for operators, such as controllers or pilots, without compromising session continuity or access privileges. This capability is essential in operational environments where role handover, collaborative monitoring, or workstation shifts are frequent and necessary. The user mobility architecture is composed of three cooperating subsystems:

- Context Association Manager: Handles CONTEXT_ASSOCIATION and CONTEXT_DISASSOCIATION REQUEST messages. This allows users to bind or unbind themselves to/from contexts in accordance with their operational role and access control. The context association manager tracks context ownership metadata and supports dual-role binding (e.g., controlling/monitoring).
- Presence Binding Registry: Manages a real-time routing table of active user-to-context associations. It tracks multiple attachment points (IP endpoints) using ATTACH/DETACH message flows for devices supporting mirroring or standby and resolves multi-user scenarios by enforcing role validation and non-conflict policies.

- Context Transition Notifier: 1) Triggers notifications across CM Agents and ATC Agents when a context ownership or presence state changes. 2) Ensures consistent update propagation to all affected routing tables. 3) Validates that new user-context pairs comply with access rules and operational state requirements.

This subsystem enables rapid handovers in control rooms, redundancy setups for ATC operators, and dynamic mirroring or observation scenarios across workstations without requiring session teardown or data loss. **Figure 3.23** shows the message flow for user mobility. This scenario illustrates how a user dynamically associates with an existing or newly created context and updates routing information for communication continuity.

The use of CONTEXT_ASSOCIATION_REQUEST, CONTEXT_ASSOCIATION_RESPONSE, and the corresponding ATTACH/DETACH messages is illustrated to reflect changes in user location or terminal presence. The process allows for hot-standby participation, monitoring roles, or mirroring setups. This diagram showcases how the system synchronises user-context bindings and manages routing tables across the CM Agent and ATC Agent to support seamless user transitions.

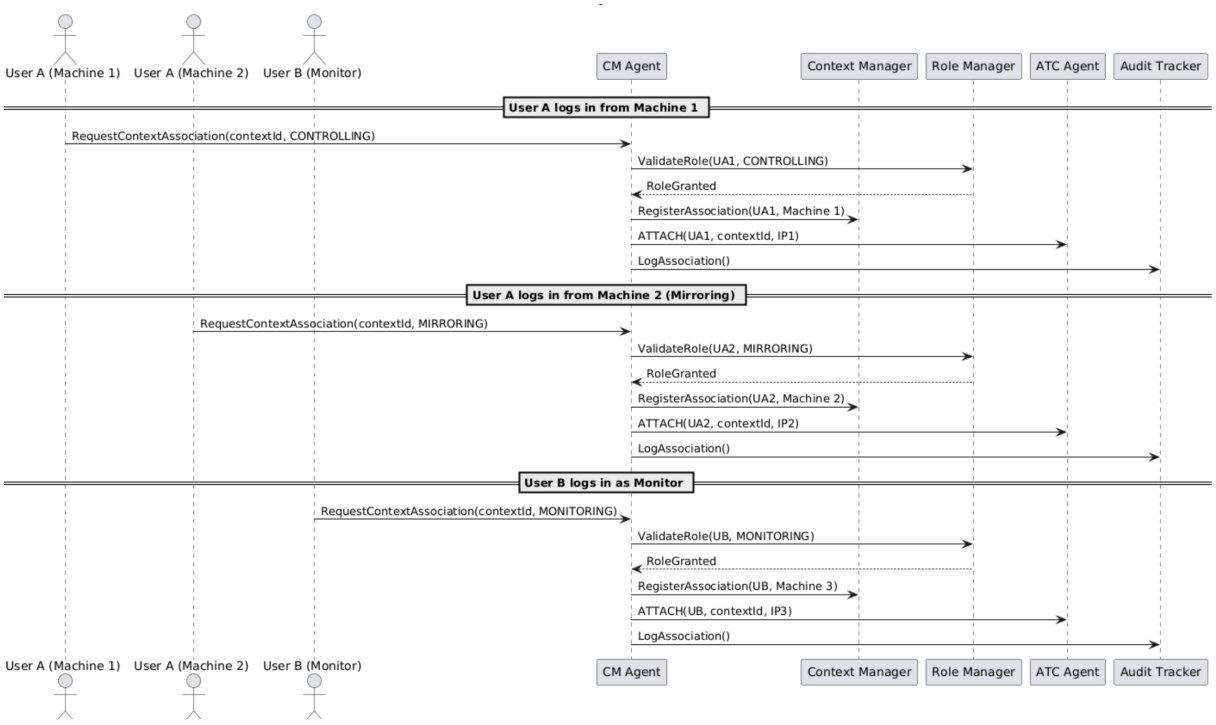

Figure 3.23: User Mobility Message Flow

3.4.2 Session Mobility

The session mobility subsystem in the ATMACA architecture is responsible for preserving live communication sessions between users and aircraft (or between peers) as operational responsibilities shift such as during sector handovers or controller transitions. It ensures that all parties remain logically connected and synchronised, even as session ownership moves between contexts.

The session mobility architecture consists of the following tightly integrated functional components:

- Session Lifecycle Coordinator: Initiates and manages the lifecycle of sessions, particularly during transfers. It uses messages like SESSION_HANDOVER_REQUEST, SESSION_TRANSFER_COMPLETED, and SESSION_OWNER_CHANGE to coordinate movement of active sessions across users or contexts. In addition, it ensures that the session remains valid, uninterrupted, and securely transitioned during handover.
- Session Ownership Tracker: Maintains authoritative records of current session owners and authorised contexts. It works in coordination with CM Agents to validate new owners and update routing metadata post-transition. It prevents conflicts or overlaps through strict handover validation rules and rejection handling.
- Session Routing Bridge: Coordinates message flows between ATC Agents during inter-agent session transitions. It ensures all post-handover messages are redirected to the new owner using updated endpoint references. In addition, it triggers additional UPDATE, ATTACH, or DETACH sequences if the new owner resides on a different agent or network zone.

This subsystem guarantees uninterrupted session handovers by orchestrating ownership negotiation, state preservation, and real-time routing adjustment. It enables flexible operational control handoffs, enhances resilience in high-load sectors, and aligns with global interoperability standards such as ICAO GOLD.

Session mobility refers to the ability to transfer or migrate active communication sessions between operational contexts without disrupting the service continuity. This form of mobility is especially important in air traffic control environments where controllers or facilities may need to hand over session responsibility during sector transitions, duty changes, or failover scenarios.

Figure 3.24 highlights the lifecycle of a session handover between two operational contexts within the ATMACA architecture. It visualises how a session initiated by one controller (or context owner) is securely transferred to another without service disruption. This message sequence uses SESSION_HANDOVER_REQUEST, SESSION_OWNER_CHANGE, and SESSION_TRANSFER_COMPLETED to coordinate between ATC Clients, ATC Agents, and CM Agents. Additionally, UPDATE, ATTACH, and DETACH messages may be involved to reflect connection or routing state changes. This diagram captures the complexity of distributed session control while ensuring data integrity, authorisation, and operational handoff tracking.

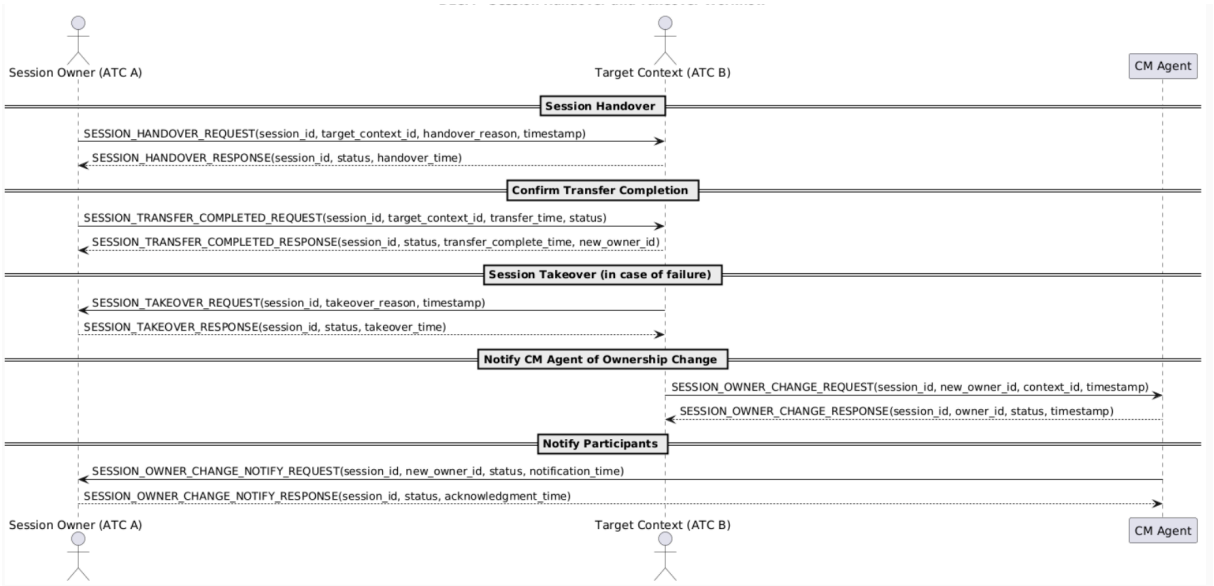

Figure 3.24: Session Mobility Message Flow

3.4.3 Terminal Mobility

The terminal mobility subsystem ensures that mobile endpoints can maintain operational connectivity as they move across physical or network boundaries. It is tightly integrated with the DLCM connection management layer and includes specialised monitoring and recovery mechanisms tailored for client movement and reconnection scenarios. Terminal mobility includes three main components:

- Mobility Update Manager: 1) Handles transport layer changes, such as IP address updates, triggered by Layer 2 or Layer 3 handovers. 2)Initiates the MOBILITY_UPDATE_REQUEST from client to ATC Agent to inform about the new transport state. 3) ATC Agent replies with MOBILITY_UPDATE_RESPONSE and updates routing and session bindings accordingly. 4)Ensures minimal disruption to ongoing sessions or services during terminal movement.

- Reachability Check Engine: 1) Activated when an ATC Agent detects a prolonged silence from a previously active client. 2) Periodically issues NODE_REACHABILITY_CHECK_REQUEST messages to query the client's presence. 3) If a response (NODE_REACHABILITY_CHECK_RESPONSE) is received, the ATC Agent revalidates and rebinds the client to the existing context/session. 4)Triggers timeout-based cleanup if no response is received within threshold, marking the client as unreachable.
- Dynamic Binding Registry: 1) Maintains real-time state of all terminal bindings including IP address, role, session association, and last activity timestamp. 2) Used to route future context, session, or service messages to the correct transport-level endpoint. 3) Ensures single-point control for rebinding, handover tracking, and disconnection validation.

  These components work together to deliver seamless terminal mobility, enabling systems to detect movement, adjust routes, and confirm availability with minimal human intervention. The design ensures that critical ATC systems can tolerate dynamic mobility patterns without service interruption or data loss.

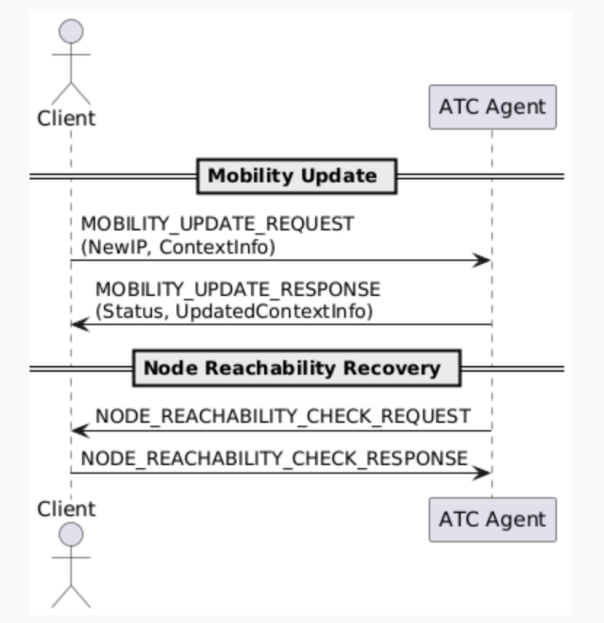

Figure 3.25: Terminal Mobility Message Flow

Figure 3.25 illustrates the refined terminal mobility message exchange flow within the ATMACA architecture. Unlike previous models, this version excludes the ATTACH and DETACH message sequences and focuses solely on mobility-specific interactions. It highlights the use of MOBILITY_UPDATE_REQUEST and MOBILITY_UPDATE_RESPONSE messages for informing the system of a client's updated IP or transport parameters during mobility events. Moreover, it includes the NODE_REACHABILITY_CHECK_REQUEST and NODE_REACHABILITY_CHECK_RESPONSE pair, used when the ATC Agent detects potential loss of communication with a known client and needs to validate its presence or initiate recovery. This sequence ensures minimal disruption to session and service continuity during client movement or connectivity loss.

3.4.4 Service Mobility

The service mobility subsystem manages the availability, routing, and continuity of services (e.g., CPDLC or data fusion functions) when either the client, the service provider, or both are mobile. It supports dynamic discovery, handover, and rerouting mechanisms to ensure uninterrupted service interaction regardless of physical location or network changes. The design supports both stationary and mobile application servers and uses the ATC Agent as a logical broker for service traffic. The service mobility includes four components:

- **Service Registry Manager**
    - Maintains a centralised registry of available services and their associated application servers.
    - Populates the registry during application server LOGON flows and updates it upon receiving new UPDATE_REQUEST messages.
    - Maps services to the Application Server URI, IP address, realm, and capabilities for efficient lookup and routing.

- **Service Router**
    - Routes SERVICE_REQUEST messages from clients to the appropriate Application Server via the ATC Agent.
    - Ensures routing to the closest or most appropriate server using current registry and network topology.
    - Handles bidirectional flow by processing responses like SERVICE_DELIVERY, SERVICE_ERROR, and SERVICE_REJECT.

- **Mobility-Aware Delivery Handler**
    - Supports dynamic rerouting when a service provider migrates or a client connects to a different ATC Agent.
    - Responds to UPDATE_REQUEST from Application Servers and synchronises changes with all dependent clients via UPDATE_RESPONSE.
    - Guarantees service continuity by redirecting in-progress service sessions to newly available endpoints.

- **Session Bridge (for Session-based Services)**
    - Creates and tracks service sessions between clients and Application Servers when services are stateful (e.g., continuous or contract-based).
    - Binds session ID to both client and server context. Ensures session integrity during service mobility events.
    - Coordinates with the session manager to avoid duplication or loss of service context.

These subsystems enable ATMACA to offer a resilient, proximity-optimised, and mobility-aware service layer. Whether services are hosted on fixed ground infrastructure or mobile airborne nodes, the system guarantees their availability and continuity through real-time registry updates, smart routing, and session coordination.

To support seamless transitions across network boundaries and operational domains, the mobility management workflow orchestrates coordinated actions between clients, agents, and servers. The following subsections detail each stage of this process, highlighting how session continuity, context updates, and service integrity are maintained throughout mobility events.

Service mobility refers to the ability of services to remain reachable and operational even as their hosting environment changes. ATMACA supports two forms of service mobility: mobility of fixed service nodes across infrastructure, and mobility of service nodes such as aircraft acting as airborne service providers.

Figure 3.26 illustrates the message flow that enables service mobility within the ATMACA architecture. This includes both ground-based and mobile, such as Application Servers and aircraft-hosted service providers. All service-related communication is strictly routed through the ATC Agent, which maintains a dynamic service registry table to map service requests to the most appropriate and closest service node.

The diagram highlights the standard service interaction flow: SERVICE_REQUEST is initiated by a client, forwarded to the appropriate Application Server via the ATC Agent, followed by SERVICE_PROCESSING, SERVICE_DELIVERY, or error/rejection messages. Additionally, UPDATE_REQUEST and UPDATE_RESPONSE messages enable rerouting or migration of

services to new endpoints. This model guarantees continuity and proximity in service delivery, even as clients or service providers move across agents or domains.

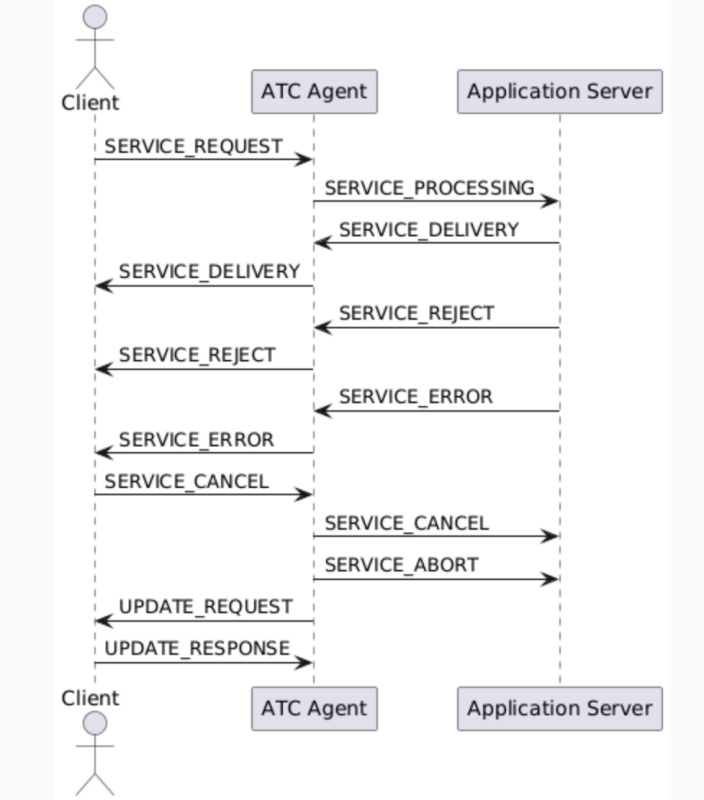

Figure 3.26: Service Mobility Message Flow

4 CONCLUSION

This deliverable specifies the DLCM application within the ATMACA protocol stack. DLCM generalises DLIC into modular services (context, session, connection, mobility, presence, and service delivery management) with explicit message sets, defined role semantics (controlling, mirroring, monitoring), and fail-safe handover procedures. It clarifies the separation of responsibilities between the CM Agent and ATC Agent, and consolidates user, session, terminal, and service mobility to maintain service continuity through controller handover, sector transfer, and transport rebinds.

As next steps, the project will integrate the ATMACA base communication protocol, DLCM, and the CPDLC application with the HMI. This phase will involve extensive simulation and emulation exercises to verify functional correctness and assess performance. Predefined scenarios under Task T2.4 will be used, and Key Performance Indicators (KPIs) will be measured. The results and lessons learned, including those from the planned flight trial, will be documented in Deliverable D6.2.

5 REFERENCES

[1] ICAO Doc 9896, Manual on the Aeronautical Telecommunication Network (ATN) using Internet Protocol Suite (IPS) Standards and Protocols, 2nd Edition, 2015.

[2] EUROCONTROL SPEC-192, EUROCONTROL Specification for Data Link Common Services for the Aeronautical Telecommunication Network (ATN), Edition 1.0, 11 December 2023.

[3] ICAO Annex 10 – Aeronautical Telecommunications, Volume III – Communication Systems, 2nd Edition (July 2007).

[4] ATMACA-SESAR, Deliverable D2.1, Review of Current and Future ATM Communication Network, 2024.

[5] IETF RFC 5522, Network Mobility (NEMO) Route Optimization Requirements for Operational Use in Aeronautics and Space Exploration Mobile Networks, October 2009, https://doi.org/10.17487/RFC5522.

[6] IETF RFC 6830, The Locator/ID Separation Protocol (LISP), January 2013, https://doi.org/10.17487/RFC6830.

[7] ATMACA-SESAR, Deliverable D3.1, ATMACA Protocol Design Plan, 2025.

[8] ATMACA-SESAR, Deliverable D3.2, ATMACA Protocol Development Technical Diagram, 2025.

[9] ATMACA-SESAR, DES HE SESAR solution D2.2. ATMACA Functional Requirements Document (FRD)-Initial, 2025.

[10] ATMACA-SESAR, Deliverable D2.3 ATMACA Operational Service and Environment Description (OSED), 2025.

[11] ATMACA-SESAR, Deliverable D4.2, CPDLC Application Design Document, 2025.

Appendix A. DLCM Service Interfaces Catalogue

This appendix describes the public DLCM service interfaces available to applications (e.g., CPDLC, DFIS). In ATMACA deployments, all nodes handle registration with the ATM Server and DLIC logon in the background. Applications may call these interfaces once the node is logged on.

Application (registration/binding)

Method	Type	Purpose	Inputs	Output
Register(req)	Call	Announce application and negotiate capabilities.	App_Id, Node_Type, Return_Endpoint, Capabilities, Timeout	RegisterAppResponse{App_Token, Assigned_Namespace}
Unregister(req)	Call	Graceful application shutdown and cleanup.	App_Token	Ack
Heartbeat(ping)	Call	Liveness check / keep-alive.	App_Token, When	Ack
OnShutdown(h)	Event	DLCM is about to restart/ reconfigure.	—	RAII subscription / callback

Context (facility/sector/role)

Method	Type	Purpose	Inputs	Output
Pull(req)	Query	Fetch context snapshot (optionally delta since timestamp).	ContextId, since?, timeout?	ContextData{metadata, last_updated}
OnUpdate(h)	Event	Context metadata changed.	—	RAII subscription / callback
OnRoleChange(h)	Event	Role changed (e.g., Monitoring → Controlling).	—	RAII subscription / callback

Session (application state)

Method	Type	Purpose	Inputs	Output
Pull(req)	Query	Get session data.	SessionId, since? timeout?	SessionData{payload, last_updated}
Push(req)	Call	Persist state changes (e.g., msg status, timers).	SessionId, app_id, data_block, sequence_no, timeout?	Ack
OnOwnerChange(h)	Event	Ownership changed.	—	RAII subscription / callback
Create(req)	Call	Establish an application session.	context, app_id, role, credentials, timeout?	SessionCreateResponse{session_id, …}
Recover(req)	Call	Attempt session recovery after disruption.	session_id, context, failure_reason, timeout?	SessionRecoveryResponse

Connection (DLIC)

Method	Type	Purpose	Inputs	Output
Logon(req)	Call	Perform DLIC logon before using services.	context, role, session_token, timeout?	LogonResponse{success, session_token}
OnUpdate(h)	Event	DLCM/agent contact or capability changed.	—	RAII subscription / callback
AckUpdate()	Call	Acknowledge update event.	—	UpdateAck
OnContact(h)	Event	Handover/contact with new ATSU.	—	RAII subscription / callback
AckContact()	Call	Acknowledge contact event.	—	ContactAck
GroundForward(req)	Call	Ask DLCM to ground-relay payload.	origin, target_agent, payload, timeout?	GroundForwardingResponse{delivered, result}

Health (links/transport)

Method	Type	Purpose	Inputs	Output
OnLinkHealth(h)	Event	Link state changed (Up/Down/Degraded).	—	RAII subscription / callback
GetLinkStatuses()	Query	One-shot snapshot of all links.	—	vector<LinkStatus>

Co-funded by the European Union