



Artificial Intelligence in Manufacturing leading to Sustainability and Industry5.0

Project Acronym:

AIMS5.0

Grant agreement no: 101112089

Deliverable no. and title	**D3.2 – Final micro service architecture requirements and architecture realization**	
Work package	WP 3	Cyber Architecture
Lead contractor	Infineon Technologies AG Thomas Gutt, mailto:thomas.gutt@infineon.com	
Deliverable responsible	LTU Jerker Delsing, jerker.delsing@ltu.se	
Version number	V1.0	
Date	23/05/2025	
Status	Final	
Dissemination level	Public (PU)	

Copyright: AIMS5.0 Project Consortium, 2025

Table of contents

1. Introduction 4
2. Cyber architecture requirements 5
3. Cyber architecture 6
3.1 Architecture base line (LTU) 6
3.2 Micro service-based platform enabling AI integration 8
3.2.1 System of systems composability for AI functionality 9
3.2.2 Software deployment and management for AI functionality 12
3.2.3 Safety and security in AI-supported production automation 17
3.2.3.1 NFC/RFID for security 17
3.2.3.2 Intelligent Antennas and Secure IoT Architectures 23
3.2.4 Robust real-time networking for AI applications 25
4. The AI-gym - AI functionality as services 31
5. Standardization-based feedback and guidelines 34
5.1 Cybersecurity Standards and Frameworks for Industrial Automation 34
5.2 Standards and Guidelines for Trustworthy AI Integration 35
5.3 Standardization Feedback Loop 35
6. Conclusions 37
7. Appendix 38
7.1 Appendix Requirements 3.2 (PUB) 38
7.2 Appendix Requirements 3.5 (SEN) 38

List of figures

Figure 1: The microservice cyber architecture will provide an SOA architecture infrastructure supporting fundamental service-oriented properties like Look-up, Late binding, and Loose coupling, plus a number of support functionalities to build an integration platform tailored to for example a specific domain. Using this platform working domain solutions can be developed. 6
Figure 2: Eclipse Arrowhead 4.6.1 core and support microsystems 7
Figure 3: A custom device of SSN 10
Figure 4: MLOps workflow within SSN based on Arrowhead 11
Figure 5: Savvy SmartBox Edge device 13
Figure 6: Dell PowerEdge Fog server 13
Figure 7: Multi-tier, inter-cloud AH integration 14
Figure 8: Edge / Fog data & AI pipeline 15
Figure 9: Edge / Fog / Cloud AI pipeline 15
Figure 10: AH Local Cloud interoperability with SmartBox & machine 16
Figure 11: Data collection, processing & model training 16
Figure 12: RF sequence - write to tag 19
Figure 13: TAM1 settings 19
Figure 14: RF sequence of tag authentication 1 20
Figure 15: TAM2 settings 21
Figure 16: RF sequence of tag authentication 2 21
Figure 17: SAM AV3 settings - user interface 22
Figure 18: Exemplary Hierarchical Schedule Tables for 2 FEC 27
Figure 19: Achieved Target Utilization and VCPU Overhead 29
Figure 20: The three layer architecture of the AI-gym 32
Figure 21: Use Case 11 Greenhouse Environment 33

1. Introduction

This WP3 deliverable addresses the cyber architecture requirements as collected from use cases and other technology work packages. The first architectural concepts that address these requirements have been defined and fundamentally validated in use cases. The architectural concepts are intended to build upon previous work in stepping-stone projects like Arrowhead Tools and Productive5.0, which both have made significant contributions to the open-source project Eclipse Arrowhead. The intention of AIMS5.0 and WP3 is to continue such contributions to the Eclipse Arrowhead, further building its usefulness in real industrial environments.

2. Cyber architecture requirements

Requirements for microservice-based cyber architecture have been pursued within T3.1. Requirements have been collected from use cases in WP5 and 6 and the technology WP1-3.

These two sources give two different types of requirements. The technology work packages provide, in general, more fundamental technology requirements, while the use case provides more application-oriented requirements. These differences need to be considered when evaluating the fulfillment of the requirements.

All requirements collected so far are listed in Appendix – Requirements.
For each requirement, the following information is collected.

- Submitter name
- Submitter email
- Providing task
- Category
- Integration platform, Engineering tools, Engineering tools chain integration, Training material, Governance, Verification use case
- Objective
- Requirement description
- Rationale
- Y1 fulfillment
- What's missing
- Evaluation (text)

The requirement Appendix is a living internal document to which further requirements will be added, and the fulfillment evaluation will be added for Y2 and Y3. Thus, a clear picture of the advancement of cyber architecture and its reference implementation will become visible.

Currently, there are 45 requirements collected. For a smaller number of these requirements, an analysis of the Y1 fulfillment has been possible and is thus as well commented on in the Appendix. One division of the appendix is public and released as a part of this document (D3.2), while the other division is confidential and will be included in D3.5.

Additional requirements have been collected from the use cases and WP1, 2 & 3. To total set of requirements with initial fulfillment analysis can be found in Annex Requirements 3.2 and Annex Requirements 3.5.

3. Cyber architecture

3.1 Architecture base line (LTU)

Cyber architecture is fundamentally based on a microservice and service oriented architecture based paradigm. This paradigm has been used in previous projects like Productive5.0 and Arrowhead Tools. All previous results have been transferred to the open-source project Eclipse Arrowhead, http://www.arrowhead.eu and http://github.com/eclipse-arrowhead.

The work on the cyber architecture within AIMS5.0 is based on the current status of Eclipse Arrowhead and is further guided by the collected requirements and the project DoA.

The cyber architecture and its implementation in real-world use cases can be described with the help of Figure 1.

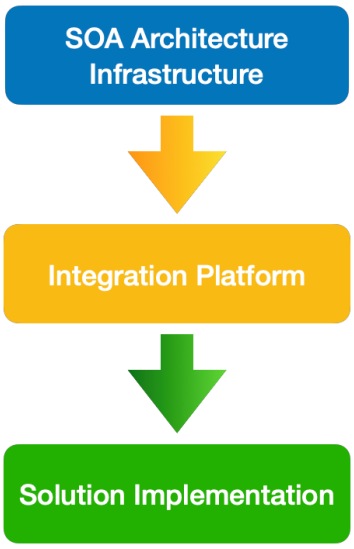


Figure 1: The microservice cyber architecture will provide an SOA architecture infrastructure supporting fundamental service-oriented properties like Look-up, Late binding, and Loose coupling, plus a number of support functionalities to build an integration platform tailored to for example a specific domain. Using this platform working domain solutions can be developed.

The SOA architecture infrastructure will provide:

- Fundamental SOA property support: Look-up, Late binding and Loose coupling
- Security support
- Interoperability support
- System of Systems Integration support
- Basic engineering and operations support
- Model-based engineering support

To evaluate the investigated topics, the project architecture base line was chosen as the Eclipse v4.6.1. The released candidate and prototype microsystem are depicted in Figure 2 below.

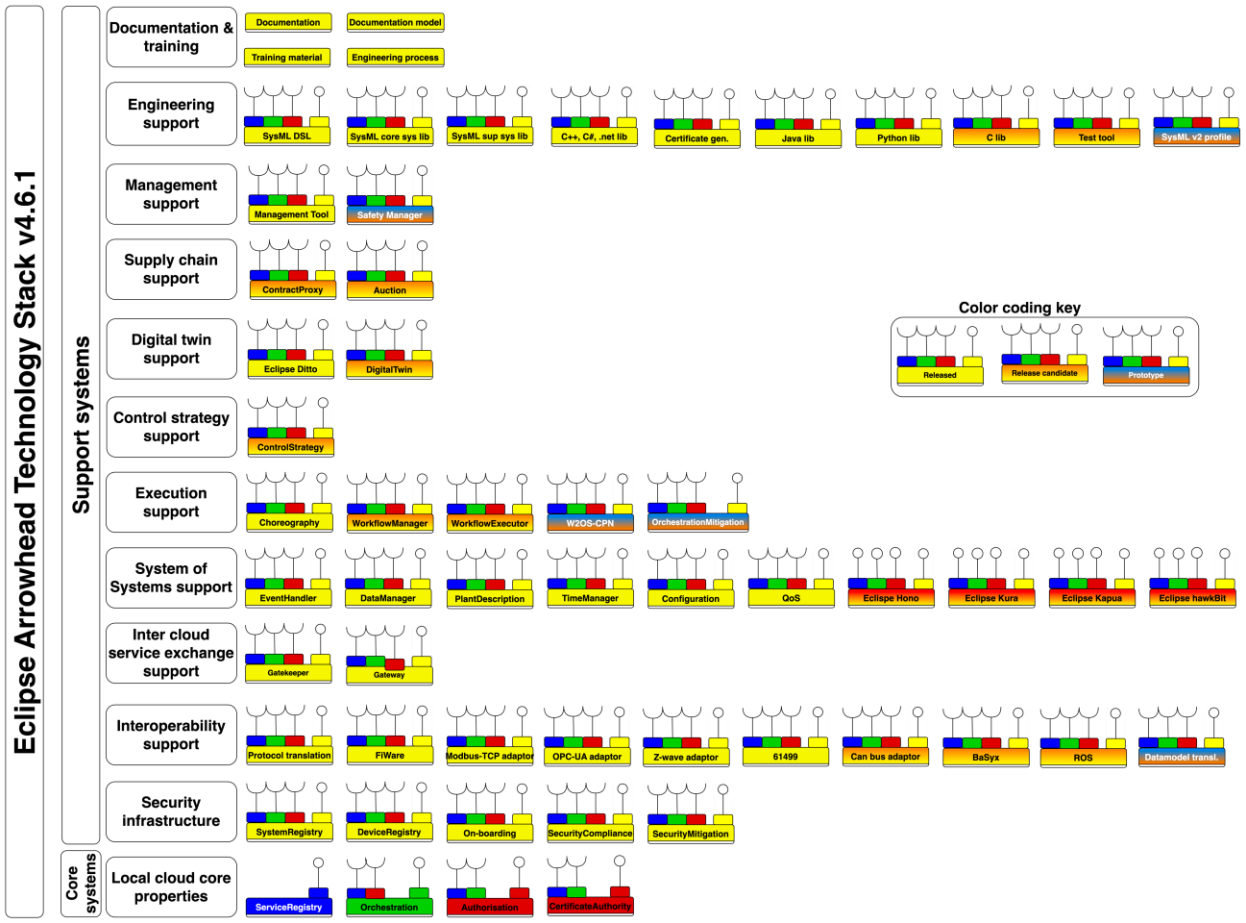


Figure 2: Eclipse Arrowhead 4.6.1 core and support microsystems

The WP3 ambition is to mature and extend this baseline. The overall ambition is to make Eclipse Arrowhead the major open source microservice/SOA architecture and reference implementation that is to be deployed on heterogeneous devices from deep edge to fog with Arrowhead fPVN project the sister project of AIMS5.0 has taken the lead for the development of version v5.0 of Eclipse Arrowhead.

WP3 AIMS5.0 supports the high-level ambition by a set of actions addressing the following additional improvements and extensions to the Eclipse Arrowhead architecture and its reference implementations:

- Robustness, interoperability, and composability to legacy OT and IT automation installations.
- System of systems composability for AI functionality
- Software deployment and management for AI functionality
- Safety and security in AI-supported production automation
- Robust real-time networking for AI applications
- The AI-gym, AI functionality as services for rapid integration, testing, and V&V

Compared to our baseline, the following sections describe the advancements during Y2 of AIMS5.0.

3.2 Micro service-based platform enabling AI integration

The micro service-based platform progress beyond the baseline described above is primarily addressing v.5 of Eclipse Arrowhead. The work is driven by the Eclipse Arrowhead Roadmap WG lead by LTU in our role as WP3 lead of AIMS5.0. Primary contributors are LTU, AITIA, BEM, Sinetiq AB, Thingwave AB, University of Groningen and others.

The fundamental mental principles of Eclipse Arrowhead has been thoroughly documented and is openly available at https://github.com/eclipse-arrowhead/roadmap/blob/main/5.0%20Draft/Foundational%20principles/Eclipse%20foundational%20principles%20and%20developments%20v2.pdf.

In addition the Generic System of System Document (GSoSD) for Eclipse Arrowhead v5.0 has been created and is openly available at https://github.com/eclipse-arrowhead/roadmap/blob/main/5.0%20Draft/GSoSD/GSoSD%20Arrowhead%20Core%20Systems%205.0b.docx

The core systems of Eclipse Arrowhead v5.0 are now four. This is one more than in v4.6 due to the splitting of the Authorization system into Authentication and Authorization. This allows for other authentication technologies e.g. OAuth, BankID.

AIMS5.0 contribution, apart from leading the v5.0 WG, is the Arrowhead DSL primarily developed by LTU. The four v5.0 core systems have SysML models developed using the Arrowhead DSL. Further a considerable number of the Arrowhead support system models have been updated to comply with the v5.0 specifications and extensions. Notable extensions which have been implemented in the Arrowhead DSL library are:

- Modeling and adding the SysMonitor service to the models of both core and support systems of the Arrowhead technology stack.
- Update core and support systems to reflect the split of system authentication service consumption authorization into different microsystems and associated microservices
- Use of the new Arrowhead v5.0 naming convention of microsystem, microservices and metadata/attributes.

In addition, modeling of core system interactions using SysML interaction diagrams has been started. The same goes for work-flow modeling using SysML state-machine modeling.

The Arrowhead DSL is further updated to now include mature three stages (TRL 5-6) of the IEC 81346 engineering process. Using the Arrowhead terminology these stages are:

- Black box design,
- White box design and
- Implementation

Early work on the deployment engineering step indicates that the DSL will include at least that step by end of AIMS5.0.

The practical usage of the Arrowhead DSL is facilitated through cooperation with Arrowhead fPVN project and their partner CEA who has implemented the Arrowhead DSL into the Eclipse IDE and Papyrus tools.

Extension to the basic Arrowhead DSL currently created based on SysML v1.6 is a transformation to base the DSL on SysML v2.0. Early analysis work on that indicates that a rather straight forward translation can be made. Once a complete translation of the Arrowhead DSL to SysML v2 complex System of Systems can be captured and modeled in a language having formal semantic base.

A concrete impact of this work is that the Veerum company concretely is working on a translation from SysML v2 to the programing language Dezyne. Once Both SysMLv2 and Dezyne are languages having a formal based semantics. Executable code can be generated by the Dezyne tool. This executable code can in turn be formally verified using tools from TU/e. This implies a formal based tool chain from specifications captured in SysML v2 using the Arrowhead DSL over translation to the Dezyne coding language which can generate compilable C, C++ and C# code on which formal verification can be made using TU/E tools.

The verification of such engineering process and tool chain is facilitated by modeling of a complex smart city use case currently being developed within AIMS5.0 using the Arrowhead DSL.

3.2.1 System of systems composability for AI functionality

During this period, the improvement of the Sentinel Surveillance Network (SSN) has been continued. The development follows two approaches:

- Sentinel Winery Pilot
- MLOps integration

Sentinel Winery Pilot

AITIA had the opportunity to test the SSN in a real-world scenario. The requirement was to detect a flock of starlings (and individual birds also) around a given vineyard to prevent the possible damage caused by the birds. We have deployed multiple instances of the nodes with the intent of testing the robustness and scalability requirements. The pilot required a custom action node which was developed and has the capability of sending alert notifications via e-mails and WhatsApp messages. To detect birds two targeted models were created by using the YOLO v8 deep learning technology from pictures found in public sources, but also pictures taken at the test site. None of the commercial camera was sufficient for our needs (outdoor environment, battery powered, solar-energy charged and direct access via Wi-Fi), therefore a custom device – *Figure 3: A custom device of SSN* - with necessary software were developed. During this time some development on the monitor node was also delivered to be able to display battery charging and CPU temperature level information from the camera devices.



Figure 3: A custom device of SSN

The conclusion of the pilot was that

- the SSN was working reliably,
- the cameras worked reliably, but energy and cost efficiency must be improved,
- that AI performed well, but hallucination has occasionally occurred (model localization eliminates this issue).

Based on the experiences of the pilot, AITIA has started to develop the 2nd prototype of the custom camera device, focusing on energy consumption and cost efficiency. Also, a new action node is under development to integrate remote-controlled gas cannons in order to scare the animals away.

MLOps Integration into Sentinel

The intention of having an AI model training feature within the SSN was a major goal from the beginning. Furthermore, the Sentinel Winery Pilot has proved that model localization is always

required in production environments. Therefore, developing the MLOps integration was our main focus.

The concept is the following:

Users can collect pictures from the SSN database and mark their use-case specific targets (i.e.: humans, vehicles, etc.). After this the model training can be started, which involves the uploading of pictures and annotation data to a dedicated component (Model Manager), that manages the whole process by interacting with other components (Roboflow, Jenkins). When the model training is finished, users can assign the new model to the selected image nodes. Deploying (downloading and applying) the new model is happening automatically. Detailed workflow is described on *Figure 4: MLOps workflow within SSN based on Arrowhead*.

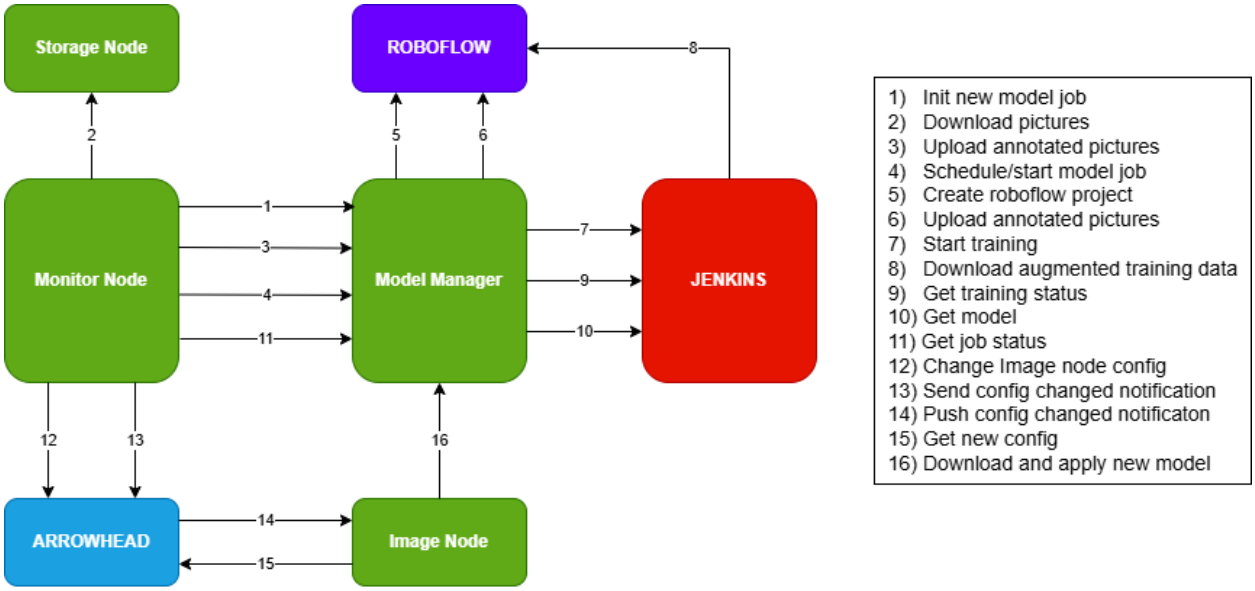


Figure 4: MLOps workflow within SSN based on Arrowhead

The MLOps integration is still in progress, however the following parts of the workflow are already implemented:

- Monitor Node: Picture collection and annotation plus starting the training process.
- Model Manager: All of the planned functionalities.
- Jenkins Pipeline: Prototype implementation with only one training cycle.

In the next period the work will continue with the following tasks:

- Monitor Node: Job status dashboard, model assignment to image nodes and extending/ re-training existing models.
- Image Node: Automatic model deployment.
- Jenkins Pipeline: More robust production ready implementation with multiple training cycles.

On the long term we would like to replace Roboflow (used primarily for augmentation) with an own or an open-source solution.

3.2.2 Software deployment and management for AI functionality

This subtask is focusing on architectural solutions to enable the composability of AI based services in the system. Significant advancements were made in the development and deployment of architectural solutions for AI-based services within the industrial ecosystem. These efforts focused on ensuring the composability and robust management of AI functionalities across various system layers, leveraging the capabilities of the Arrowhead framework.

- **Initial Architectural Design**: The first stage involved defining comprehensive architecture concepts to integrate AI functionalities seamlessly from the edge to the cloud. This setup emphasized the end-to-end data lifecycle, starting with data acquisition, moving through processing at the edge, and extending to secure cloud storage.
- **Edge Data Processing**: The implementation of IoT gateways capable of collecting and processing data in real-time was completed. Devices were equipped with container-based environments (e.g., Docker) to deploy applications such as tool breakage detection, ensuring rapid response times and high data fidelity.
- **Connectivity and Integration**: Using Arrowhead, secure and interoperable connections between various industrial sensors, controllers, and data collection units are enabled. The integration allowed seamless data flow, essential for maintaining operational continuity and effective monitoring.
- **Cloud Storage Configuration**: A scalable and secure cloud storage environment was established, facilitating remote data access and analysis, which marked the completion of the foundational data management infrastructure.

During the second year of the project, the architecture has been realized by integrating and initiating the AI-gym framework, a versatile environment designed for training and deploying AI models with varying latency requirements:

- **Hardware Specification and Deployment**: Significant progress was made in defining and deploying the specific hardware for each architectural tier, moving from conceptual designs to tangible infrastructure:

    - Edge Layer: Standardization settled on Savvy SmartBox units, chosen for their balance of industrial robustness, adequate processing power (Intel Celeron N2930, 8GB RAM, 120GB SSD), essential connectivity (dual GbE), and the ability to run custom firmware. This firmware is critical, providing not only the base operating environment but also embedded microservice orchestration capabilities and dedicated data collection, interoperability, and forwarding logic, making the SmartBox a self-contained edge intelligence hub.



Figure 5: Savvy SmartBox Edge device

- Fog Layer: Dell PowerEdge R660xs servers were deployed to fulfill the Fog role, selected for their server-grade reliability and computational power (Intel Xeon Silver 4410Y 12C/24T, 64GB RAM, RAID 1 SSD storage). Running VMware ESXi 7.0.3 enables efficient resource management through virtualization, allowing multiple Fog services (data aggregation, site-level analytics, Arrowhead core systems, AI-gyms) to run concurrently and securely on robust hardware.

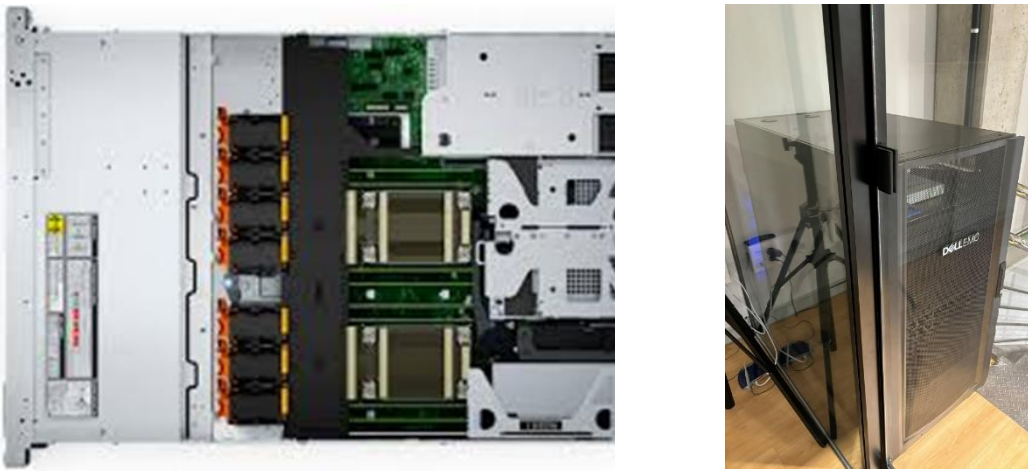


Figure 6: Dell PowerEdge Fog server

- Cloud Layer: The specific use case was defined for Savvy‘s proprietary industrial asset management platform, clarifying its role in handling centralized functions like long-term data archival, computationally intensive large-scale model training, fleet-wide device management, and providing global operational dashboards beyond the scope of a single site. To support intensive model training and validation tasks, a new dedicated server was incorporated into the platform, specifically to host the Cloud-level AI-gym. This server features a 16-core Intel Xeon Gold 5120 CPU (@2.20GHz) and 32GB RAM, providing substantial computational resources for these demanding AI workloads within the cloud environment.

- **Refined Arrowhead Framework Integration**: The initial Arrowhead connectivity from Year 1 has been significantly enhanced to enable robust, multi-cloud architecture.

    - Distinct Arrowhead Local Clouds were fully implemented for Edge, Fog, and Cloud layers, each hosting its own mandatory core systems (Service Registry, Authorization, Orchestrator), ensuring autonomy and localized service management within each tier.

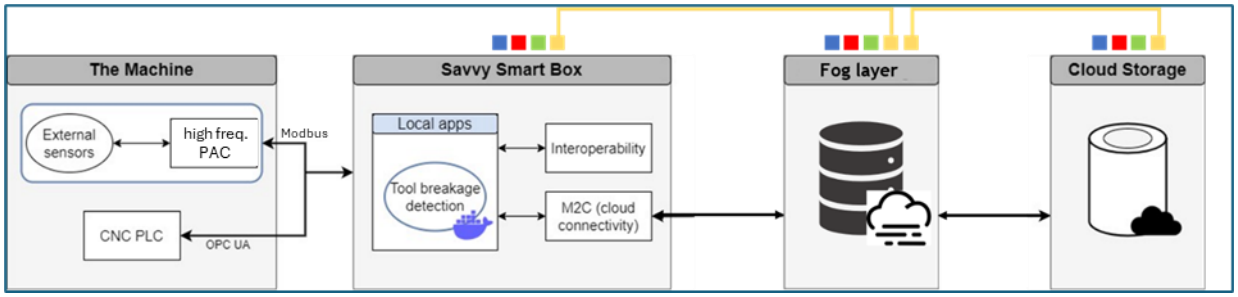


Figure 7: Multi-tier, inter-cloud AH integration

    - Gateway and Gatekeeper services were deployed and configured within each Local Cloud. This has proven crucial for enabling secure, controlled, and discoverable interactions between the different tiers, allowing, for example, an Edge service to securely consume a service offered by the Fog Local Cloud.

    - ActiveMQ-based message relays were integrated into the Fog and Cloud layers, providing a reliable, asynchronous communication channel, and decoupling direct service calls, thus ensuring message persistence during temporary network issues, and potentially bridging different communication protocols used internally within Arrowhead systems.

    - Clear separation of data pathways was established: Arrowhead manages the secure pipeline for ML model deployment, cascading validated models from the Cloud/Fog AI-gyms down to the Edge devices. Conversely, the Savvy SmartBox firmware directly handles the upward flow of raw sensor and process data towards the Fog and Cloud for aggregation and analysis.

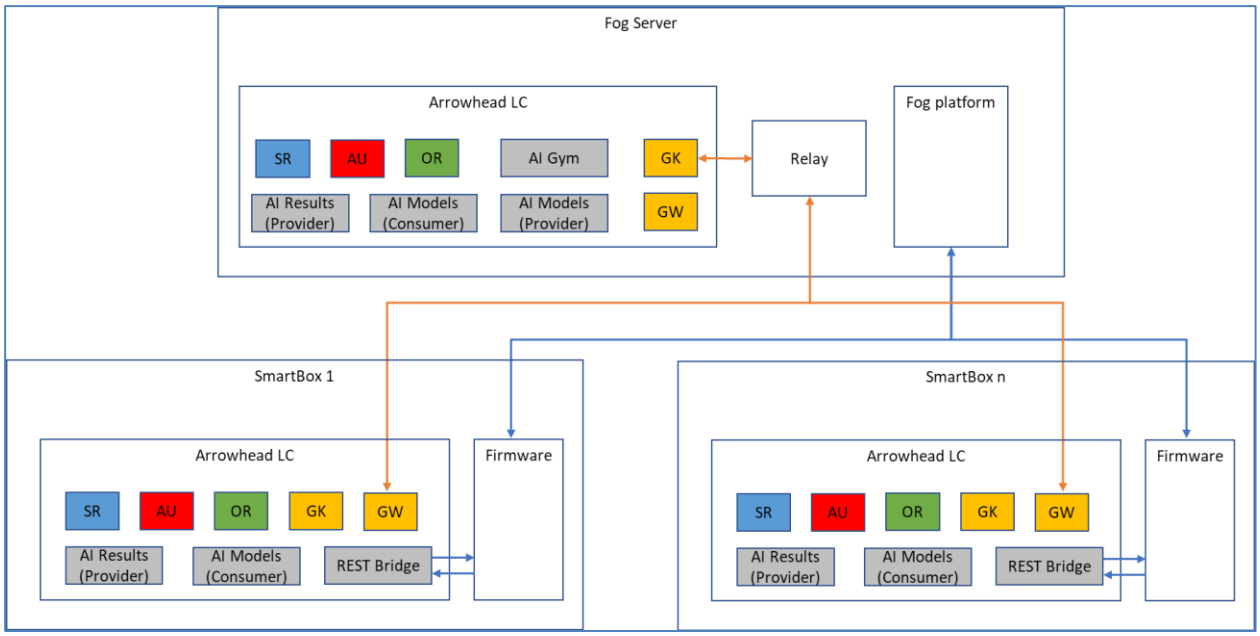


Figure 8: Edge / Fog data & AI pipeline

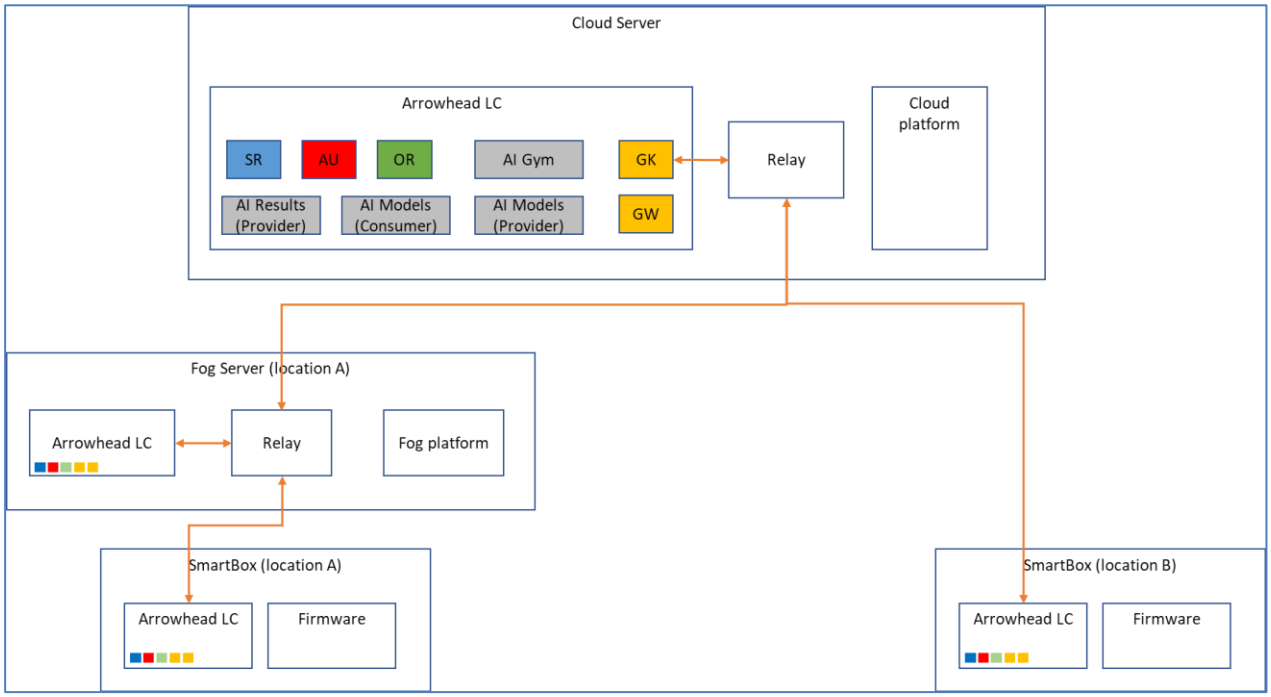


Figure 9: Edge / Fog / Cloud AI pipeline

- Protocol-specific interoperability support AH services have been considered, but eventually disregarded in favor of a generic, REST-based interoperability service that bridges Edge-level AH Local Clouds with Savvy SmartBox firmware, which already incorporates a plethora of mature, industrial communication protocol connectors.

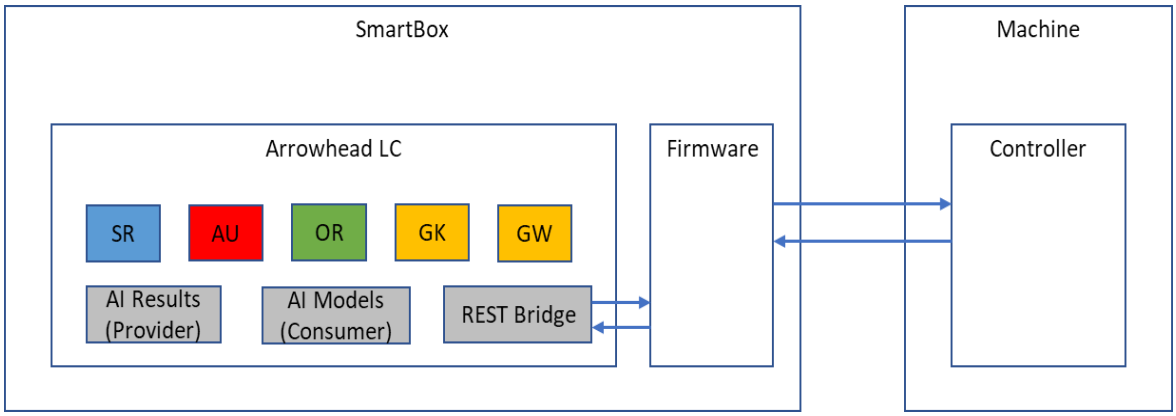


Figure 10: AH Local Cloud interoperability with SmartBox & machine

- **AI/ML Strategy and Edge Capabilities**: The approach to implementing AI/ML, particularly at the edge, was detailed and evaluated against hardware realities.

    - Specific ML techniques (Random Forest, SVM, GBM, CNN) were identified as relevant candidates for edge applications like tool breakage detection (WP5, Use Case 5), based on their algorithmic characteristics and data requirements.

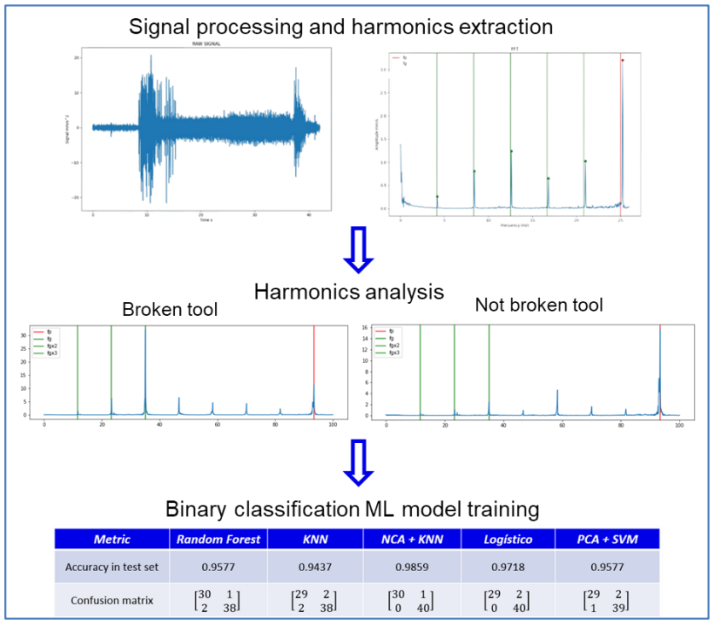


Figure 11: Data collection, processing & model training

    - A thorough analysis was conducted on the feasibility of running these models on the Savvy SmartBox hardware. This highlighted the practical CPU/RAM constraints of the Celeron processor, confirming its suitability for optimized traditional ML models but indicating potential performance bottlenecks for complex CNNs without further acceleration.

    - Investigation and preliminary evaluation of potential AI accelerators were performed. This involved assessing options like the Google Coral TPU (favored

for its efficiency with TensorFlow Lite CNN models), FPGAs (offering flexibility at the cost of complexity), integrated NPUs/VPUs, and low-power GPUs, considering factors like performance boost, power consumption, cost, and development effort to inform future hardware upgrade paths.

- "AI-gyms" were integrated into the Fog and Cloud layers as dedicated environments for model development (hosted on the dedicated Cloud server and within the Fog ESXi environment). Utilizing the "PrototAIP AH adapter," these gyms function as standard Arrowhead services, allowing seamless interaction for submitting training jobs, retrieving validation results, and managing the life cycle of models before their deployment via the Arrowhead pipeline.

- **Testing and Validation**: Testing moved beyond basic functionality to encompass robustness, real-world conditions, and specific failure modes.
  - Rigorous testing protocols were executed in both controlled laboratory settings (for component and integration testing) and more realistic pre-pilot environments designed to closely mimic actual shopfloor conditions (network variability, concurrent operations, legacy system interactions).
  - System robustness was validated through targeted tests, including simulated power outage recovery scenarios for both Savvy SmartBox units and Dell Fog servers. These tests confirmed data integrity on persistent storage, automatic restart of firmware and microservices, and successful reconnection and state synchronization within the Arrowhead framework upon power restoration.
  - End-to-end data flows were meticulously verified, tracing the path of models deploying downwards via Arrowhead and sensor data flowing upwards via the SmartBox firmware, ensuring pipeline integrity under various load conditions.

3.2.3 Safety and security in AI-supported production automation

3.2.3.1 NFC/RFID for security

In AI-supported NFC/RFID label production automation, security is a critical aspect, particularly during the personalization phase where each NFC label, such as those using NTAG chips, is configured with unique access credentials. Personalization involves securely writing data and setting access keys that protect sensitive memory areas, ensuring only authorized entities can read or modify the contents. To enforce this, Secure Access Modules (SAMs) are used to perform cryptographic operations and verify access rights. The CISC NFC Inline test system enhances this process by integrating a built-in SAM card slot, enabling secure and real-time verification of access rights during high-speed inline personalization. This integration ensures that only properly authenticated devices or systems can personalize or read protected NFC tags, maintaining data integrity and trust in automated production environments. AI plays a pivotal role in optimizing this process by intelligently managing personalization workflows, detecting anomalies in tag behavior, and predicting potential security breaches or production faults. Through machine learning algorithms, AI can analyze patterns in large volumes of NFC data,

dynamically adjust process parameters for efficiency, and ensure each label meets stringent security standards, ultimately enhancing both the reliability and scalability of secure NFC label production.

CISC implemented a secure framework for high-volume RAIN and NFC labels production. Using the NFC Inline measurement system with support of the secure access module (SAM), CISC enabled secure personalization and verification of NTAG-based NFC labels at high production speeds. Access keys and unique identifiers are written during inline production, with cryptographic verification handled via the SAM to prevent unauthorized access or cloning. Secure communication from edge to cloud ensures production logs—including tag IDs, encoding parameters, and cryptographic data—are transferred in real time to end-user databases, enabling full traceability and auditability.

The integration of the Secure Access Module (SAM AV3) into the test and measurement system was designed to enable secure authentication and data handling for UCODE DNA tags. This implementation supports key management (write and activation), TAM1 (challenge-response) and TAM2 (challenge-response with encrypted memory return), all controlled via a GUI-based Quality Assurance (QA) interface.

System Architecture

- Core Components:
    - Reader Firmware: Interfaces with the RF front-end and executes EPC Gen2 commands.
    - SAM Module Interface: Communicates with the SAM card using ISO 7816 APDU commands over a serial interface.
    - GUI / QA Application: Provides user configuration for key parameters and executes test sequences.
    - SAM Key Management Engine: Handles key loading, storage referencing (KeyNo, KeyVersion), and activation.
    - Result Logger: Captures output from tag communication and SAM validation for GUI display.
- Data Flow:
    1. User Input → GUI
    2. Command Sequence → QA Engine
    3. Tag Communication ↔ Reader Firmware
    4. SAM Operations ↔ SAM Interface
    5. Response Validation → Result Logger

Writing AES128 Keys to Tags

The test system facilitates writing AES128 keys using the BlockWrite command, targeting specific memory banks of UCODE DNA tags. Key parameters like memory bank, start address, header address, activation word, and key value are set via a user interface. A verification process follows to confirm key correctness before activation. The full write-verify-activate sequence is executed in under 50 ms, demonstrating the system’s real-time capabilities.

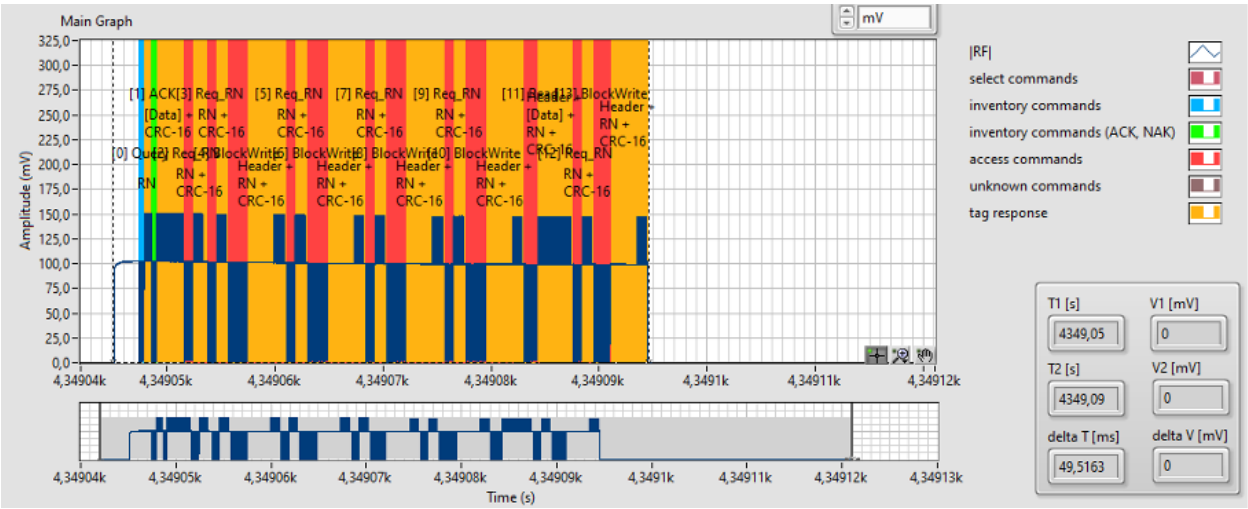


Figure 12: RF sequence - write to tag

The whole sequence takes approximately 49.5 ms. This value can vary, depending on the time for response after BlockWrite command. This tag took around 2.3 ms to respond. This value can be up to 20 ms, according to Gen2 standard.

Tag Authentication using TAM1

For TAM1 authentication, both the tag and the SAM must contain the same key. The system initiates with host authentication using the SAM's master key, retrieves a challenge (IChallenge), and communicates with the tag using this value. The tag's response is validated by the SAM, and results are displayed in the GUI. Authentication failures are logged with detailed error messages, aiding rapid troubleshooting. The entire process, including SAM communication, typically completes in ~662 ms.

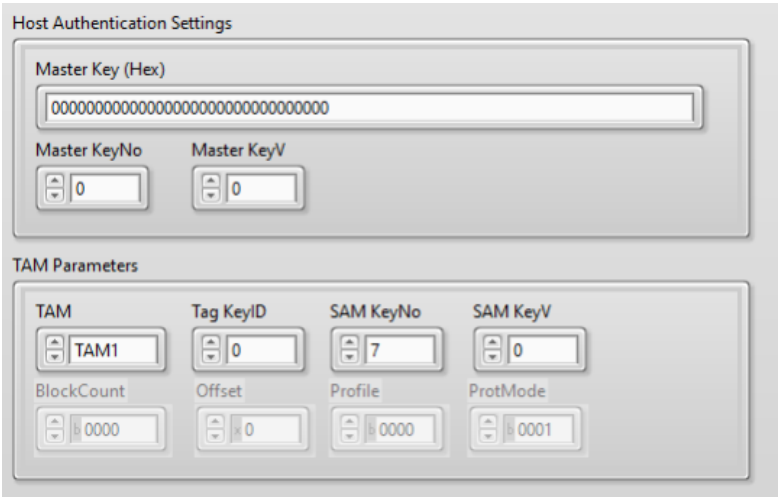


Figure 13: TAM1 settings

The following sequence will be performed by the test system, if host authentication key is valid and host authentication is performed successfully:

1. Retrieve IChallenge from SAM.
2. Send QUE-ACK-REQRN-Authenticate incl. IChallenge from SAM to the tag.
3. Pass tag response to SAM and receive info from SAM if authentication passed or failed.

The RF sequence looks as follows:

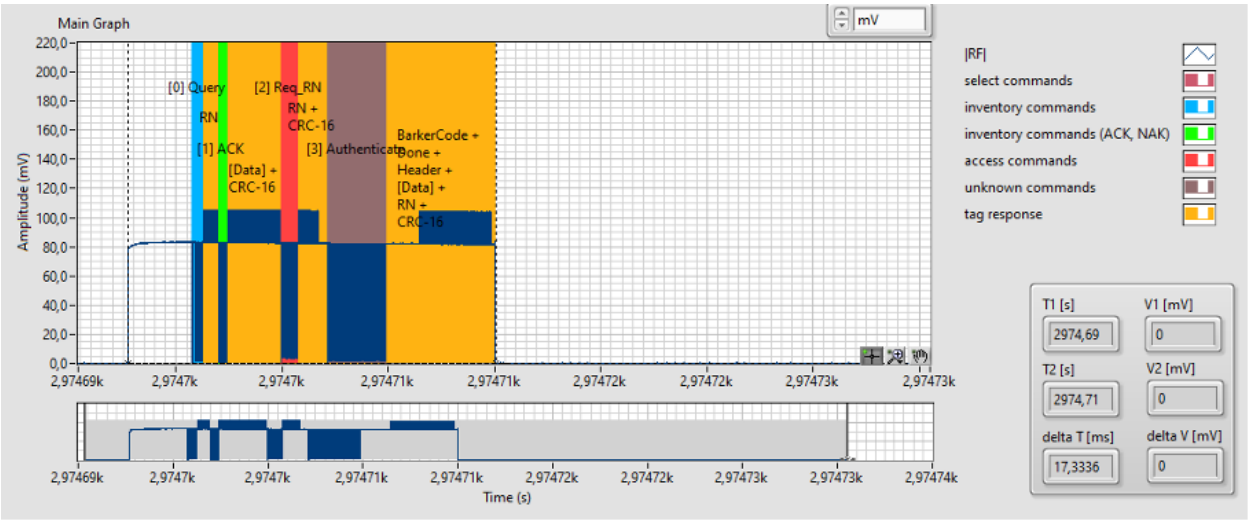


Figure 14: RF sequence of tag authentication 1

The RF sequence itself takes ~**17.5 ms.** Total sequence incl. command exchange with SAM and data processing takes **662 ms.** Note: After every 100 tag, re-authentication with the host is done, to prevent unwanted destroying of SAM due to too many commands, which will add some additional time.

In case of status = fail, additional error will be added in the tag info column. One of the following errors is possible:

- Error SAM: error in the communication with the SAM card occurred, i.e. no IChallenge could be received. Potential issues: wrong host authentication key, unactivated SAM card, key at the defined location is not activated.
- Error Tag Authentication: Valid response received from tag but SAM returned authentication error. Reason: key of SAM and tag do not match.
- Error Tag Response: Invalid response of tag. Issues: Tag does not respond at all, or tag responds to authenticate with error message - key is not activated on tag.

Tag Authentication using TAM2

TAM2 authentication is more advanced, enabling encrypted memory reading. It supports only Key1 in UCODE DNA tags and requires the SAM key to be configured as an OfflineCrypto Key. Due to limitations in the SAM's native TAM2 support, a custom workaround is implemented. This involves:

- Host authentication
- Encrypted challenge exchange
- Decryption of tag response by the SAM

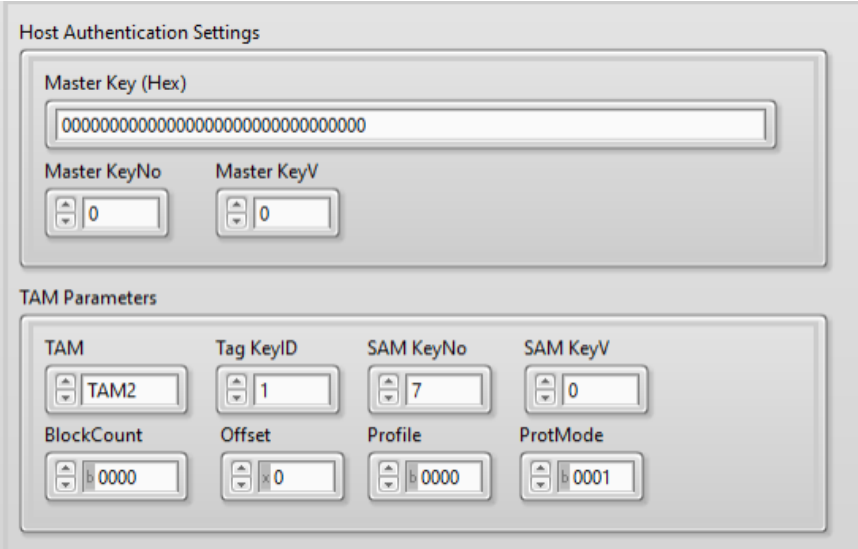


Figure 15: TAM2 settings

The following steps will be taken:

1. Request IChallenge from SAM by requesting random number from SAM.
2. Send IChallenge with TAM configuration by Authenticate command to the tag.
3. Receive Tag Response and pass on to SAM for deciphering.
    a. Set initVector to all 0.
    b. Pass first 16 bytes of TResponse to SAM and compare IChallenge parameter.
    c. Set initVector to first 16 bytes of TResponse.
    d. Pass the next 16 bytes of TResponse to SAM for decryption.
4. Pass tag info to GUI.

The RF sequence looks as follows:

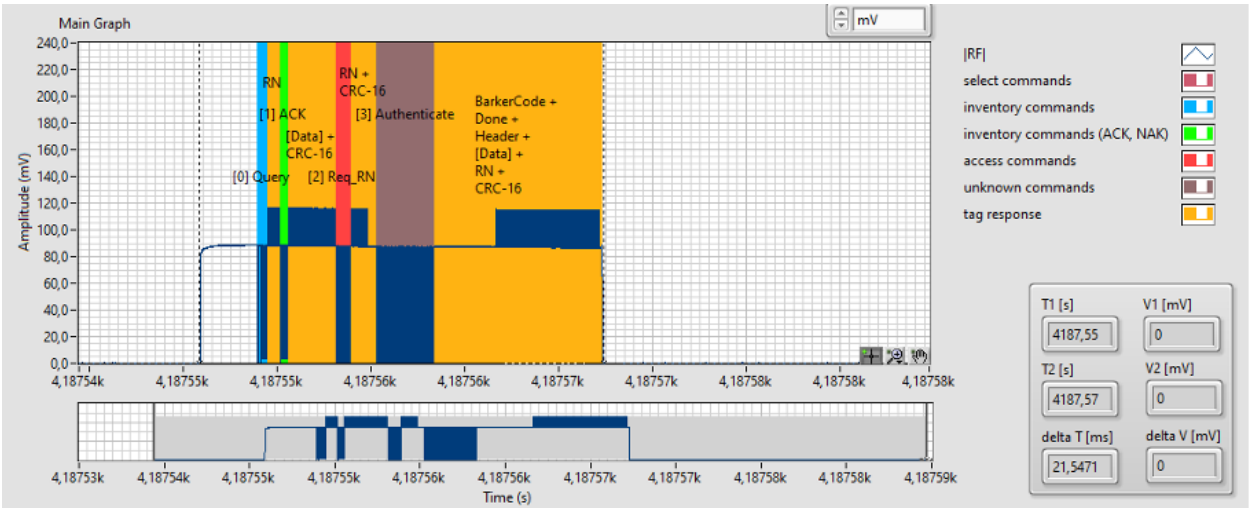


Figure 16: RF sequence of tag authentication 2

This process includes vector initialization and multiple cryptographic steps, typically executed in ~22 ms for RF operations, with total processing time varying based on tag response.

SAM AV3 Configuration

Several predefined configurations are available, including:

- Host keys for authentication
- PICC keys for TAM1
- OfflineCrypto keys for TAM2

Custom configurations for demo or customer-specific setups are outlined, showing flexibility in adapting to use-case requirements.

Quality Assurance GUI Integration

- Users select the desired function (Write Key, Authenticate TAM1/TAM2) from a drop-down
- Fields for keys, addresses, and parameters are prefilled or editable
- Execution is triggered via GUI and results shown in a table format:
    - EPC
    - Key written/read
    - Auth status (pass/fail)
    - Decrypted data (TAM2)

Logging Includes:

- Execution timestamps
- Status per tag
- Error codes

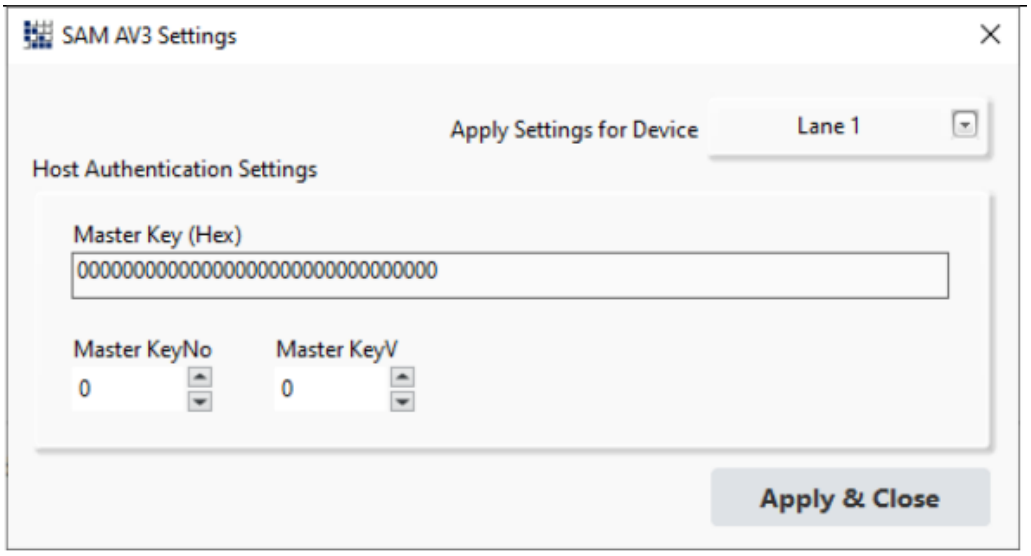


Figure 17: SAM AV3 settings - user interface

Conclusion

The integration of the SAM module into the test and measurement environment delivers robust security capabilities for RFID communication using UCODE DNA tags. With support for AES128 key handling and both TAM1 and TAM2 authentication protocols, the system ensures high-security standards. The SAM integration achieves secure, high-speed tag handling within the test and measurement workflow. The low-level handling of key storage, activation, and

encrypted communication ensures robust and verifiable operations. Despite known limitations in native TAM2 support, the custom fallback implementation maintains secure workflows while enabling encrypted data verification. The modular design allows for future support of additional authentication schemes or hardware versions. Moreover, the intuitive GUI, detailed diagnostics, and fast execution times support practical deployment in diverse operational contexts. This integration exemplifies a modern, secure RFID solution tailored for today's data-sensitive applications.

3.2.3.2 Intelligent Antennas and Secure IoT Architectures

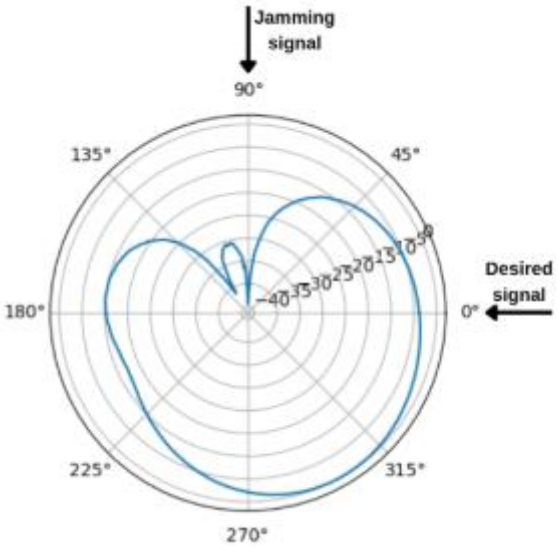
Jamming poses a significant threat to IoT services that depend on wireless communication because it can cause major disruptions to system operations. Interfering modules can degrade performance and cause complete system failures, halting essential services. Additionally, unauthorized wireless devices can disrupt communication between legitimate components, leading to potential malfunctions, data loss, or security vulnerabilities. Therefore, robust protection against interference and unauthorized access is critical to maintaining the reliability and security of wireless IoT systems.

One of the techniques to address the risk of jamming influence is beamforming - a signal processing technique used in antenna arrays to direct radio waves towards a specific direction, enhancing signal strength and reducing interference, Beam switching is a simplified form of beamforming that offers a faster and more cost-effective alternative – in this approach an antenna radio pattern is switched between a finite number of possibilities. One of the examples of beam-switching antenna is ESPAR (Electronically Steered Parasitic Array Radiator) antenna. The GUTs ESPAR antenna has one active element and 12 passive elements that can be switched between two states, providing 2^12 = 4096 different radiation patterns. By traversing 4,096 configurations, the ESPAR antenna can be used to discover the optimal radiation pattern, minimizing jamming,reducing noise interference while maximazing gain towards the signal source. This versatility allows for fine-tuning of the antenna's performance, ensuring effective communication in challenging environments.

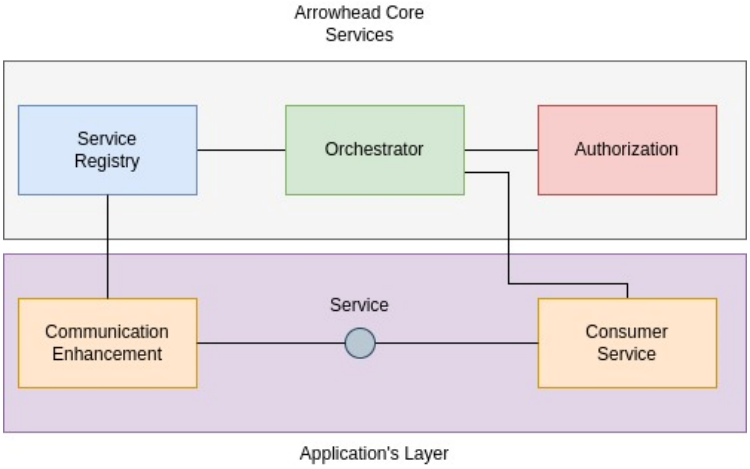


The quality of the communication channel (the parameter to be optimized) can be measured using parameters such as the Received Signal Strength Indicator (RSSI) or Packet Error Rate (PER). RSSI is a value that reflects the strength of the signal received by the antenna in a given

direction. By monitoring the RSSI, the antenna can assess the effectiveness of different radiation patterns in receiving signals from the transmitting source in various orientations. This allows the antenna to identify the strongest reception patterns, corresponding to optimal communication. Packet Error Rate (PER), is the ratio of lost packets to the total number of packets sent. A high PER indicates poor communication quality. When optimizing antenna's radiation pattern, the goal is to increase signal strength and reduce packet loss. Using RSSI and PER as input parameters for AI/ML methods it is possible to intelligently optimize the antenna's configuration. These parameters offer complementary information about signal quality and communication reliability, making them ideal for AI-driven optimization algorithms.



GUT deployed a dockerized instance of the Arrowhead Core Services to facilitate the registration, discovery, and authorization of services exposed by the "Provider" system for use by the "Consumer" system. In this setup, GUT introduced the ESPAR antenna connected with the TTECH Nerve IIoT platform, as an example of IEC 62443 cybersecurity standard compliant device, plays a vital role in enhancing the system's security. The platform serves as the processing unit responsible for data processing and managing communication with the beam-switching antenna. In this setup, the solution acts as a provider and registers a specific service with the Arrowhead Core Services. This service triggers one of the AI-driven antenna adaptation algorithms (developed in WP2), which optimizes the antenna's beam switching based on dynamic conditions. Only authorized consumers (those systems that have successfully passed the authentication and authorization processes) are permitted to interact with the provider and access the registered services. This ensures that communication and service interactions are secure and limited to trusted entities.



3.2.4 Robust real-time networking for AI applications

Enhancing Arrowhead's real-time data handling with TTTech framework's execution control. As detailed in D3.1, TIAG uses Arrowhead's microservice architecture, where self-contained local real-time nodes run the TTTech orchestration framework in the OS. These nodes have real-time execution capabilities, including real-time scheduling and time-sensitive communication.

Some applications on these nodes must meet real-time requirements, such as local deadlines and end-to-end latencies, even across nodes and clouds. Our framework integrates into the OS to control CPU bandwidth for each docker container, ensuring real-time behavior (e.g. minimizing deadline misses) and maximizing utility with time-utility functions. Thus, our framework enhances Arrowhead's real-time communication and execution capabilities on local nodes.

In more details, the real-time orchestration framework developed by TTTech optimizes resource usage during the initial offline phase and maintains real-time performance during runtime by reorchestrating soft real-time containers if service degradation is detected. However, communicating services with hard end-to-end timing requirements necessitate a stricter execution paradigm. Time-triggered computation and communication enables end-to-end latency guarantees of virtualized services with low jitter. Therefore, in the reporting period MX-MY, TTTech investigated an extension to time-triggered schedule synthesis that in addition to task and communication constraints also includes virtualization constraints. The resulting, scheduler can supplement the offline phase of container orchestration by generating schedule tables for containers or virtual CPUs, hosted tasks, and time-triggered communication on a TSN backbone providing end-to-end timing guarantees.

In contrast to the approach described in D3.1, after generating an offline allocation of containers (or virtual machines) to FEC, the presented offline scheduler generates cyclic schedule tables for containers or virtual CPUs, hosted services, and communication on the TSN backbone. On the FECs operating systems with time-triggered scheduling support [Karachatzis2023] ensure exact temporal behavior by dispatching VCPUs, tasks, and TSN streams according to the schedule tables, i.e., there is no reorchestration intended for services with hard real-time requirements.

DREAMS [Ahmadian2018] outlines building blocks for mixed-critical system architectures, focusing on multicore systems-on-a-chip with on- and off-chip networks. A key component is a virtualization layer using the Xtratum hypervisor [Crespo2010] for time-partitioning. Time-triggered VCPU scheduling also found its way into private clouds, where the Linux kernel can utilize a static schedule table to manage VCPUs for TT VMs. Time partitioning is also featured in the PikeOS microvisor. It features a scheduler with sub-schedulers for real-time and non-real-time VMs, where real-time VMs are prioritized [Kaiser2008]. Two white papers [Theiling2013, Motzkus2016] discuss PikeOS's time-triggered operation, which has also been utilized in EU-funded projects integrating TT communication and VMs.

Our work extends task and TSN communication constraints [Craciunas2016-1, Craciunas2016-2] by system virtualization constraints [Ruh2024] enabling us to generate schedule tables for hierarchically scheduled systems. Figure 7 shows an example of a resulting schedule providing end-to-end latency guarantees for two FEC with 2 processor cores, each hosting a total of 8 VMs with 32 services that communicate via a TSN backbone. We can see that nodes 1 and 2 host services communicating with each other via the TSN backbone. Furthermore, we realize that on node 2 core 0 is idle not scheduling any time-triggered tasks.

We propose a Satisfiability Modulo Theories (SMT)-based scheduler to find schedule tables for the hierarchical scheduling problem. SMT determines the satisfiability of first-order logic formulas given a background theory. In the case of our scheduling constraints, we use linear integer arithmetic (Z,+,>=) as the background theory. For a given set of constraints formulated in (Z,+,>=) (for details see [Ruh2024]), an SMT solver yields offsets and sizes determining when and for how long VCPUs, services, and TSN streams execute if a solution exists or reports their unsatisfiability. However, even though SMT concerning (Z,+,>=) is decidable, it is NP-hard, so an efficient algorithm to solve arbitrary sets of constraints is not known. Steiner [Steiner2010] first applied SMT to the scheduling of TTEthernet and proposed an incremental approach, successively adding constraints of a variable-sized subset of frames to the solver's context. If the solver returns a model for a subset of frames, the incremental scheduler fixes the frames in the TT schedule. Otherwise, the scheduler backtraces, increasing the size of the subset of frames. This has shown a substantial runtime improvement in the average case.

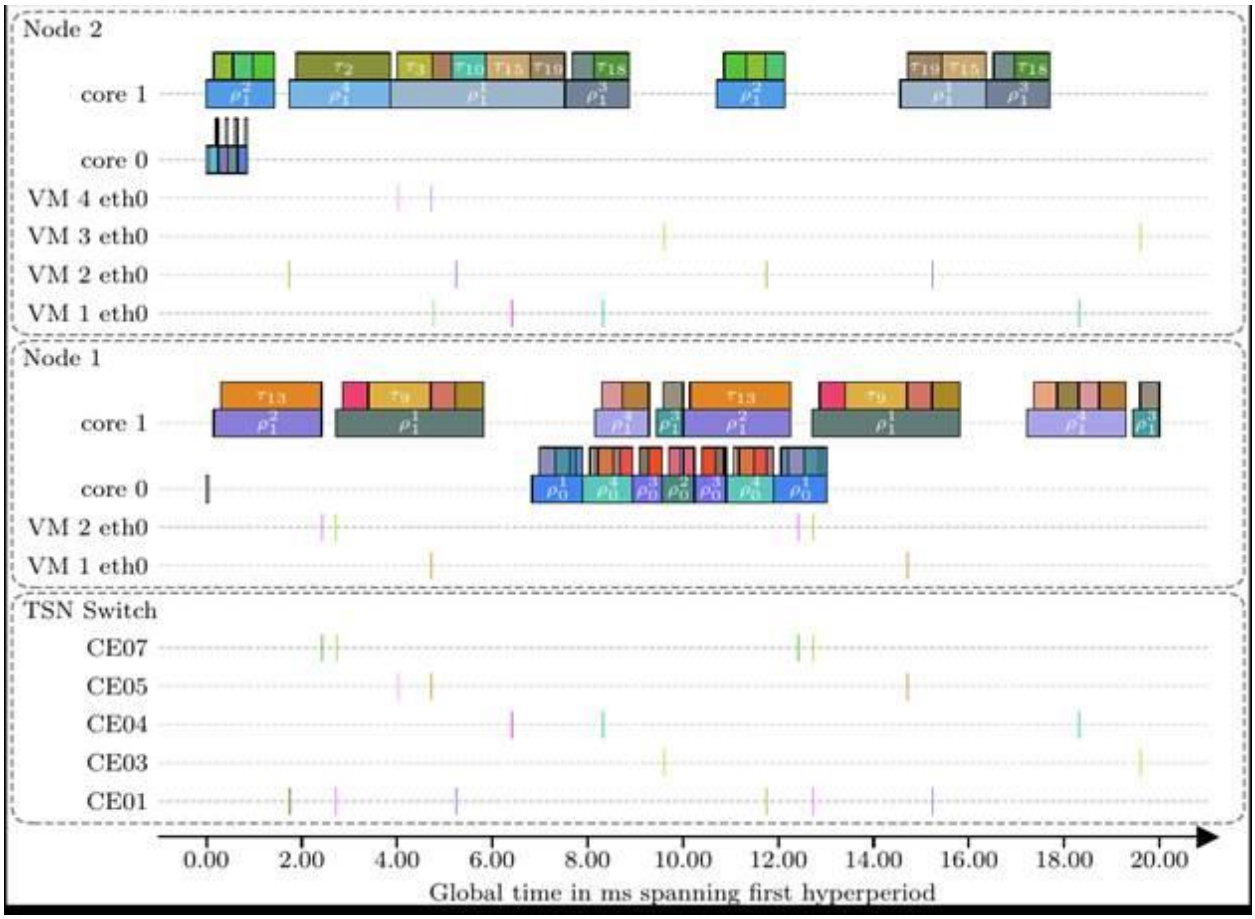


Figure 18: Exemplary Hierarchical Schedule Tables for 2 FEC

We present an incremental SMT-based scheduler (iSMT) that implements VCPU, task, and TSN communication constraints [Ruh2024] and minimizes virtualization overhead, i.e. VCPU context switching time. The scheduler first builds a dependency graph and differentiates between schedulable entity and resource dependencies. A schedulable entity dependency is a bidirectional dependency between two schedulable (VCPUs, services, TSN streams) so that their placement and dimensioning is reciprocal. Therefore, if the scheduler adds a pair of dependent schedulable entities to the SMT context in differing increments, we see excessive backtracking since the placement and dimensioning of the first added schedulable entity does not consider the subsequently added entity's constraints. Once the scheduler has identified tightly coupled schedulable groups that must be scheduled together, it sorts them in descending order by their number of streams, the additive inverse of the utilization without overhead, and the additive inverse of the minimal schedulable period within the group. As a result, the scheduler solves tightly coupled groups with many streams, contributing to high utilization of computational resources while resources are still sparsely populated.

The sorted set of tightly coupled schedulable groups acts as an input to the iSMT scheduler:

- Initialize the internal state: the set of frozen schedulable groups, a head and tail indexing the hot schedulable groups, the current backtracking distance and stack, and 2 dictionaries that for each VCPU hold the number of required segments and the optimization summand, respectively.

- As long as the set of frozen schedulable groups is smaller than the set of hot schedulable groups, the head and tail determine the tightly coupled schedulable groups solved within the same SMT context.
- Each group of tightly coupled schedulables is sorted first by type (service, VCPU, or TSN stream), and then by period, so that within a group high-frequency services are processed first and low-frequency TSN streams last. The scheduler iterates a group's schedulables and generates job segment constraints differentiating services, VCPUs, and TSN streams. The scheduler keeps track of the number of segments generated per VCPU in the dictionary and uses it once the scheduler generates job segments for the VCPU. If a schedulable is a TSN stream, for each network link as part of the TSN backbone we generate frames. Segment and frames are used to generate constraints for the SMT context.
- The scheduler iterates system resources, including CPU cores, VCPUs, and links, using the segments and frames to generate and add resource constraints and check the satisfiability of the resulting SMT context.
- If the solver finds a solution, it optimizes the number of required VCPU segments. There are two possible optimizations. The SMT solver returns VCPU segments not containing any job segments since it grouped job segments sharing a VCPU together, or the SMT solver placed two segments belonging to the same VCPU in direct succession so that the resulting VCPU preemption is dispensable. Therefore, for each VCPU, the scheduler counts the number of empty and consecutive VCPU segments and stores the optimization summand. If the scheduler has detected, it pops the corresponding SMT context and reiterates the same group of tightly coupled schedulables, optimizing the number of VCPU job segments. When the scheduler cannot further decrease the number of VCPU job segments, it pushes the current SMT context on the solver's stack, freezes the found variable assignments, and adjusts the head and tail to index the next hot group of tightly coupled schedulables.
- When the SMT context is unsatisfiable, the scheduler backtracks. If the backtracking distance is zero and there is a head on the backtracking stack, the scheduler restores the head from the backtracking stack and unfreezes the corresponding tightly coupled schedulable groups. If the backtracking distance is greater than 0 or the backtracking stack is empty, but the scheduler can further backtrack, i.e. the head is not at index 0, it decreases the head and proceeds as before with unfreezing the corresponding tightly coupled schedulable groups. If the head decreases to 0, backtracking has included all tightly coupled schedulable groups in a single SMT context, and the solver found no solution, so the scheduler returns an empty set.

We compared the schedulability and runtime of iSMT with a more efficient approach that only utilizes SMT solving to schedule the network and simulated Earliest Deadline First (EDF) scheduling to find segment offsets and sizes of VCPUs and services. We consider 5 system sizes identified by the number of nodes, number of TSN switches, and number of streams. For each system size (#nodes, #switches, #streams) and benchmark, we generated 10 problem instances. We randomly generate between 64 and 128 VMs with a random number of VCPUs for each host node with 4 cores. To generate random task sets per CPU core, we use task characteristics as found in automotive applications identified as TTTech [Finzi2024] and Bosch [Kramer2015].

Figure 8 shows the schedulability as colored area on the left y-axis and the runtime as line on the right, logarithmic y-axis. We can see that iSMT's runtime scales badly with increased system size and target average processor utilization. For the TTTech benchmark, iSMT could only schedule systems up to a size of 2 nodes, 1 switch, and 25

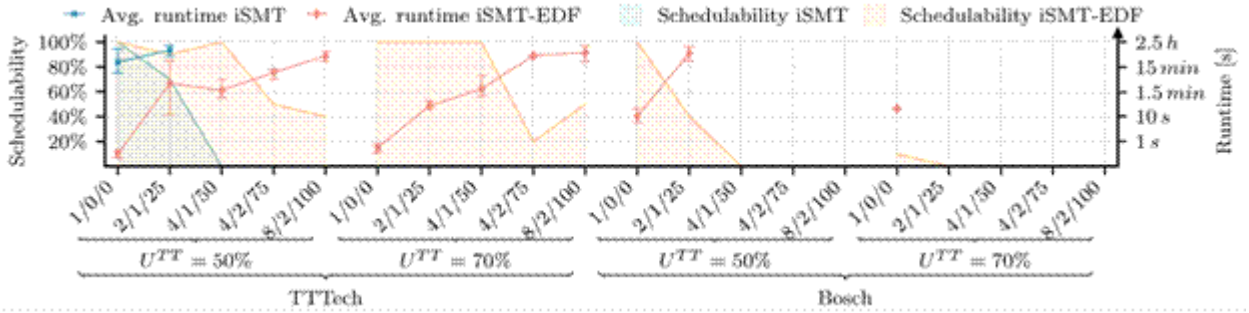


Figure 8. Schedulability with Increasing System Size

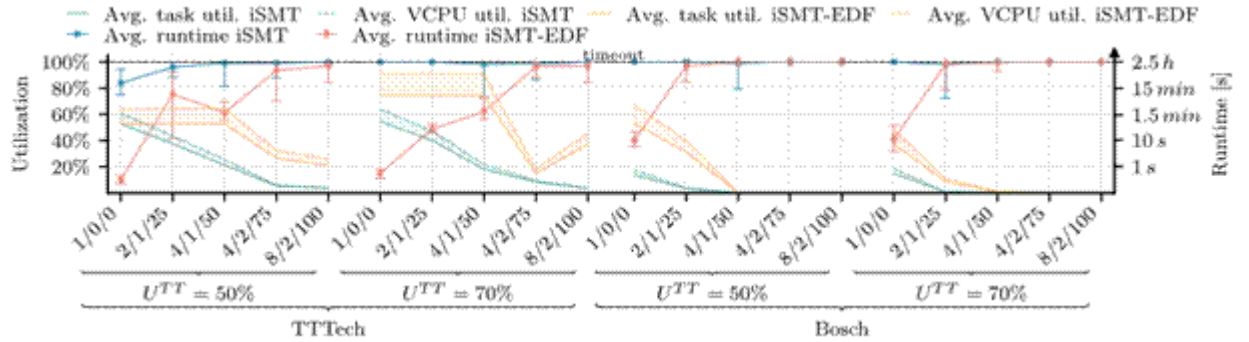


Figure 19: Achieved Target Utilization and VCPU Overhead

TSN streams within the timeout of 2.5h and not a single Bosch benchmark instance. In contrast, for the TTTech benchmark, the EDF scheduler can schedule 4 out of 10 systems of a system size of 8 nodes, 2 switches, and 100 TSN streams. Even for the Bosch benchmark instances, the EDF scheduler generated tables for systems up to a size of (2, 1, 25). Lastly, Figure 9 shows the utilization of tasks and VCPUs scheduled within the timeout for the respective schedulers. The colored area indicates the VCPU context switching overhead. We can see that even though iSMT's runtime does not scale as well as for the EDF scheduler, it successfully reduces VCPU context switching overhead. For a target utilization of 50%, considering only successfully scheduled instances, iSMT yields an average VCPU context-switching overhead of 8.4%. Contrary, the EDF scheduler yields an average VCPU context-switching overhead of 11.7% across all system sizes of the TTTech benchmark for a target utilization of 50% and 15.8% with a target utilization of 70%. For the EDF scheduler and the Bosch benchmarks, we find an average VCPU context-switching overhead of 14.3% for the schedulable problem instances with a system size of 1/0/0 and 2/1/25 and with a target utilization set of 50%.

These results were presented at the 32nd International Conference on Real-Time Networks and Systems (RTNS 2024) in Porto.

AI-driven tool for understanding timing diagram

In cooperation with TIAG, TU Wien has worked on enhancing the understanding of timing diagrams with visual-language learning. Digital hardware design is a complex process that

involves translating high-level requirements into register-transfer level implementations using hardware description languages such as Verilog and VHDL. Although modern tools, such as High-Level Synthesis methods and pee-verified intellectual property (IP) cores, have reduced the time and complexity involved, engineers must still maintain a deep understanding of the behavior and interaction of individual hardware components. Timing diagrams (TDs), which visually represent signal transitions over time, play a central role in the design and verification processes. A solid understanding of TDs is therefore essential across many phases of digital system development.

To support engineers in interpreting timing diagrams, we introduce TD-Interpreter, an AI-driven tool capable of understanding and explaining TDs in a variety of contexts. This project investigates the use of multi-modal large language models (MLLMs), which integrate visual and linguistic reasoning capabilities, to build a visual question answering (VQA) pipeline tailored to the structure and semantics of timing diagrams. The system is designed to extract precise signal-level information and provide accurate, context-aware answers to engineering-related queries.

TD-Interpreter is built upon recent advances in MLLMs, which have demonstrated strong performance across a range of tasks including image captioning, visual reasoning, and text-to-image generation. However, applying commercial models like GPT-4o directly to timing diagram interpretation presents several challenges. There are privacy concerns related to a company's IP, there are limitations to fine-tuning commercial models, and there is a lack of domain-specific training data. To address these challenges, our approach emphasizes low-cost, transparent solutions for dataset creation, model training, and local deployment.

First, we solve the problem of dataset acquisition. Since no public dataset of annotated timing diagrams is available, we developed a synthetic data generation pipeline. This system takes publicly available Verilog models, automatically generates testbenches with randomized inputs, simulates them, and converts the results into Wavedrom-format timing diagrams. These diagrams are then automatically annotated using the information from the simulation results, yielding detailed ground truth, such as the sequence of values for a given signal s, or the value of s at a given clock cycle t. In addition to synthetic data, we manually created more challenging, reasoning-based diagrams using HDLBits problems as well as classical digital design tasks such as designing synchronous and asynchronous FIFOs and implementing communication protocols like AXI and AHB.

Second, we solve the fine-tuning problem by successfully fine-tuning LLaVA, a lightweight 7B MLLM, using a subset of our dataset. Initial experimentation has shown promising results in handling VQA tasks related to timing diagram interpretation. Building on this we are currently exploring VLM-R1, a state-of-the-art vision-language model built on DeepSeek-R1. DeepSeek-R1 has achieved benchmark results comparable to OpenAI's GPT-4 (o1) in advanced reasoning based tasks. VLM-R1 extends this to the visual domain and has shown strong performance in mathematical reasoning with under 4B parameters, making it a compelling candidate for further development of TD-Interpreter.

4. The AI-gym - AI functionality as services

The **AI-Gym** framework integrates state-of-the-art tools such as **Apache Zeppelin** and **Eclipse Arrowhead**, providing a structured environment for AI model development, testing, and deployment. We detail below the three-layer architecture of the **AI-Gym** and its role in facilitating seamless integration within **Industry 4.0** ecosystems.

The below described AI-gym is partly modeled in SysML using the Arrowhead DSL. Sufficient details are still pending to enable a more complete modeling of the AI-gym. With the intention to facilitate uptake of the AI-gym technology already at a conceptual modeling design stage.

1. AI-Gym Architecture

The **AI-Gym** is built on a three-layer architecture, see Figure 10, ensuring a smooth transition from AI development to IIoT deployment.

1.1 Model Development Layer – Apache Zeppelin

This layer serves as the foundation for AI model creation and experimentation. Apache Zeppelin provides a notebook-style interface, supporting multiple languages such as Python, Java, and Scald. Key features include:

- Interactive development of machine learning models.
- Real-time visualization of data insights.
- Support for various ML frameworks, enhancing adaptability.
- Developers can iteratively refine AI models based on live feedback, optimizing their performance before moving to the integration stage.

1.2 Integration Layer – Eclipse Arrowhead Adapter

This layer facilitates the secure integration of AI models into IIoT environments via the Eclipse Arrowhead service-oriented architecture. Key functionalities include:

- Service registration: AI models developed in Zeppelin are registered as services within the Arrowhead Service Registry.
- Authorization management: Ensuring only trusted entities access IIoT data.
- Data flow orchestration: Enabling structured interaction between AI models and IIoT components through the Event Handler.

1.3 IIoT Data Layer – Eclipse Arrowhead Local Cloud

The final layer enables AI models to be tested with real-world IIoT data streams, simulating actual deployment environments. This ensures:

- Scalability: The models can handle dynamic industrial data.
- Security: Only authorized services interact with IIoT devices.
- Reliability: Performance is assessed under diverse conditions before deployment.

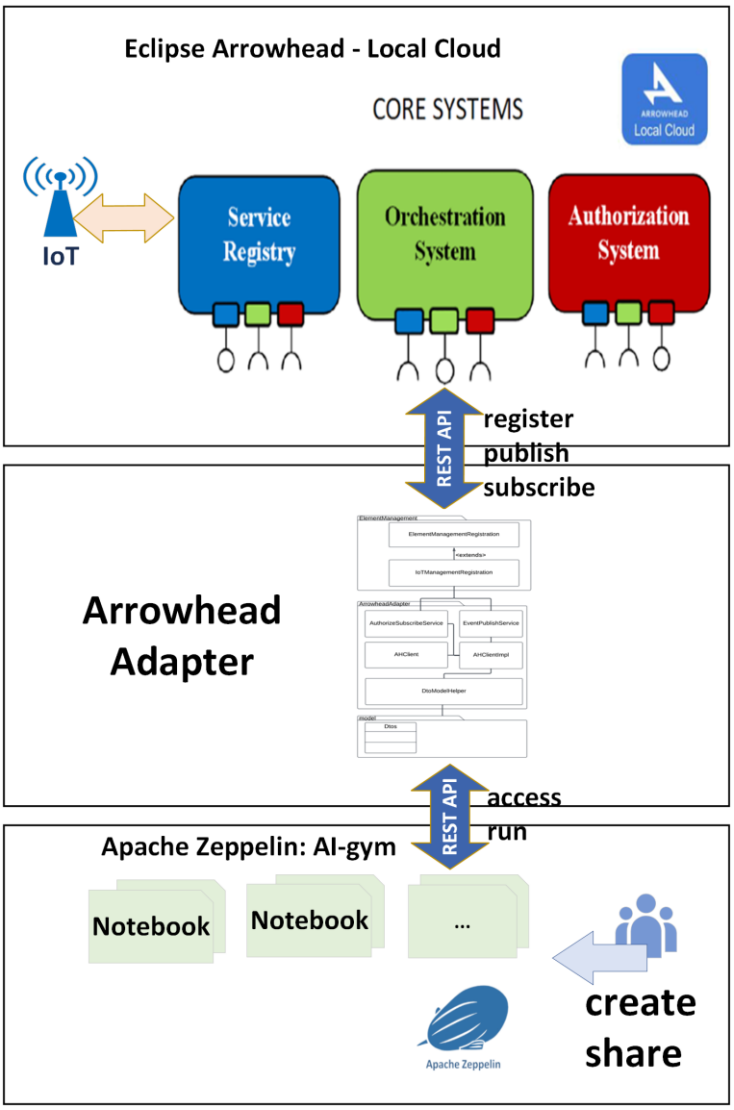


Figure 20: The three layer architecture of the AI-gym

2. Integration Workflow

The integration of **Apache Zeppelin** and **Eclipse Arrowhead** follows a structured workflow:

2.1 Service Registration

Zeppelin notebooks are exposed as services in the **Arrowhead Service Registry** via a REST API. Each service includes metadata such as service definition, endpoint URI, and capabilities.

2.2 Authorization Handling

Before accessing IIoT data streams, AI models must obtain authorization from the **Arrowhead Authorization Service**, ensuring compliance with security standards.

2.3 Data Exchange via Publish/Subscribe Model

The **Arrowhead Event Handler** facilitates real-time data exchange between AI models and IIoT devices through a publish/subscribe mechanism. This ensures seamless communication for sensor readings and actuator controls.

3. Implementation and Validation

The AI-Gym framework was implemented and validated using **UC11**: *AI-supported Industrial IoT for Indoor Food Production*.

3.1 Use Case 11:

The AI-Gym was deployed in a greenhouse environment equipped with temperature and humidity sensors, as well as fans, see Figure 11. Key components include:

- Apache Zeppelin: Hosting a fine-tuned LLM-based predictive model for climate control.
- Eclipse Arrowhead: Handling secure data exchange and service orchestration.
- IIoT Sensors & Actuators: Providing real-time environmental data.



Figure 21: Use Case 11 Greenhouse Environment

3.2 Fine-Tuning of LLM Model

A lightweight GPT-2 model was fine-tuned using structured datasets for temperature prediction. The model was:

- Trained on historical greenhouse data.
- Exposed to real-time IIoT data streams.
- Integrated with the AI-Gym to dynamically adjust temperature settings.

5. Standardization-based feedback and guidelines

The AIMS5.0 WP3 cyber architecture is developed in a context that integrates digital industrial systems, real-world IoT deployments, and microservice-based automation with Eclipse Arrowhead. The framework supports composability, secure deployment, and orchestration of services across heterogeneous devices and networks. In this context, the architectural work is guided by a number of relevant industrial cybersecurity and AI-related standards, regulatory frameworks, and reference architectures applicable to automation, digitalization, and Industry 4.0 systems. New considerations are the NIS2 and the CRA, which are very recent additions to the European regulatory landscape, with deadlines for conformity in the case of the CRA until 2027.

5.1 Cybersecurity Standards and Frameworks for Industrial Automation

Given the deployment of Arrowhead in complex, heterogeneous environments—including OT/IT convergence, edge-to-cloud data flows, and interoperability layers—the following standards and frameworks are most applicable:

- **IEC 62443 Series – Industrial Automation and Control Systems Security**
  This series provides a comprehensive framework for securing industrial systems. Key parts relevant to AIMS5.0 include:
  - *IEC 62443-1-1 to 1-3*: Terminology and foundational concepts for zones, conduits, and security levels.
  - *IEC 62443-3-3*: System-level security requirements, covering secure communication, authentication, access control, and availability—directly applicable to Arrowhead's SOA-based orchestration and deployment strategies.
  - *IEC 62443-4-2*: Component-level security requirements, relevant for core microsystems and edge device integration (e.g. secure onboarding and runtime communication in Arrowhead deployments).
- **ISO/IEC 27001:2022 – Information Security Management Systems**
  Relevant to the governance of distributed cyber-physical infrastructures like Arrowhead Local Clouds and federated systems. Especially applicable where secure system configuration, access control, and incident handling must be standardized.
- **NIS2 Directive (EU) – Network and Information Security**
  NIS2 increases cybersecurity requirements for operators of essential and important services, including industrial IoT, critical manufacturing, and digital service providers. Many AIMS5.0 components—such as automated deployment, real-time networking, and secure service orchestration—align with these resilience goals.
- **Cyber Resilience Act (CRA)**
  The CRA introduces mandatory cybersecurity-by-design and vulnerability management obligations for connected products. Given the microservice and SoS architecture of Eclipse Arrowhead (especially in context of secure update mechanisms and robust deployment of AI components), the CRA is highly relevant for the implementation side of the platform.

- **ETSI EN 303 645 – Cyber Security for Consumer IoT (adaptable for IIoT)**
  While aimed at consumer IoT, many of the guidelines such as secure boot, update mechanisms, and exposure minimization apply to IIoT devices and edge nodes used in the AIMS5.0 pilots.
- **RAMI 4.0 – Reference Architectural Model for Industry 4.0**
  Arrowhead aligns well with the RAMI 4.0 model, acting as a practical implementation of service-based communication and orchestration. RAMI 4.0 does not define security measures per se, but acts as a structuring reference for aligning systems, assets, and functions across the value chain.

5.2 Standards and Guidelines for Trustworthy AI Integration

AIMS5.0 integrates AI capabilities deeply into industrial workflows (e.g., model deployment at the edge, MLOps orchestration, data acquisition and monitoring), which makes alignment with emerging AI trust and safety frameworks critical. The following standards and documents provide essential guidance:

- **ISO/IEC 42001:2023 – AI Management System Standard (AIMS)**
  Provides a structured management system approach to govern AI capabilities across their lifecycle. Applicable to the AI-Gym, Sentinel system, and MLOps workflows by guiding risk-aware deployment, monitoring, and model updating.
- **ISO/IEC TR 24028:2020 – Trustworthiness in AI**
  Addresses the core attributes of trust in AI systems: robustness, safety, security, privacy, and explainability. This is especially relevant for the integration of AI models into edge/fog/cloud architectures and the verification mechanisms (e.g. formal models in Dezyne, SysML).
- **ISO/IEC TR 24029-1:2021 – AI Robustness Testing**
  Offers methodologies for evaluating the robustness of AI systems, relevant for hallucination detection and mitigation in the Sentinel use case, as well as for testing edge-deployed AI services under variable operating conditions.
- **EU AI Act (Draft Regulation)**
  AIMS5.0 use cases such as visual surveillance, predictive control, and automated adaptation may be classified as high-risk under the AI Act. The Act mandates conformity assessments, traceability of models, dataset quality, human oversight, and cybersecurity of AI systems—particularly relevant for AI-Gym and automated MLOps workflows.
- **OECD AI Principles / G7 Hiroshima AI Process**
  Provide non-binding but globally recognized principles that align with the AIMS5.0 goals, particularly transparency, safety, robustness, and accountability.

5.3 Standardization Feedback Loop

AIMS5.0, actively contributes to ongoing standardization initiatives. This includes:

- Participating in working groups shaping future industrial cybersecurity and AI standards.
- Documenting conformance and gap analysis against IEC 62443 and CRA principles as the platform matures toward Arrowhead v5.0.

This ongoing feedback loop between implementation and standardization ensures that AIMS5.0 outcomes are not only practically validated but also help inform emerging best practices in secure and trustworthy AI-based industrial systems.

CISC contribution to standardization

CISC Semiconductor GmbH plays an active and influential role in the standardization of quality assurance within the RAIN RFID ecosystem. As a co-chair of the RAIN Alliance Manufacturing Quality Certification (MQC) Standardization Working Group, Dr. Vojtech Derbek, Vice President of RFID Engineering at CISC, contributes directly to the development of standardized processes and criteria aimed at ensuring consistent manufacturing quality across RAIN RFID products. CISC provides both strategic leadership and technical expertise to the working group. The company leverages its extensive experience in RFID system development, particularly through its RFID Xplorer product suite, which is widely utilized for the evaluation and verification of RAIN and NFC tag performance. These tools, developed under CISC, serve as a foundational basis for performance measurement and quality assurance, offering practical insights that inform the standardization process.

CISC’s contributions to the MQC initiative are rooted in its commitment to fostering interoperability, reliability, and global harmonization of RFID technologies. By aligning its testing methodologies with emerging certification requirements, CISC ensures that its solutions remain compliant with and supportive of industry standards. Furthermore, CISC acts as a bridge between industry practice and standardization efforts, promoting the integration of scientifically validated methods and real-world applicability. Through its involvement in the MQC Working Group, the company actively supports the development of certification frameworks that will enhance quality consistency, facilitate market acceptance, and strengthen stakeholder confidence in RAIN RFID technologies.

6. Conclusions

In conclusion WP3 has a good progression in line with the description of the action. A concrete focus for the last year of AIMS5.0 will be on:

- Cohesive integration of the different developments into the Eclipse Arrowhead technology stack
- Verification of the cyber architecture in the use cases
- Analysis of how the architecture do address the stated requirements

7. Appendix

7.1 Appendix Requirements 3.2 (PUB)

7.2 Appendix Requirements 3.5 (SEN)