Grant Agreement Number:	957204 (H2020-ICT-38-2020)
Project Acronym:	MAS4AI
Project Start Date:	1st October 2020
Project Full Title:	Multi-Agent Systems for Pervasive Artificial Intelligence for assisting Humans in Modular Production

D6.1 – Smart Factory Testbed Setup – Initial Results

Dissemination level:	PU
Date:	2022-03-31
Deliverable leader:	DFKI
Contributors:	DFKI
Reviewers:	LMS
Type:	R
WP / Task responsible:	DFKI
Keywords:	SmartFactoryKL, prototype, CPPM, Basyx, AAS

Executive Summary

This document comprises the deliverable D6.1 that describes the initial results of the prototypical setup on the Multi-Agent System based on the Janus SARL runtime as MAS and BaSyx as a middleware for hosting AASs, in the Smart Factory testbed demonstrator. The goal is to implement the architectural framework of MAS4AI in this testbed and to evaluate initial results, which can be used in the work packages WP2 – WP5 of the project and for other use case implementations in WP6 of the MAS4AI project.

At first, the scope and the hardware and software setup of the Smart Factory testbed demonstrator is described and requirements and method for the prototypical implementation are presented. The results of the iterative MAS4AI prototypical implementation, following the proposed method, are shown and the results of each iteration are shown. At least, the technical results of the prototypical implementation of the MAS4AI framework in the testbed environment are described.

Document History			
Version	**Date**	**Contributors**	**Description**
V01	2022-01-18	DFKI	Initial Version of the document
V02	2022-03-10	DFKI	Creation of all sections and content
V03	2022-04-12	DFKI	Final Version

Table of Contents

Table of Figures

1 Introduction

The main objective of the SmartFactory-KL Use Case follows the general objectives of WP6, to prove and validate the MAS4AI approach in industrial test beds. The modular Industrie 4.0 testbed of the Technology Initiative SmartFactory KL provides the possibility to build up, test and deploy the MAS approach combined with the prototypical development and implementation of defined agents with their requirements from WP 1.

The SmartFactory KL testbed provides the basis for the first functional prototype which is based on the MAS4AI architecture concept and consist of a modular production environment and the integration of message-based middleware solutions like OPC UA or the BaSyx-Middleware. The usage of several software components like discovery and registry services, which are essential in the testbed architecture, as well as the agent development and their configuration and execution is also considered in the context of the MAS deployment.

The development of the first prototypical MAS4AI implementation, which is documented in this document, follows several iterations. With each iteration, new components and features have been considered for interacting in the used MAS framework. The developments in the MAS framework Janus SARL, which is applied to the Smart Factory testbed to meet the requirements along predefined agent patterns, and the connection to physical hardware component for control purpose are presented as results of the first prototypical implementation.

2 Scope of Smart Factory testbed evaluation

2.1 Use Case Description

The SmartFactory-KL production environment consist of several demonstrator testbeds, in which the Production Level 4 (PL4) demonstrator of the DFKI has been developed and build up as modular skill-based factory testbed for modern automation hardware and software, as well as concepts.

The PL4 demonstrator as Cyber-Physical Production System (CPPS) consist of four Cyber-Physical Production Modules (CPPM) which encapsulate their capabilities to execute a defined production step with their own implemented skill interface. Each CPPM and the takeout place to receive the manufactured product is linked to the central rail system, which transports the workpiece carrier. The demonstrator environment is based on a service-oriented architecture [1] and can reconfigure the setup of CPPM with the plug-and-produce capability.

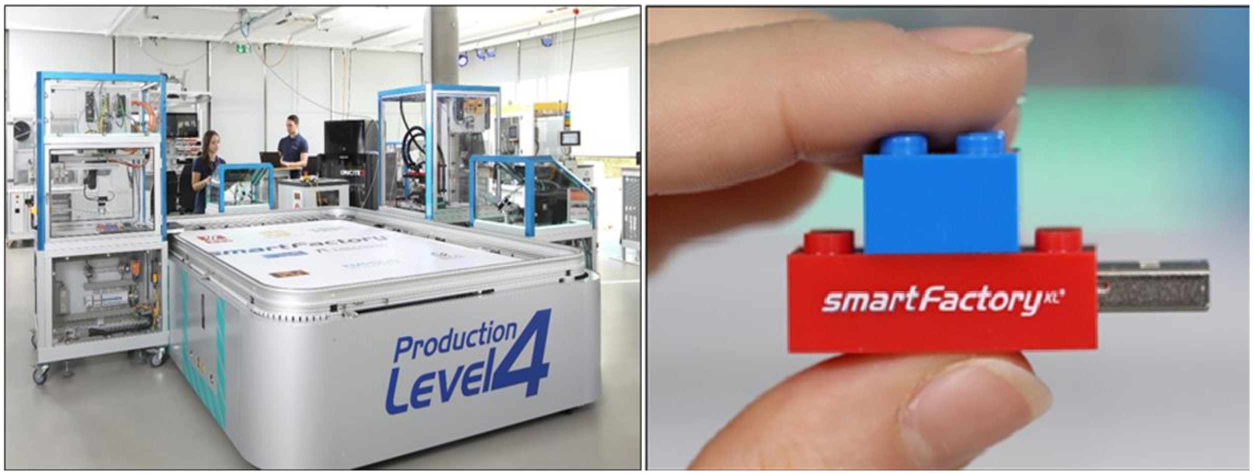


Figure 1: SmartFactory PL4 demonstrator with product

The modular setup provides a basis for distributed deployment for control and data collection in production environments with physical and virtual software assets, which should be integrated in scope of MAS4AI system validation. In the PL4 demonstrator, the product to be manufactured is modular as well and consist of several parts, which can be combined and ordered individually. The SmartFactory PL4 demonstrator with product is shown in Figure 1.

The production process of the PL4 demonstrator to produce this product follows the sequence as shown in table 1. The processed sequence of the corresponding CPPM of the PL4 demonstrator is also described with relation to the production step. The several CPPM of the demonstrator, where each module is responsible to execute one step in the production process, is shown in Figure 2.

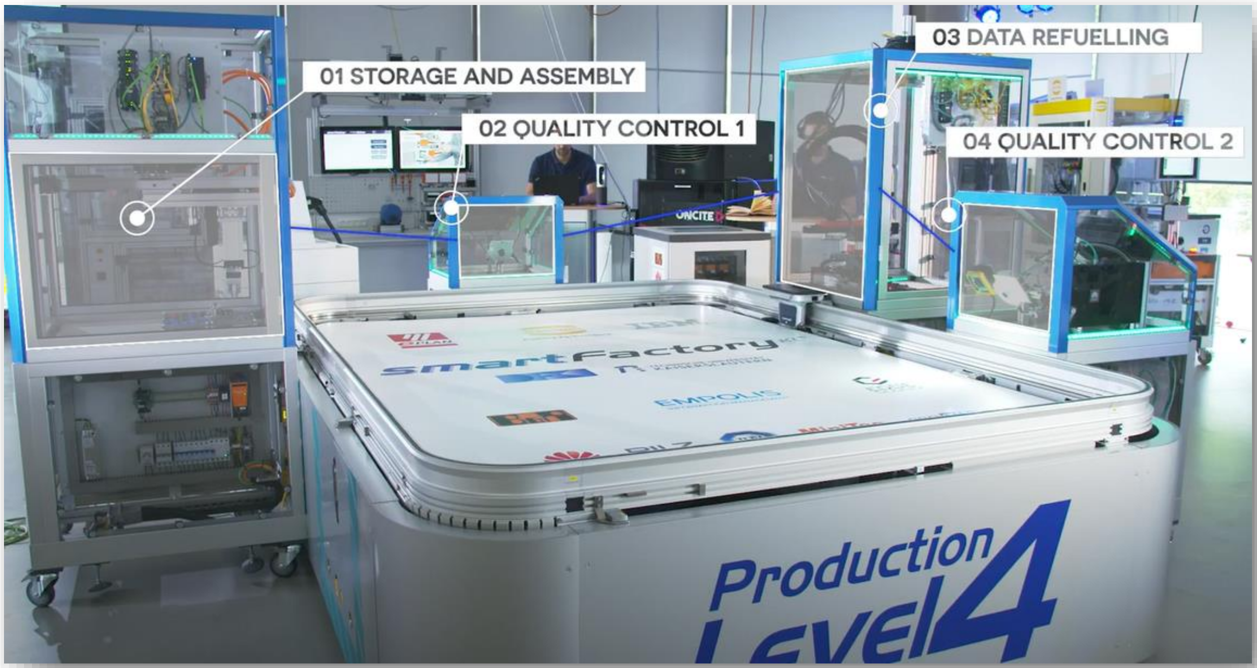


Figure 2: Smart Factory KL – Process steps of the PL4 demonstrator

Process step	Short Description
Step 1	Module 1: "StorageAndAssembly" Get the workpiece carrier with a pick and place from the current shuttle of the rail system into the production module. Pick a nubstone (in the ordered color) from the storage element and place (assemble) it on top of the base nubstone USB stick and set it onto the workpiece carrier. Set the workpiece carrier with a pick and place from the current module back to the shuttle of the rail system.
Step 2 (optional)	Module 2: "Quality 1" Image-based quality inspection of the assembled part. The image of the quality check is stored and uploaded for the following production module for data storage purpose.
Step 3	Module 3: "Data Refuel" Data-flashing of ordered data and the picture of the quality check module onto the nubstone USB.
Step 4	Module 4: "Quality 2" Final Image-based quality inspection of the product.
Step 5	"Takeout Place" Module to get the final manufactured product out of the production line.

Table 1: Process steps and related resources of the PL4 demonstrator

2.2 SmartFactory CPPM and Hardware Basis

In this chapter, the Cyber-Physical Production Modules with listed hardware assets, which are also considered in the requirements document, are shown. Each CPPM is presented with an own figure and the related table for each module describes the hardware related components, the module consists of.

Module 1: "StorageAndAssembly"

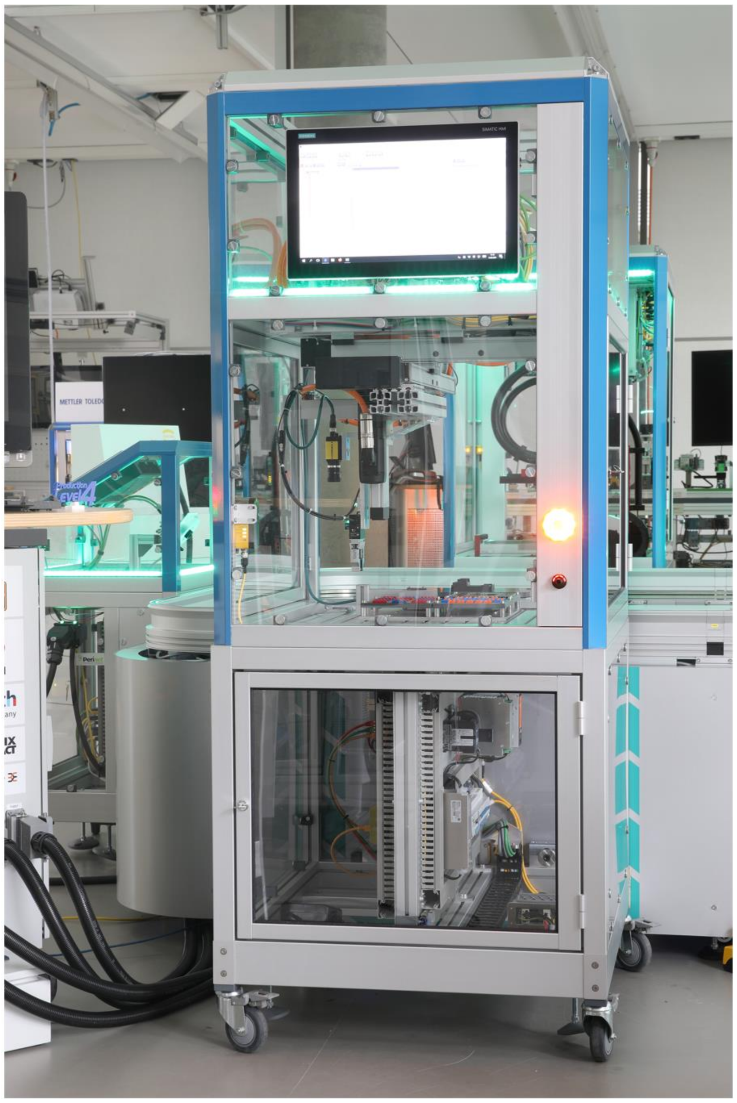


Figure 3: Module "StorageAndAssembly" (Picture by A. Sell)

Type	Name	Manufacturer
Micro Controller	Arduino Nano	Arduino
Sensor (Camera)	IS8401C-363-50	Cognex
Actor (Gripper)	EHPS-16-A-LK	Festo
Switch	Ha-VIS eCon 3070GB-A-PP	Harting
Switch	Eagle 30 Industrial Security Router	Hirschmann
Sensor (Camera)	ISR-751	Keyence
Sensor (RFID Reader)	IQT1-F61-IO-V1	Pepperl & Fuchs
Actor	PSEN sl-1.0p 1.1 / PSEN sl-1.0fm	PILZ
Sensor	PSEN cs1.1p	PILZ
Linear Axis Gripper	Linear Axis Gripper (summarized)	Rexroth
PLC	SIMATIC ET 200SP Open Controller, CPU	Siemens
PLC	PRO MAX3 240W 24V 10A	Weidmüller
Touchpanel PC	SIMATIC IPC477E	Siemens

Table 2: Hardware components of module 1

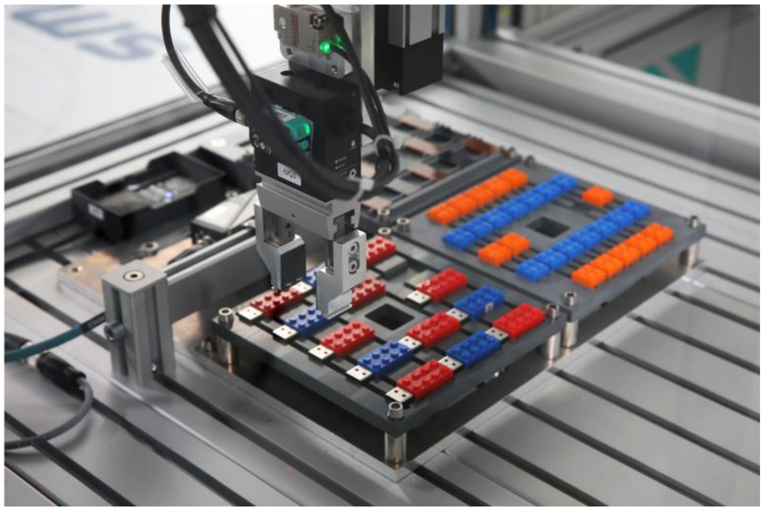


Figure 4: Module "StorageAndAssembly" (Picture by A. Sell)

Module 2: “Quality 1”

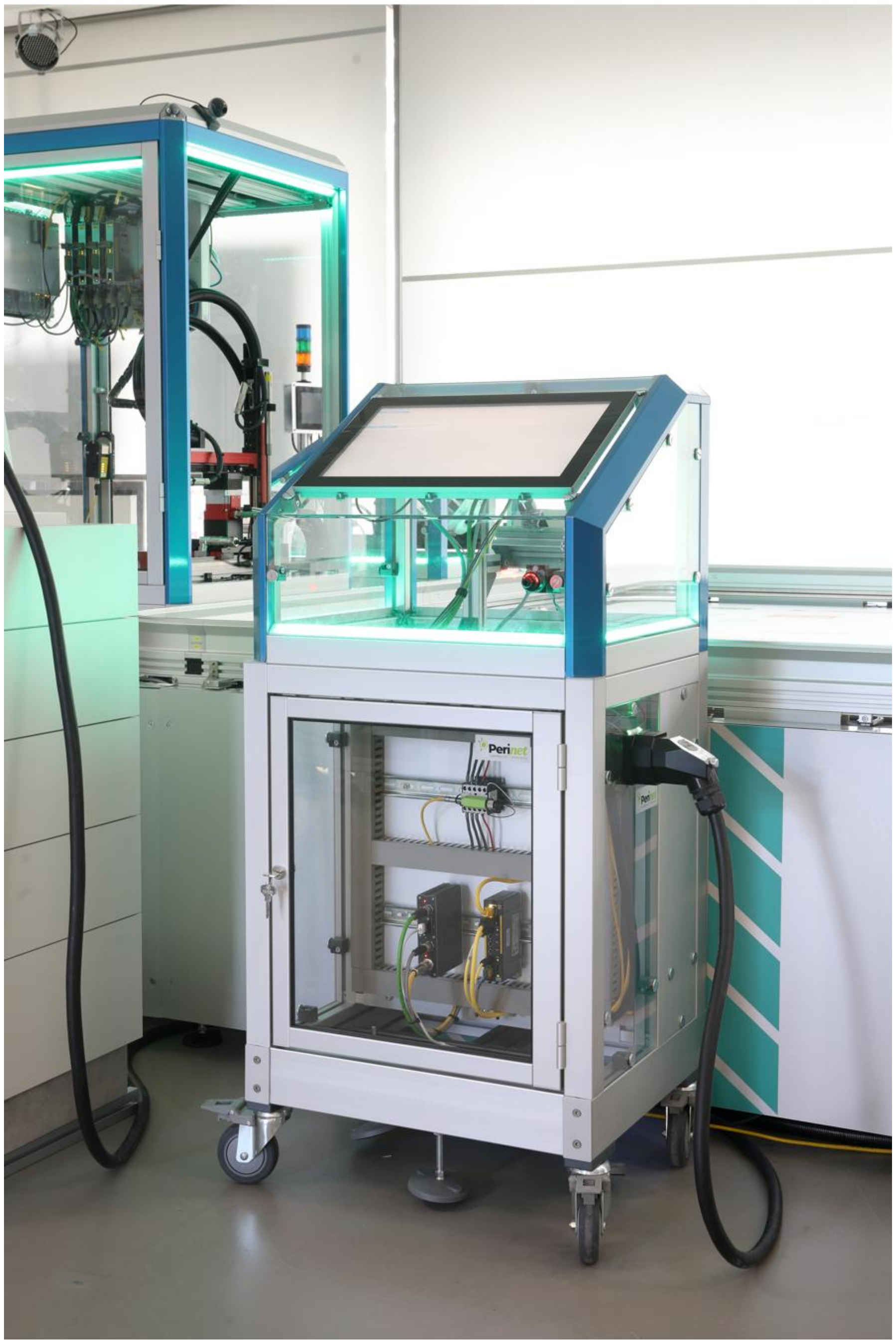


Figure 5: Module "Quality 1" (Picture by A. Sell)

Type	Name	Manufacturer
Micro Controller	ArduinoNano	Arduino
Touchpanel PC	Panel PC 3100 Einbaugerät	B&R
Sensor (Camera)	acA1440-73gc	Basler
Plug	Smart Electrical Connector (SmEC)	Harting
IPC	HAIIC MICA ETH ES	Harting
Interface Node	PeriNet PeriNODE	PeriNet / EATON
Actor	PSEN sl-1.0p 1.1 / PSEN sl-1.0fm	PILZ
Sensor	PSEN cs1.1p	PILZ
PLC	SIMATIC ET 200SP	Siemens
Switch	IES101G2SFPW	StarTech

Table 3: Hardware components of module 2

Module 3: "Data Refuel"

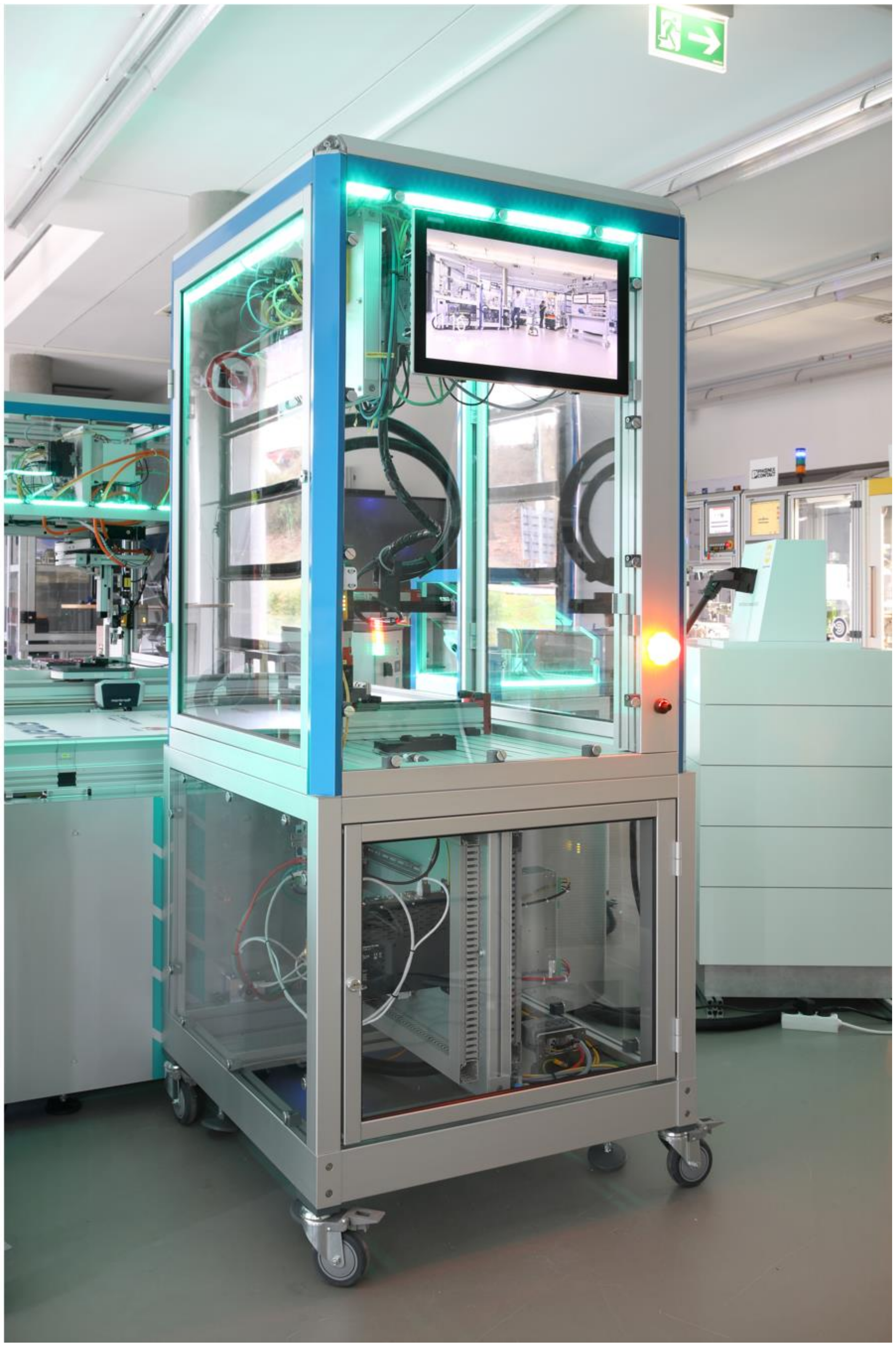


Figure 6: Module "Data Refuel" (Picture by A. Sell)

Type	Name	Manufacturer
Actor (Linear axis gripper)	Linear axis system (summarized)	Afag
Actor (Gripper)	EHPS-16-A-LK	Festo
Automation PC	APC3100	B&R
Micro Controller	Arduino Nano	Arduino
Profinet Koppler	X20	B&R
Plug System	HAN-Modular-System	Harting
Switch	Eagle 30 Industrial Security Router	Hirschmann
Controller	Servo Drive C1100	LinMot
Sensor (RFID Reader)	IQT1-F61-IO-V1	Pepperl & Fuchs
Actor	PSEN sl-1.0p 1.1	PILZ
Sensor	PSEN cs1.1p	PILZ
PLC	PNOZ m B1	PILZ
Touchpanel PC	SIMATIC IPC477E	Siemens

Table 4: Hardware components of module 3

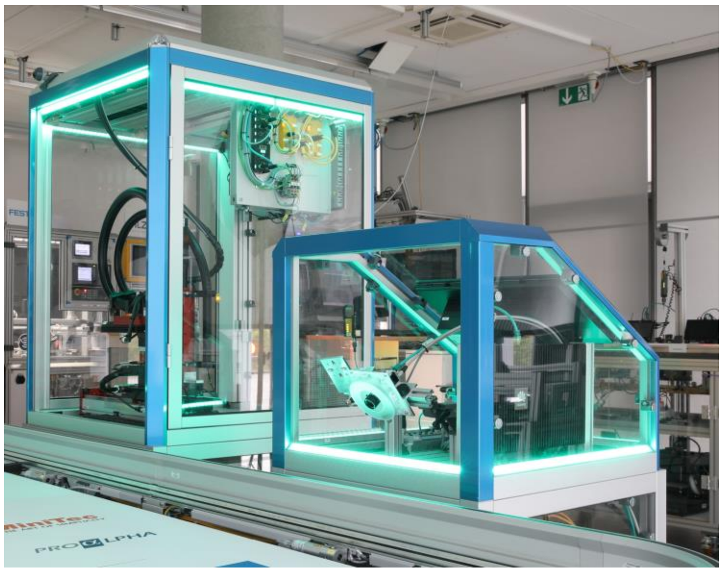


Figure 7: Modules "Data Refuel" and "Quality 2" (Picture by A. Sell)

Module 4: “Quality 2”

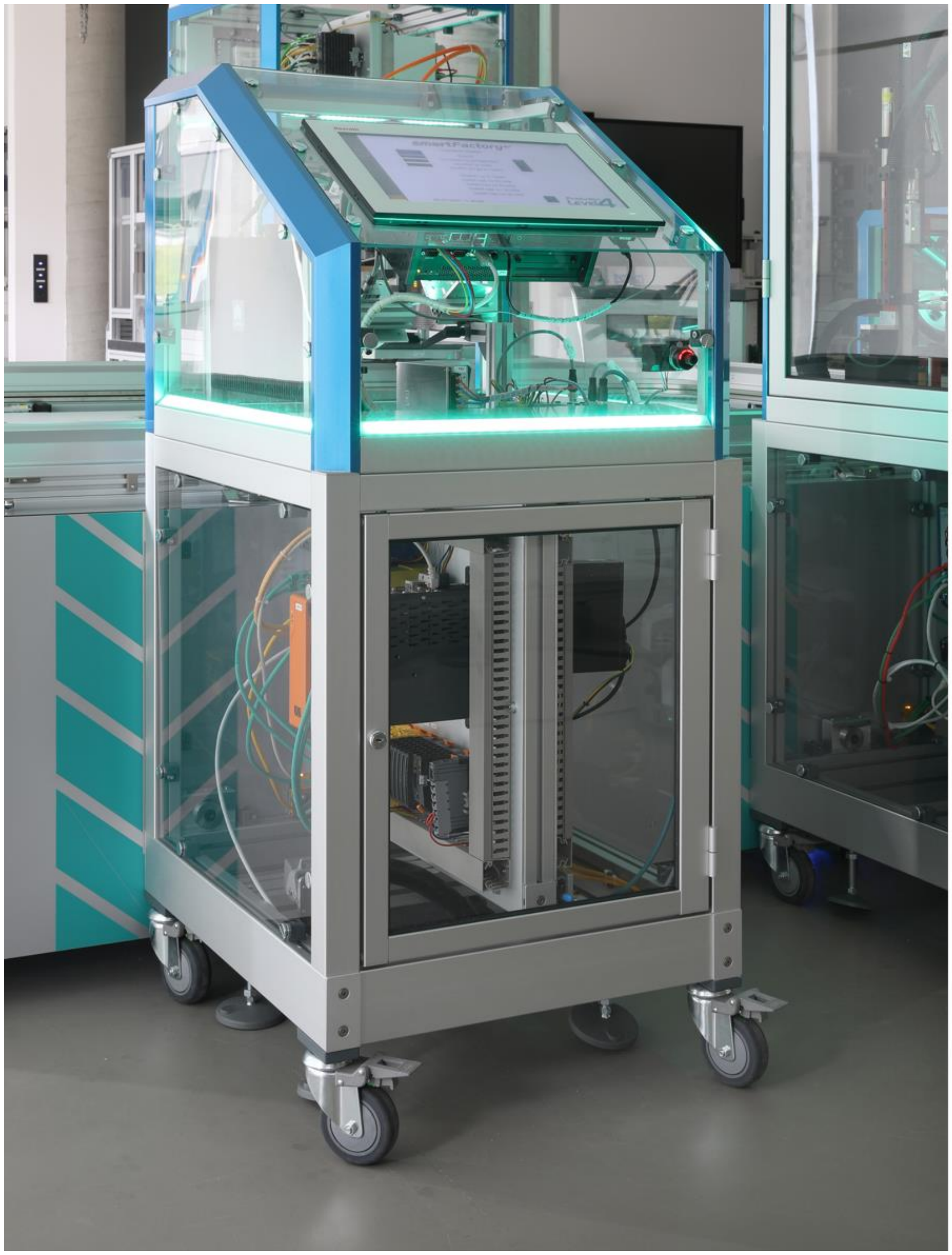


Figure 8: Module "Quality 2" (Picture by A. Sell)

Type	Name	*Manufacturer*
Micro Controller	Arduino Nano	*Arduino*
Automation PC	APC3100	*B&R*
Profinet Koppler	X20	*B&R*
PLC	X20 System	*B&R*
Sensor (Camera)	acA1440-73gc	*Basler*
Switch	Eagle 30 Industrial Security Router	*Hirschmann*
IPC	Atlas 500	*Huawei*
Actor	PSEN sl-1.0p 1.1 / PSEN sl-1.0fm	*PILZ*
Sensor	PSEN cs1.1p	*PILZ*
Touchpanel PC	VR4215.1	*Rexroth*

Table 5: Hardware components of module 4

Furthermore, there are hardware assets, which are not related to the CPPM itself, for example the transport system. These components are listed in Table 6.

Type	Name	Manufacturer
Edge Cloud Server	OneCite	GEC / Rittal
Conveyor system	TracSet	Montratec
Conveyor system (planned)	ACOPOStrak	B&R
Tablet	Ipad	Apple
Smart Glasses	Hololens 2	Microsoft
Infrastructure Box	Infrastrukturbox	SmartFactory
ATV	Robotino	Festo
Manual assembly station	HAP	SmartFactory

Table 6: Other hardware components / assets

2.3 SmartFactory Software Basis

The SmartFactory-KL testbed environment consists of various architecture components and communication protocols, that are described in this chapter. The overall architecture is shown in Figure 9.

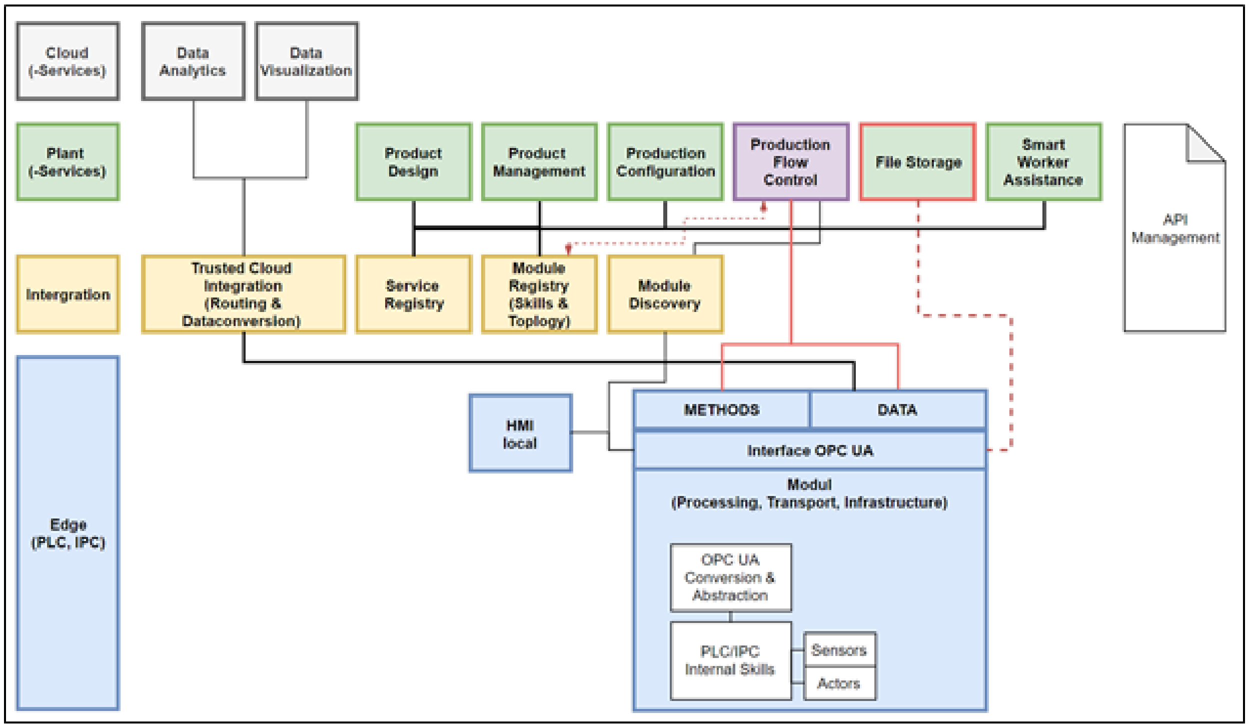


Figure 9: SmartFactory architecture

Each CPPM of the PL4 demonstrator (production units and transport units), which are related to the edge level, consist of physical assets that are controlled by internal PLC that is connected to corresponding actors and sensors. The capabilities of each CPPM are implemented with a skill-based approach using OPC UA, that provides a uniform interface to control the CPPM and to get data from it.

The architecture consists of the Plant-Service level of several software components for setup of individual product design during creation of customer orders, the build-up of a recipe in the production configuration in terms of a production plan for the demonstrator and the production configuration for orchestration of the production process. Furthermore, an interface for smart worker assistance is provided, to connect with HMI devices.

For data exchange of pictures takes from the quality check, that can be uploaded to the product by using the "Data Refuel"-Module, an own FTP Server is available, which communicated with the modules OPC UA interface. The integration components take care about registry and discovery of CPPM skills (CPPM skill registry) and Plant services (Service registry) and provide furthermore a trusted cloud integration gateway. The communication protocols, related to their architecture components, are listed in the table below.

Protocol	**Architecture Component**
OPC UA	Production Modules Production Flow Control HMI
MQTT	Cloud Applications Enterprise Resource Planning System
REST	Enterprise Resource Planning System Production Flow Control Production Design Production Configuration Registry (Skills and Topology) API Management Discovery
Profibus	Production Modules Infrastructure Nodes
FTP	Upload product pictures for quality control or for saving it on the product

Table 7: SmartFactory Architecture Protocols

The services with security mechanisms are presented in Table 8.

Cloud Services	Comm.-Protocol	Security
IBM Cloud	IBM PSB	Authentication
Gaia X Connector	MQTT & IDS (HTTP)	Authentication (Certificates)

Table 8: Clouds and Connectors

2.4 Requirements Scope of first prototype

Based on the modular service-oriented architecture of the SmartFactory KL testbed, the basis for setup of corresponding Asset Administration Shells (AAS) and Digital Twins has been prepared. To enable interoperable interfaces for the testbed assets, each CPPM can be represented as I4.0-Component, that can be ensured by adding an AAS to it. The setup of AAS furthermore enable an agent interaction with the physical hardware layer, or to software services like planning software, and can be combined with knowledge representations for semantic interoperability. The AAS can thus be built by using a related software framework like the BaSyx-Middleware, or integrated in OPC UA directly, following the OPC UA companion specification for AAS.

In scope of MAS4AI, the representation of the described physical and virtual components in form of AAS or uniform interface descriptions is focused to ensure a connection, control and data exchange between the system and a MAS framework in general. The steps to make the testbed ready for an agent-control, adding semantic technologies and for further integration of planning and machine learning agents are in the scope of the first prototypical implementation.

Related to the requirements of the MAS4AI approach, the aspects of planning, quality check and human-machine interaction are essential in general. The scope of the current use case planning, this means consideration of the availability of required serviced during production as defined in the products recipe, thus should be enhanced to be able to react to changing situations or target functions (e.g., time, costs, quality, sustainability). An improved flexible integration of quality controls in combination with the agent’s behavior and data from assets can help to detect weaknesses in the production process, which are leading for example to higher waste. Integration of HMI agents can help to support operators in manufacturing environments and setup decisions as part of a MAS system.

In the first iterations of the MAS4AI system setup and evaluation in the Smart Factory testbed environment, the focus was on initial build-up of components under consideration of the given physical and virtual assets to be integrated. The requirements for this setup, related to the agent types in MAS4AI, that should be considered in these iterations are integrated mainly for resource agents and infrastructure purpose.

To fulfil the MAS4AI objectives, a list of the foreseen agents that should be considered during implementation and evaluated with their corresponding types is shown in Table 9. This list contains agent types, based on collected requirements in WP1 and is considered also in the first iterations of the Smart Factory testbed demonstrator evaluation.

Agent	Type
Local planning agent	Production planning
Resource agent	Resource
Production module agent	Resource
Transport agent	Resource
Quality inspection agent	Quality inspection
Safety agent	Safety
HMI agent	HMI
Energy agent	Resource
Infrastructure agent	Information
Data collection agent	Information
Product agent	Product

Table 9: SmartFactory: overview of expected agents and their types

Related to several domains of the use case requirement analysis, the following scope has been defined for the Smart Factory use case.

Agent name	**Description**	**Domain**
Local planning agent	The local planning agent can perform job-shop scheduling with a limited number of machines and interaction with a global planning service.	Time, Flexibility

Quality Inspection agent	The quality inspection agent able to perform a classification of products based on defects and a classification of defective products based on the possibility of repair.	Sustainability, Quality
Production module agent	Production module agents have the possibility to interact with corresponding assets on the shop flow in the manner of resource agents.	Time, Flexibility
HMI agent	The HMI agent is an agent with interfaces to data visualization and database services.	Human Machine Interaction

Table 10: Categorization of agents

Furthermore, transport agents have specific requirements, following aspects of safety, reliability, energy management that have impact to their communication and information.

A list of expected physical interactions between agents and the process environment is also presented in Table 11. For the proposed prototype development and evaluation, the implementation of production module agents as well as transport agents are mainly in focus of the setup.

Agent	Input	Output
Production module agent	Raw material	Processed material
Transport agent	Transported good	Transported good
Quality Inspection agent	Product	Product (+Quality assessment)
Energy agent	Energy	-

Table 11: SmartFactory: expected physical interactions between agents and their environment.

The list of special considerations for agents related with ethics or trustworthiness are presented. Due to the focus of the first iterations prototype, the following aspects are not mainly focused.

Agent	Special Considerations
Safety agent	Safety regulations
HMI agent	Trustworthiness, Usability / accessibility
Data collection agent	Ethics (privacy rights, legal…)
Product agent	Reliability, Privacy, Trustworthiness

Table 12: SmartFactory: special considerations

Predefined agent communication, which is based on the identified agent types, is necessary to setup the main interactions between several agent types with events and the related message syntax and semantics, following the approach of standardized conversion. The deployment of agents in the SmartFactory workflow, as it is expected to meet the requirements, together with the corresponding links between agents that are expected to interact within the process, is presented in Table 13.

Local planning agent	Task scheduling data. Process-related time to fulfil production orders. Other data needs to be assessed during the MAS4AI project.
Quality Inspection agent	Picture and configuration criteria for batch size 1 quality control. “Ground truth” Data of the correct product configuration.
Production Module Agent	Sensor and Actor Data of the corresponding Modules in OPC UA – with properties and operations (as skill-based approach) Each current module states Skill-related time and energy data and their sequences Data about current configuration
HMI Agent	Status of the production orders Status of the production modules Machine data Database containing component documentation Information about the current operator

Table 13 - Required Agent data

The overall interactions of these agents are visualized in Table 14, considering resource agents, which are mainly in the scope of the first prototypical implementations, as well as the connection and data exchange with other agent types.

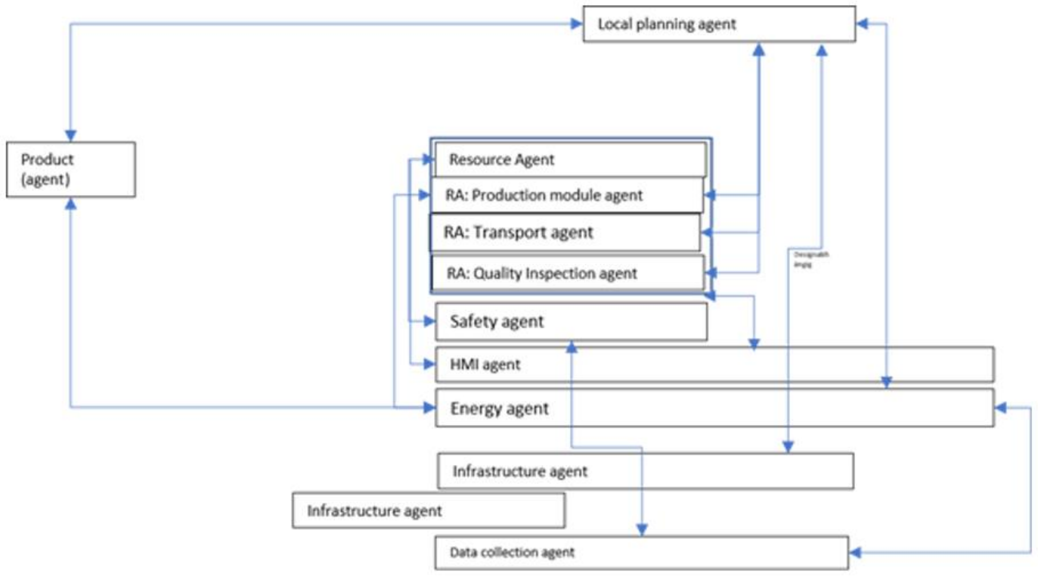


Table 14: SmartFactory: agent-based workflow

The interactions of each agent types, following the given overall setup, is also listed in the table below.

Agent	Interaction mechanisms
Local planning agent	Local planning agents: Planned schedule Product agents: type, processing step, location, deadlines Production module agents: Skills, availability, key processing indicators Quality inspection agents: Defect products Energy agents: Energy prices, adjust machine performance Infrastructure agents
Resource agent (incl. Production Module agent, Transport agent, Quality inspection)	HMI agent (to interact with humans) Safety agent (safety assessment, human interaction) Energy agent (collect energy needs, (optional) adjust performance, (optional) alter state (standby, offline, etc.)) Data collection (collects data from resource agent)
Safety agent	HMI agent (show safety level, safety information, interaction with / location of operators) Data collection agent (sensors/actors, create protocols)
Energy agent	Product agent (fill lifecycle file) Data collection agent (report energy consumption)

Table 15: SmartFactory: interactions between agents and associated requirements

2.5 Method for Prototype Development

The Smart Factory prototype development and evaluation for the MAS4AI framework follows an iterative software prototyping approach, with goal of validation of the WP2, WP3, WP4 and WP5 results of the given project. The setup of the MAS is therefore based on a MAS framework in combination with a message-based middleware solution, like Apache Kafka and BaSyx, but also considering industrial Internet-of-Thing's web service technologies like HTTP, MQTT and OWL.

The prototyping therefore is considering the following aspects into the iteration's development and prototype evaluation:

a. Digital twin preparation and integration with the AI agents for early assessment,
b. pilot setup that involves integration of existing legacy systems,
c. execution and validation of the individual agents and
d. execution and validation with the whole MAS4AI system

During the prototype setup and implementation, the main requirements for the given context are considered. This includes the consideration of protocols to let the agents communicate with each other in the MAS framework environment, as well as the interaction with the shopfloor environment. Therefore, the build-up of digital twins and AAS is necessary for further integration, also for activities towards AI agents.

The development considers the integration of available legacy systems, as presented as physical CPPM of the Smart Factors PL4 demonstrator. The software components on the factory services and integration level, like Registry and Discovery, must also be considered in an iterative method.

The setup of the agent is following the agent type definition which has been presented on basis of the MAS4AI requirements analysis. For the first prototypical iterations, the integration of the resource agents is focused. For following iterations, the prototypical implementation will be enhanced with the agent types of the requirements list.

An implementation in the Smart Factory demonstrator does also consider the components and approaches of the MAS4AI work package contents for demonstration and evaluation purpose. This means, that during the software prototype setup, new components must be integrated and evaluated into a given MAS prototype state. Therefore, the addition of new components as well as interfaces, can be considered by using an iterative development method.

3 Iterative MAS4AI-Prototype Development

The MAS4AI prototype development for the PL4 Line followed several iterations to make the assets of the demonstrator available for agent's interconnection and communication. After completing a defined iteration, the next build of the prototype has been focused.

3.1 1st Iteration – Setup first AAS and connect one transport agent to its asset

In the first iteration, the considered assets of the demonstrator (at first only the transport asset) have been analyzed and prepared towards build-up of AAS. Given predefined interfaces of the assets which can also be parametrized, the middleware level and MAS level must be prepared for setup of the AAS.

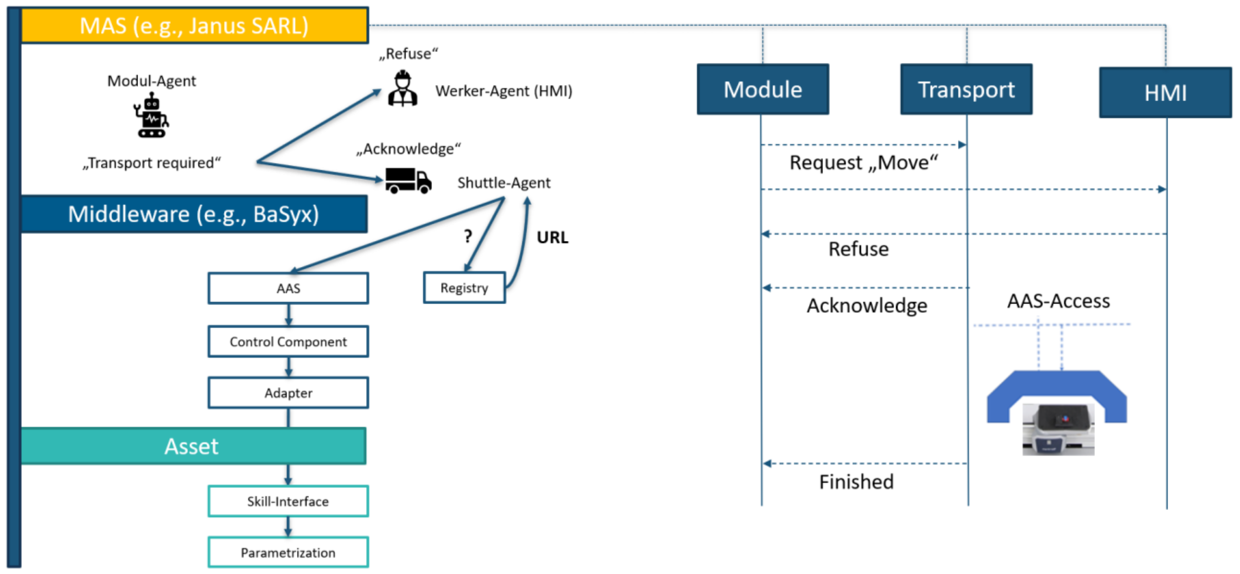


Figure 10: First iteration prototype

In case of the Smart Factory PL4 demonstrator, the interface on asset level comes with a skill based OPC UA interface. As middleware solution, the BaSyx-Middleware has been used. For the

MAS-System the Janus SARL environment is used, setting up an initial agents' configuration of simulated production modules as resource agents, which are calling for transport system. Furthermore, one transport agent has been prepared, also as resource agent, to consider the simulated requests form the production modules and to control the shuttle asset respectively.

For the first prototype iteration, the focus was on connection and control of the transport system, the PL4 line rail shuttle system. The goal was mainly to enable a general setup configuration in the BaSyx-Middleware to build up an AAS for the shuttle asset and then to setup the connection to its related OPC UA interface, by using the OPC UA adapter of BaSyx. The focus on this first iteration was, to enable an asset control out of the selected MAS system.

Given the proposed MAS4AI framework approach, the middleware and MAS level does not focus on predefined technologies or frameworks but have to offer the required functions which are related to their level. This means, that other middleware solutions or software components can be used, to setup a MAS-System or build up AAS and interconnection to the asset level. The given approach of the first MAS4AI implementation in the Smart Factory testbed therefore consider on open-source software frameworks, which comes with a suitable setup and functional scope for the proposed use case requirements.

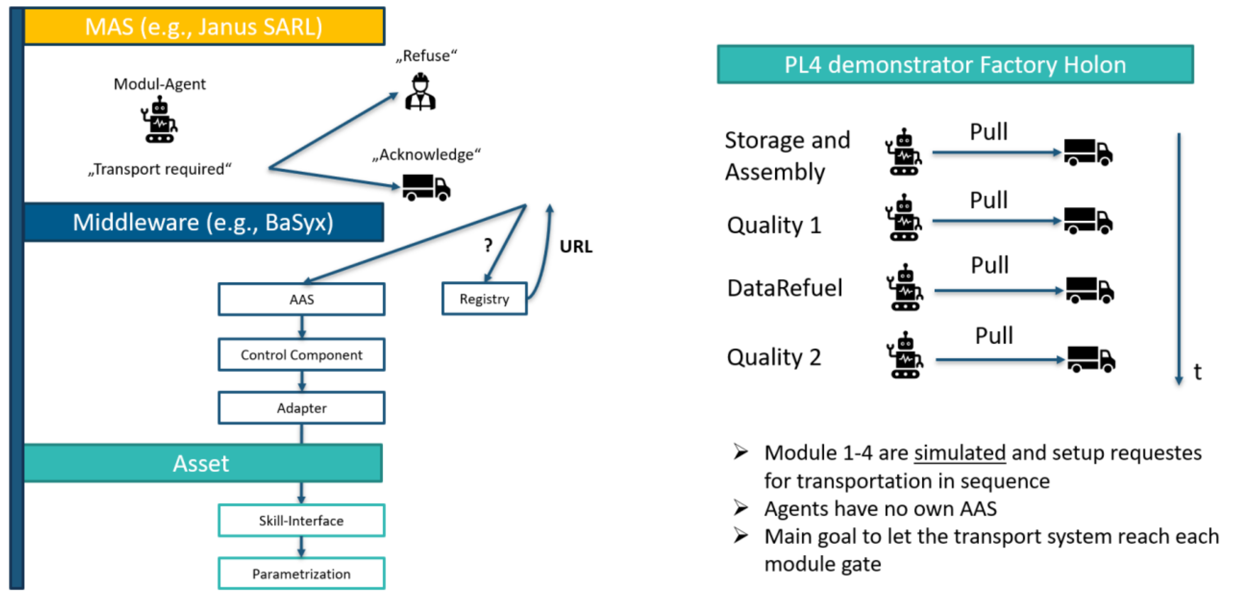
In the middleware level, the assets of the demonstrator are represented with own AAS and submodel "Control Component", that is used for the dispatch of AAS methods from a central or decentral controller. The AAS of the considered assets are registered in the BaSyx-Registry and the related endpoint information of the AAS is stored there already. The communication towards the assets is done with a OPC UA adapter, which depends on an interface configuration for the focused asset.

The AAS submodel contains properties and capabilities of the asset, like information about their current state, based on available state machines. The capabilities are implemented on the asset-level as related skills with the uniform OPC UA interface. If a request is sent to the AAS submodel, the BaSyx-Middleware process the information towards the field device adapter, which in this case is connected by using the OPC UA protocol. A predefined mapping-configuration between AAS submodel elements (like properties and operations) to their asset related OPC UA node is used in the adapter.

In the MAS system of Janus, one resource agent that represents the first production module, one resource agent that is representing the transport system and one HMI agent has been prepared. The request for transportation goes to the HMI agent, which refuses the given request because he is not able to process it, and to the transportation agent. The transport agent acknowledges the request, search the endpoint of the shuttle AAS inside of the BaSyx registry and process the AAS method, resulting in a shuttle control towards the requested position of the resource agent. After the request has been processed successfully, the information has been sent back to the resource agent, which have sent the request.

3.2 2nd Iteration – Setup holonic resource agents for production module sequence

The main goal of the second iteration was to enhance the control of the integrated transport shuttle in the MAS system. Resource agents are setup for each production module asset, which included the “Storage and Assembly” module, the “Quality 1” module, the “Data Refuel” module and the “Quality 2” module. Each of these agents simulated their actions in a default production process in the PL4 line. A PL4 demonstrator factory holon has been setup to let the given agents of this prototype iteration spawn. Each simulated resource agent, that stands for a production module, has the goal to send a request towards the shuttle agent, resulting in a related request to the shuttle asset.



- Module 1-4 are simulated and setup requestes for transportation in sequence
- Agents have no own AAS
- Main goal to let the transport system reach each module gate

Figure 11: Second iteration prototype

The resource agents, as simulated production modules, are requesting the shuttle agent for transport in a predefined process sequence of a PL4 line default process. The shuttle agent considers the requesting resource agent, and the requester data is also sent to the shuttle assets

AAS. The adapter processes the information, resulting in a different parametrization which is sent to the related skill interface (another shuttle gate that should be reached from the asset). In this example, the resource agents are communicating with each other, to process in sequence. In the given process, the shuttle was able to reach the position of each production module with agents, leading to a full integration and request-related dynamic control of the transport agent with its asset's parametrization.

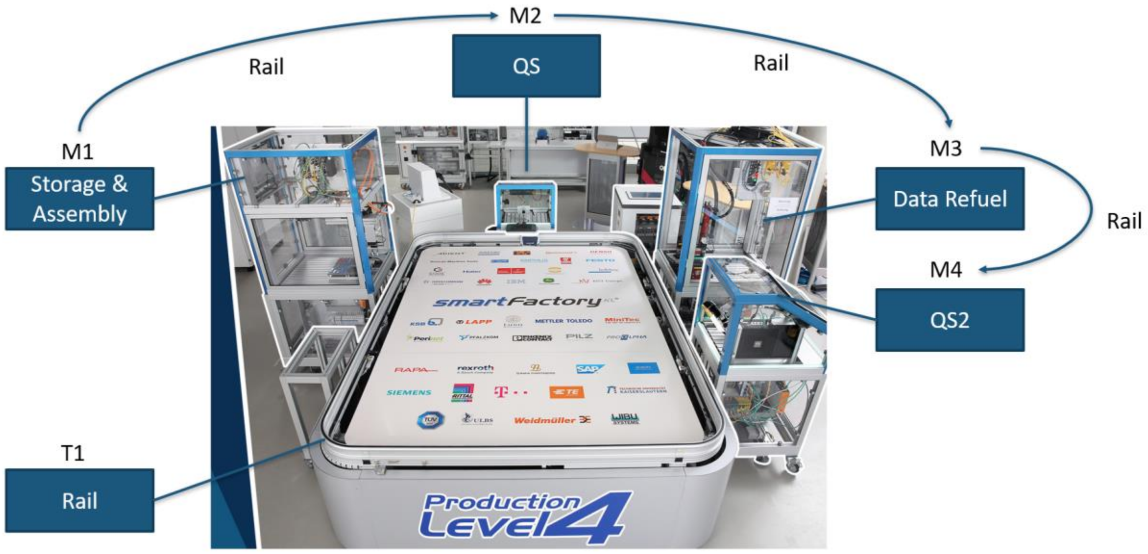


Figure 12: Shuttle Transportation of second iteration

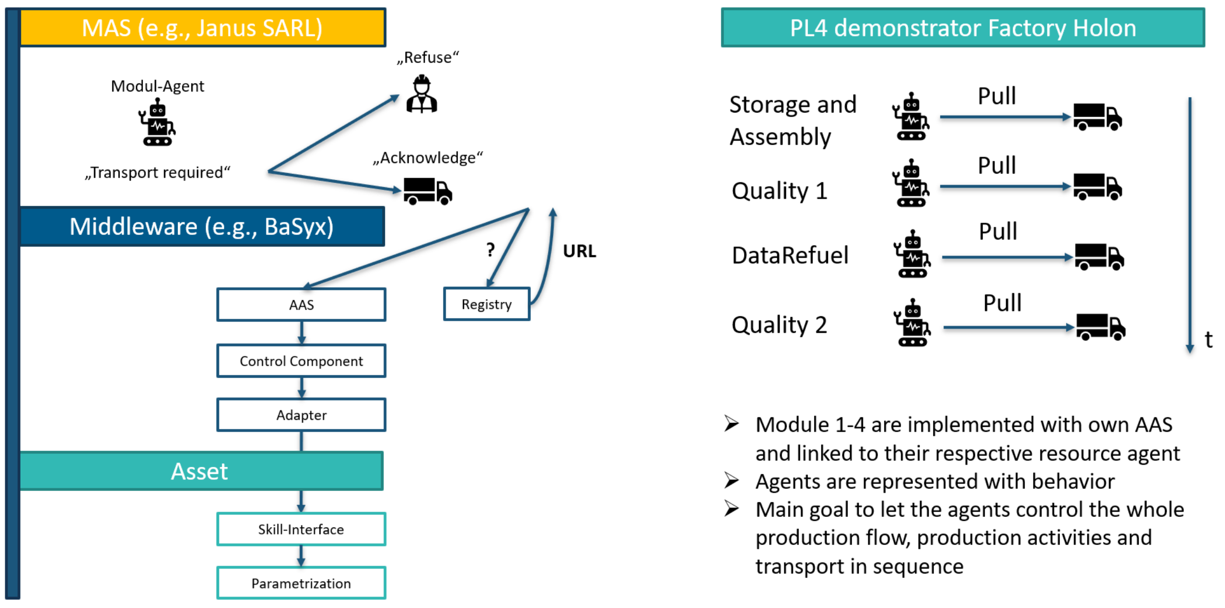
3.3 3rd Iteration – Setup assets with AAS and agents for process execution

The main goal of the third iteration was to setup the AAS for all production module and to integrate them in the BaSyx-Middleware. For each production module, the related resource agent in the MAS system is setup and the related agent behavior is developed. In this step, the first functional integration on the middleware layer has been used and adapted for each of the production modules, which are now coming with at least equal or more complex capabilities (e.g., assembly or data refuel), compared with the transport system.

In this step, the first functional integration on the middleware layer with the transport shuttle has been used and adapted to each of the production modules, which are coming with equal or more complex capabilities (e.g., assembly or data refuel), which now must be represented as resource agent behaviors and capabilities. Therefore, the behavior and capacity patterns of the

Janus framework could be adapted, to encapsulate and use the predefined behavior-specific capacity functions with the related AAS communication.

An AAS with "Control Component" submodel thus has been setup for each production module asset, which included the "Storage and Assembly" module, the "Quality 1" module, the "Data Refuel" module and the "Quality 2" module. The OPC UA adapter in the BaSyx Middleware has been prepared to use several configurations, to dispatch AAS events and data (like operations call or property updates) with resource-related parametrization to the respective OPC UA node of the intended asset.



- Module 1-4 are implemented with own AAS and linked to their respective resource agent
- Agents are represented with behavior
- Main goal to let the agents control the whole production flow, production activities and transport in sequence

Figure 13: Third iteration prototype

The agents in the MAS system are now able to process the sequence like in the prototype of the second iteration but have now the possibility to control the production process with production steps and transportations in the physical demonstrator environment.

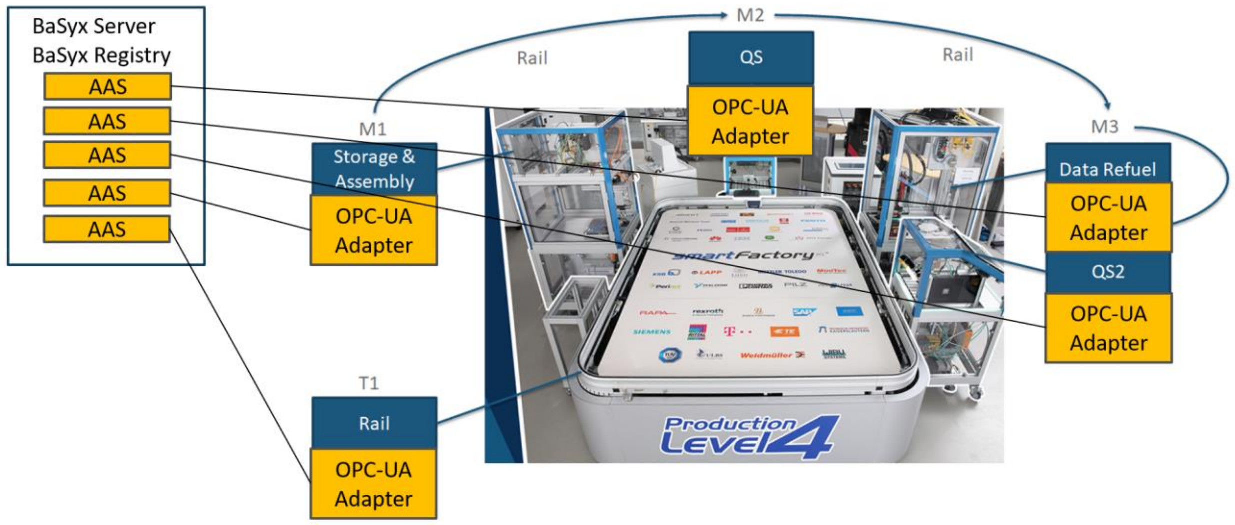
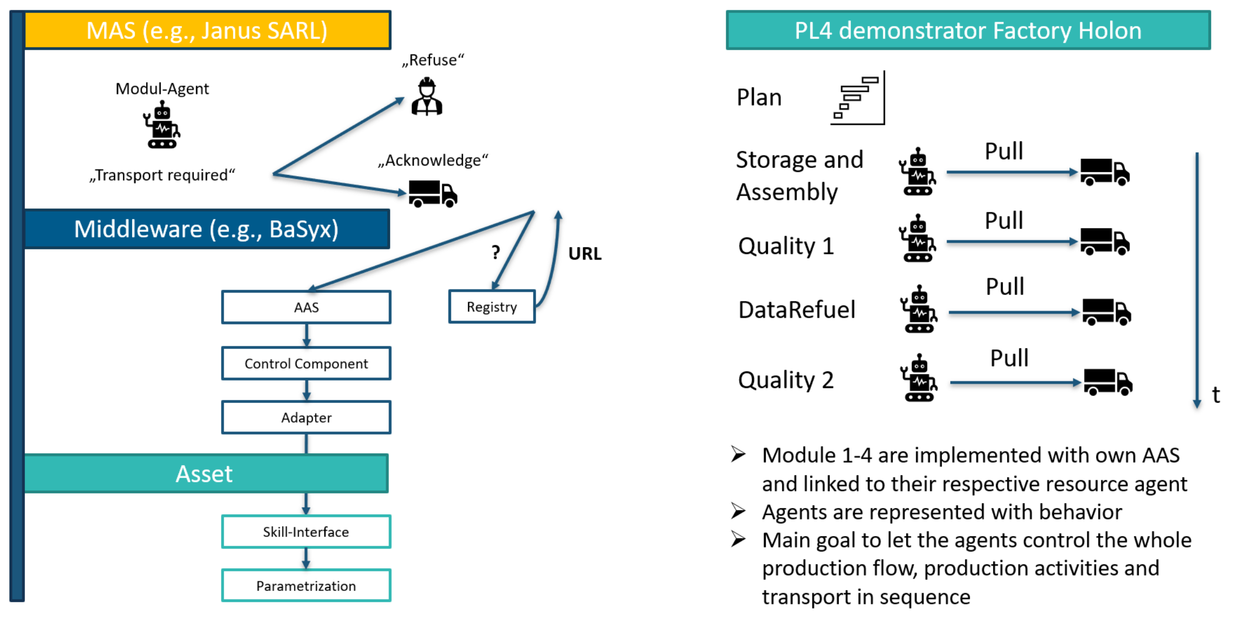


Figure 14: Shuttle Transportation of Production Modules AAS of third iteration

3.4 4th Iteration – Integration of planning component and AAS also for agents

The main goal of the fourth iteration was to integrate a component to let the resource agents work on a predefined planning. The holonic agent for the PL4 line must be able to provide data for planning purpose and to adapt production plans more flexible with current agents' behavior. Based on the results of the third iteration prototype, it is possible to process a full production sequence in the demonstrator, but due to the predefined sequence of this iteration, the agent's communication follows also predefined patterns.



- ➢ Module 1-4 are implemented with own AAS and linked to their respective resource agent
- ➢ Agents are represented with behavior
- ➢ Main goal to let the agents control the whole production flow, production activities and transport in sequence

Figure 15: Fourth iteration prototype

With the adaption of a plan, it is possible to change the current production and communication pattern more towards flexibility and reconfiguration of production units. Furthermore, a more flexible reaction due unforeseen events can be provided, for example if one production module has an error. Inside the factory holon, agents then can communicate and try to solve the problem, searching for other resource agents which can fulfil the action of the module that is not available anymore.

Related to the demonstrator setup, it is possible to schedule a plan for the PL4 demonstrator holon, and instead of quality check of the module “Quality 1” and then “Quality 2” the quality checks should be done twice by using the same quality module, for example if the quality of the check could for a given product could only be done with one of these modules. Otherwise, in case of error of one quality module, for example if “Quality 2” cannot be used, it is possible to adapt the predefined plan and then search for alternative modules. In this case, an alternative option would be module “Quality 1”, or also the HMI agent, if a worker is available and can do the quality check manually.

The adaption of a production plan comes also with the requirement of integration of a local planning agent. The integration needs the availability of AAS models for the current agent types and the possibility to host them in the BaSyx-Middleware, as well to register them in the registry. The starting and registration of holonic agents as well as their agent types furthermore requires the integration of an RDF-Store because the lack of the BaSyx-Registry for AAS to store more information than the endpoint information. This information is essential to search for suitable AAS as well during the initial start of holonic agents as well as for reconfiguration purpose.

The prototypical implementation of the fourth iteration is described in this document for the purpose reconfiguration and is currently in development and evaluation in the Smart Factory testbed. This iteration focuses on reconfiguration and adaption to a plan, as well to build up this reconfiguration due to the error of one of the quality modules. The integration of the RDF-Store as well as planning and quality check agents also are a good possibility to evaluate the MAS4AI WP3, WP4 and WP5 approaches. Results of this integration and further MAS4AI framework evaluation will thus be considered in future research.

4 Smart Factory MAS4AI Implementation

The Smart Factory testbed implementation of the MAS4AI framework follows the proposed setup and content of the work packages. The structure is presented in Figure 16.

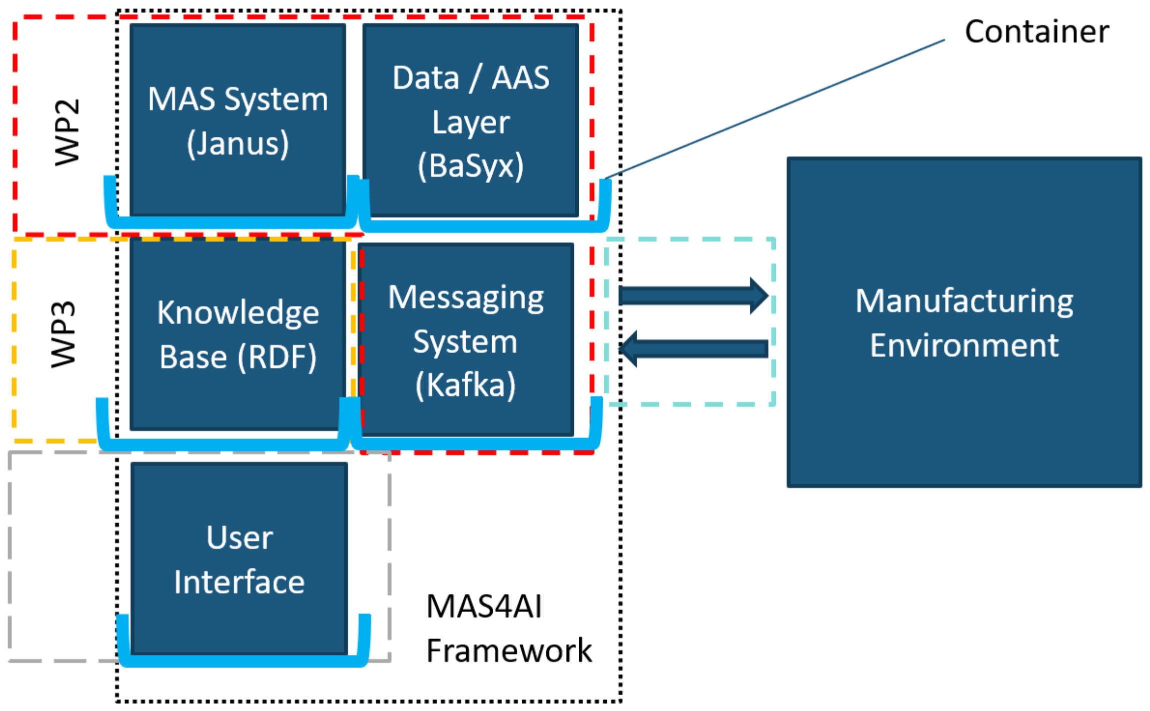


Figure 16: MAS4AI Framework

Related to the main aspects, the Smart Factory testbed demonstrator focused on topics:

- Implementation and evaluation of the MAS4AI setup as proposed in WP2
- Create a holonic representation for the existing (legacy) PL4 Demonstrator
- Integration of message systems like Apache Kafka for external communication of agents and communication or data collection of assets in a distributed environment
- Flexible integration at the AAS layer, for example by using BaSyx-Middleware or OPC-UA usage for real-time purpose on station level (e.g., also combined approach)
- Integration of RDF-Store for AAS semantic annotation as well as for requests during the start of the system
- HMI-Integration for system configuration and maintainer support

The holonic setup of the MAS4AI framework, as proposed in WP2, is presented in Figure 17. A holonic agent represents an entity, that can be deployed in a distributed environment and comes with an own AAS, in which the configuration and parametrization of underlying agent types inside this holon can be stored. A holonic agent in this way can thus consist of an own MAS, in which all related agents could be created and configured. An example of such an holon could be the resource holon of the Smart Factory PL4 demonstrator line, consisting of several resource agents that are controlling the production modules as well as the transport assets. Inside such a holon, also other agent types could be deployed, like quality agents as well as (local) planning agents for example.

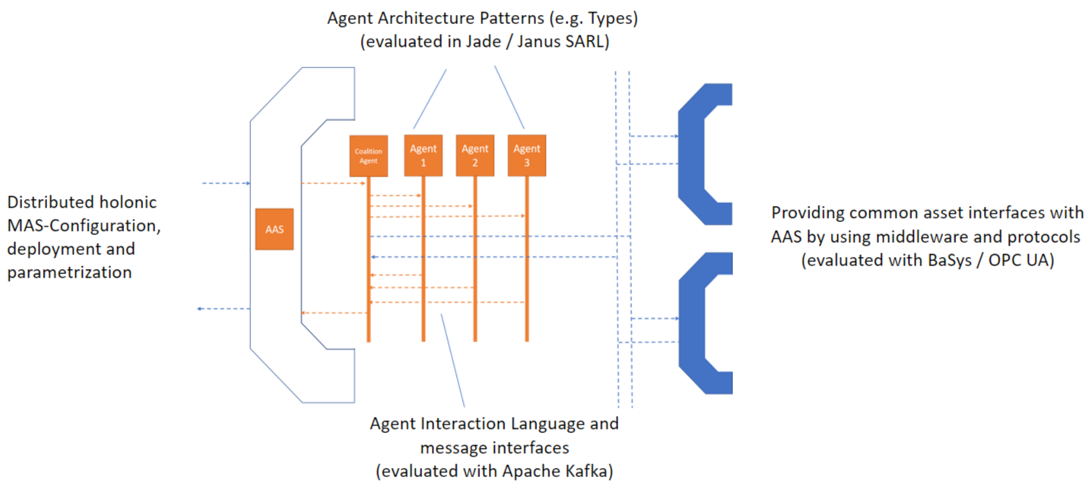

Agent Architecture Patterns (e.g. Types)
(evaluated in Jade / Janus SARL)
Coalition Agent
Agent 1
Agent 2
Agent 3
Distributed holonic MAS-Configuration, deployment and parametrization
AAS
Providing common asset interfaces with AAS by using middleware and protocols (evaluated with BaSys / OPC UA)
Agent Interaction Language and message interfaces
(evaluated with Apache Kafka)

Figure 17: MAS4AI Holonic Setup

The agent types are integrated as agent patterns, which are setup in common MAS frameworks like Jade or Janus. Nevertheless, the concept of MAS4AI considers a framework-independent approach, so that the agent types and their communication could be setup in similar way also in other MAS.

The agent interaction language is used in the first prototypical implementations by using the Janus agent communication mechanism. The approach to use framework-mechanisms is useable if all agents of an holon are setup and active in a same runtime environment. If the (holonic) agents are deployed on distributed hardware (e.g., if agents are deployed on machines or some AI agents are deployed in several cloud environments), the connection to a central message communication system like Apache Kafka is needed for data exchange and communication.

The interaction of agents with hardware or software components can be realized by using the AAS, providing predefined interfaces by using for example the BaSyx-Middleware or OPC UA.

4.1 AAS and Digital twin preparation

Related to the MAS4AI holonic setup, the preparation and setup of AAS or digital twins is necessary for the integration into the MAS4AI framework. This part of the framework is represented in the "Data / AAS Layer" and is necessary to build up a virtual representation of the manufacturing environment. In the use case of the Smart Factory PL4 line, this also considers physical as well as virtual assets.

The physical assets, in form of CPPM, are integrated and represented as AAS into the BaSyx-Middleware. Therefore, the setup of several elements in the BaSyx-Middleware has been done, which should be considered here. The implementation of the AAS has been done with the BaSyx-Middleware. The related software elements of the middleware which are used for integration and representation of an asset with the AAS are presented in the Figure 18. For the AAS preparation, the BaSyx-Middleware [2] in Java is used with version 1.02.

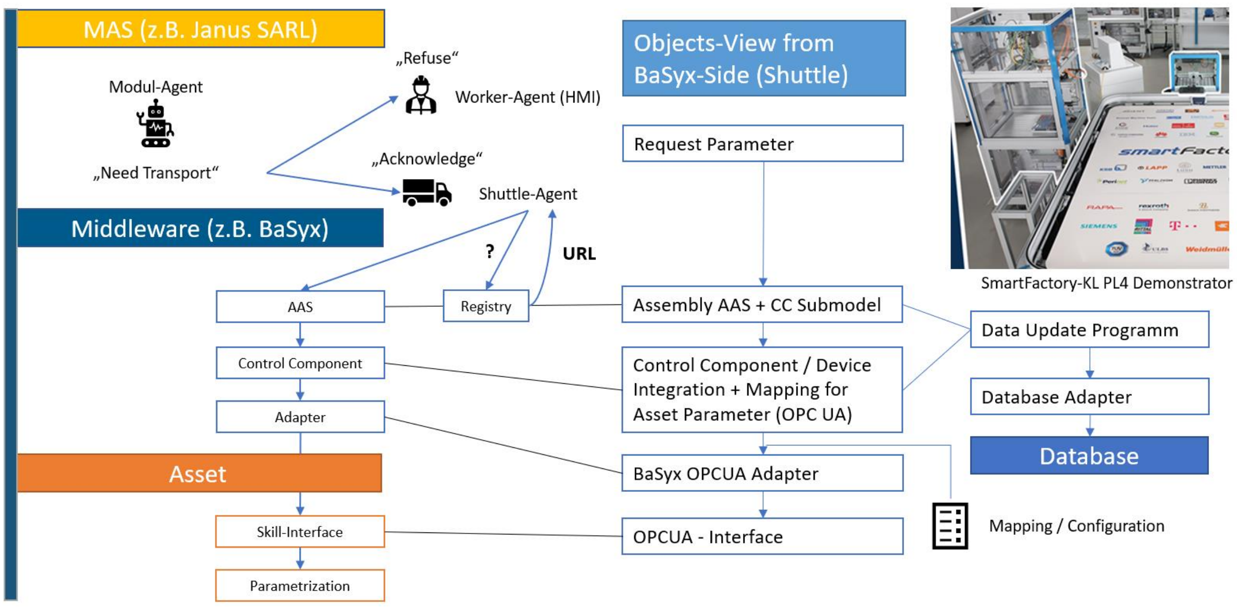
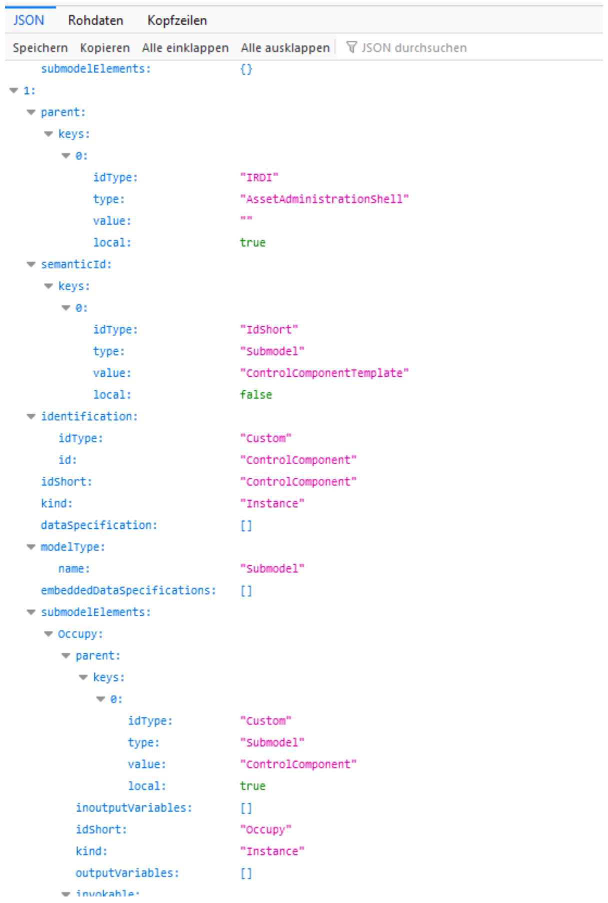


Figure 18: Technical objects of BaSyx Middleware

The process of processing events, which are sent to the AAS, and data which can be got from the system, is described from a top-down view, introducing the used components with their role in the prototype.

If an agent of the MAS-System, for example a resource agent needs to find an AAS of an asset, they will investigate the BaSyx Registry to get the related endpoint. If an AAS is found, then the agent can perform a HTTP Rest-Call towards the AAS with corresponding parameters. Parameters could be the agent ID, for example to lock the asset access for the given agent and to avoid that another agent can interact with this asset for control purpose. In case of the shuttle AAS, the information about the destination of the shuttle request must be set.

The assets AAS with the submodel “Control Component” is stored in the registry [3] and is hosted by using a BaSyx AAS Server [4]. The integration and processing of events inside the BaSyx-Middleware can be performed by using the “Control Component” pattern [5], or by needs of data acquisition from shop floor devices, also with a “Device Integration” object which provides the connectivity with the required protocol of the asset [6].



```
    submodelElements:              {}
▼ 1:
  ▼ parent:
    ▼ keys:
      ▼ 0:
          idType:                  "IRDI"
          type:                    "AssetAdministrationShell"
          value:                   ""
          local:                   true
  ▼ semanticId:
    ▼ keys:
      ▼ 0:
          idType:                  "IdShort"
          type:                    "Submodel"
          value:                   "ControlComponentTemplate"
          local:                   false
  ▼ identification:
      idType:                      "Custom"
      id:                          "ControlComponent"
    idShort:                       "ControlComponent"
    kind:                          "Instance"
    dataSpecification:             []
  ▼ modelType:
      name:                        "Submodel"
    embeddedDataSpecifications:    []
  ▼ submodelElements:
    ▼ Occupy:
      ▼ parent:
        ▼ keys:
          ▼ 0:
              idType:              "Custom"
              type:                "Submodel"
              value:               "ControlComponent"
              local:               true
        inoutputVariables:         []
        idShort:                   "Occupy"
        kind:                      "Instance"
        outputVariables:           []
      ▼ invokable:
```

Figure 19: BaSyx-AAS example of the “Control Component” submodel as JSON file [1]

To collect data from the shopfloor level, data update programs can be setup in BaSyx to collect data on the AAS level or to get data directly from a virtualized related object (e.g., from

[1] Asset Administration Shell JSON-Serialization in Web-Browser

"Control Component" or "Device Integration" object). The AAS and the objects in BaSyx are working with the concept of the virtual automation bus (VAB) [7], so that collected data in this bus can be used to update AAS but also to store this data into several databases, using the BaSyx-Connectors.

The connection towards the shop floor device can be done with a "Control Component" or also "Device Integration" object in the runtime. Therefore, an adapter like OPC UA must be integrated into these objects, to process events and to get data from the asset. In this place, the integration of a mapping or configuration file was very helpful for the flexible integration of the Smart Factory prototype, so that events towards the AAS submodel can be translated into the related OPC UA node and to transform given parameters into the suitable OPC UA parameter set.

4.2 Pilot setup with integration of existing legacy systems

The prototypical implementation needs to consider the integration of legacy systems of the Smart Factory testbed environment. Related to the holonic approach of MAS4AI, legacy systems are categorized into a holonic level.

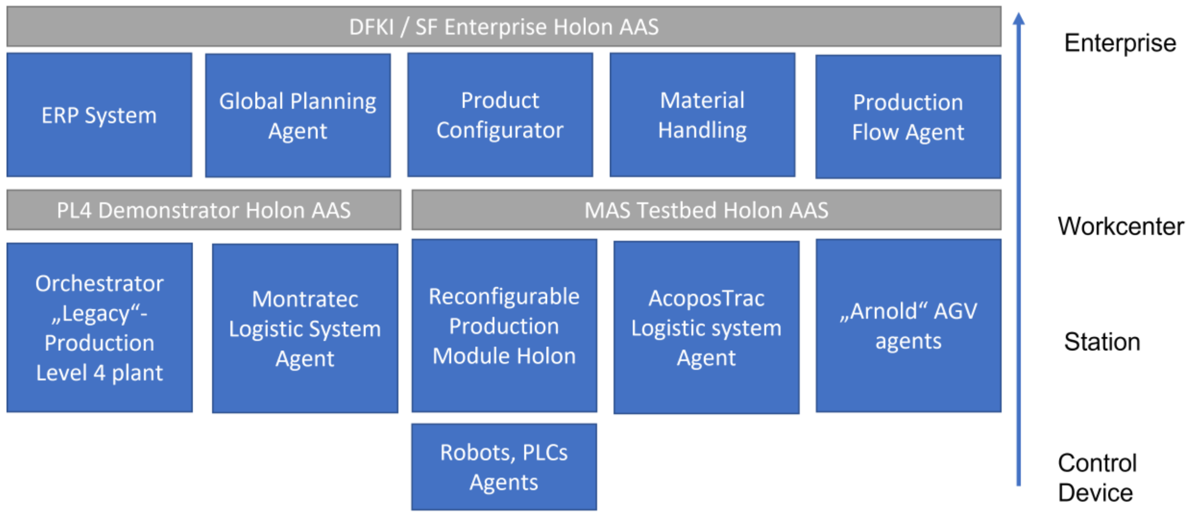

DFKI / SF Enterprise Holon AAS
ERP System
Global Planning Agent
Product Configurator
Material Handling
Production Flow Agent
PL4 Demonstrator Holon AAS
MAS Testbed Holon AAS
Orchestrator „Legacy"- Production Level 4 plant
Montratec Logistic System Agent
Reconfigurable Production Module Holon
AcoposTrac Logistic system Agent
„Arnold" AGV agents
Robots, PLCs Agents
Enterprise
Workcenter
Station
Control Device

Figure 20: Pilot setup with integrated legacy systems as holons

At enterprise level, several software components should be considered for integration. These consist of ERP system, the global planning component, the product configuration for setup of suitable recipes for product orders, the material handling, and the production flow unit, which has the possibility to orchestrate the PL4 demonstrator environment. These services should be integrated under a common factory holon, interacting with related agent types.

On the level of production lines, a holon could be setup for the PL4 demonstrator holon which controls the production modules of this resource holon as well as the shuttle transport system. This holon already considered the integration of these legacy systems with the proposed iterative prototyping approach. Another holon for further testbed validations is currently setup and comes with a reconfigurable production module setup and shuttle-based transport system integration. These components should also be integrated in the MAS4AI evaluation of the Smart Factory testbed environment.

The legacy assets itself, represented with PLC devices, should be integrated by using middleware solutions like BaSyx.

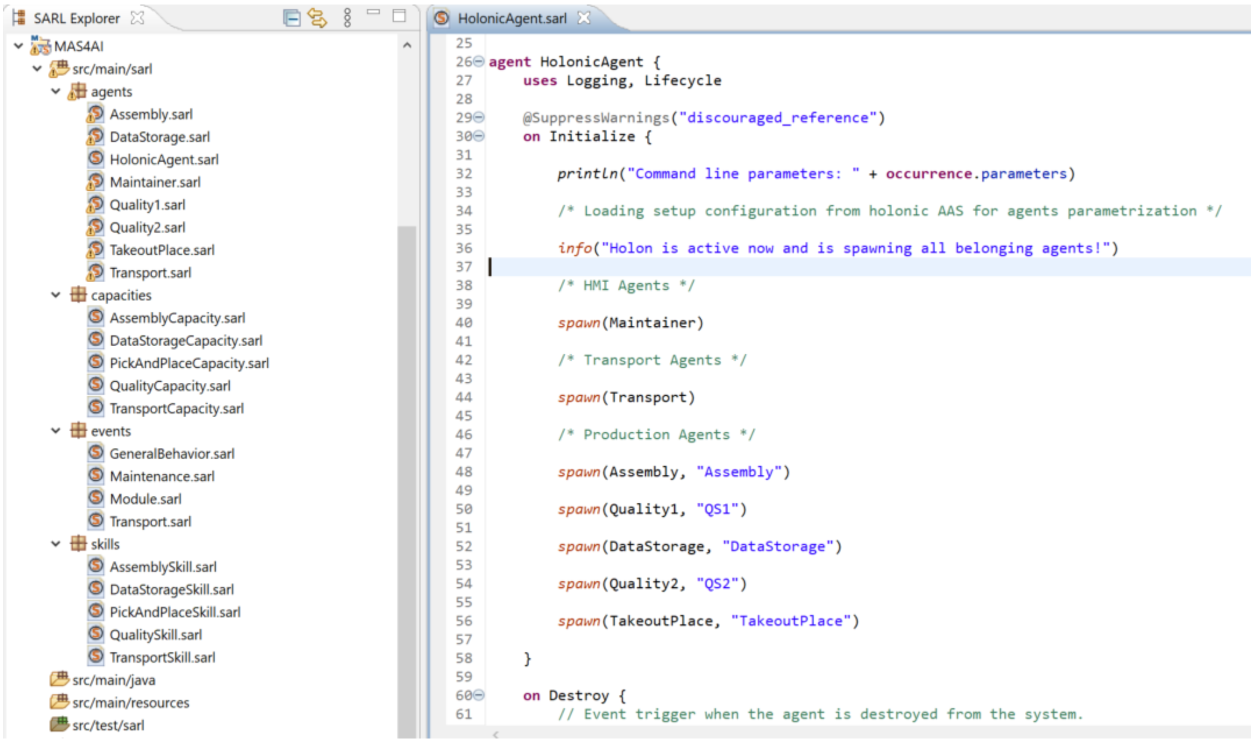
4.3 Execution and validation of the individual agents

The implementation results of the first Smart Factory demonstrator example focus on the setup of the MAS system by using the MAS framework Janus which is based on the agent-oriented programming language SARL [8] and the Data / AAS layer of the BaSyx-Middleware. As presented in the prototypical development in the previous chapter, the iterative integrated components based on the MAS4AI framework setup as proposed in WP2.

Static holon	Pattern for static holons, of the MAS4AI approach. The static holon takes care about initial setup and spawning of related agents. This object represents the structure for all lower-level static holons and manage the lifecycle of all related agents inside. The static holon thus can start a lifecycle for agent types, for example new resource agents.
Agent-Types	Resource-Agents like agents for production modules as well as for transport assets are considered. These agents can connect to the assets AAS and perform actions and read data from the asset. An HMI agent has also been considered for maintenance purpose, but not implemented with an HMI interface.

Table 16: Implemented agent patterns inside the prototype

In Janus, a holonic agent is setup to perform all related resource agents for the Smart Factory use case. In this case, the HMI-agent, and the resource agents and transport agent spawns with a unique identifier.


SARL Explorer

```
MAS4AI
  src/main/sarl
    agents
      Assembly.sarl
      DataStorage.sarl
      HolonicAgent.sarl
      Maintainer.sarl
      Quality1.sarl
      Quality2.sarl
      TakeoutPlace.sarl
      Transport.sarl
    capacities
      AssemblyCapacity.sarl
      DataStorageCapacity.sarl
      PickAndPlaceCapacity.sarl
      QualityCapacity.sarl
      TransportCapacity.sarl
    events
      GeneralBehavior.sarl
      Maintenance.sarl
      Module.sarl
      Transport.sarl
    skills
      AssemblySkill.sarl
      DataStorageSkill.sarl
      PickAndPlaceSkill.sarl
      QualitySkill.sarl
      TransportSkill.sarl
  src/main/java
  src/main/resources
  src/test/sarl
```

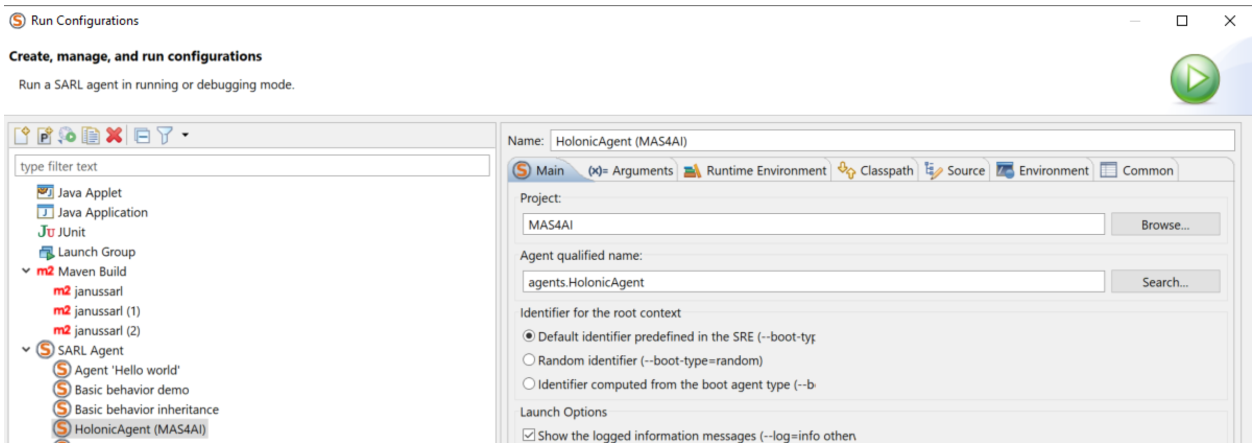
HolonicAgent.sarl

```

agent HolonicAgent {
    uses Logging, Lifecycle

    @SuppressWarnings("discouraged_reference")
    on Initialize {

        println("Command line parameters: " + occurrence.parameters)

        /* Loading setup configuration from holonic AAS for agents parametrization */

        info("Holon is active now and is spawning all belonging agents!")

        /* HMI Agents */

        spawn(Maintainer)

        /* Transport Agents */

        spawn(Transport)

        /* Production Agents */

        spawn(Assembly, "Assembly")

        spawn(Quality1, "QS1")

        spawn(DataStorage, "DataStorage")

        spawn(Quality2, "QS2")

        spawn(TakeoutPlace, "TakeoutPlace")

    }

    on Destroy {
        // Event trigger when the agent is destroyed from the system.
```

Figure 21: Static holonic agent in the prototype setup [2]

The start of the holonic agent can be done by using the Run configuration setting of the Janus Framework [9], which enables the start by using Janus tools and the default Java command line.

[2] SARL IDE (Project Setup and Holonic Agent)



Run Configurations

Create, manage, and run configurations

Run a SARL agent in running or debugging mode.

type filter text

- Java Applet
- Java Application
- JUnit
- Launch Group
- Maven Build
  - janussarl
  - janussarl (1)
  - janussarl (2)
- SARL Agent
  - Agent 'Hello world'
  - Basic behavior demo
  - Basic behavior inheritance
  - HolonicAgent (MAS4AI)

Name: HolonicAgent (MAS4AI)

Main | Arguments | Runtime Environment | Classpath | Source | Environment | Common

Project: MAS4AI — Browse...

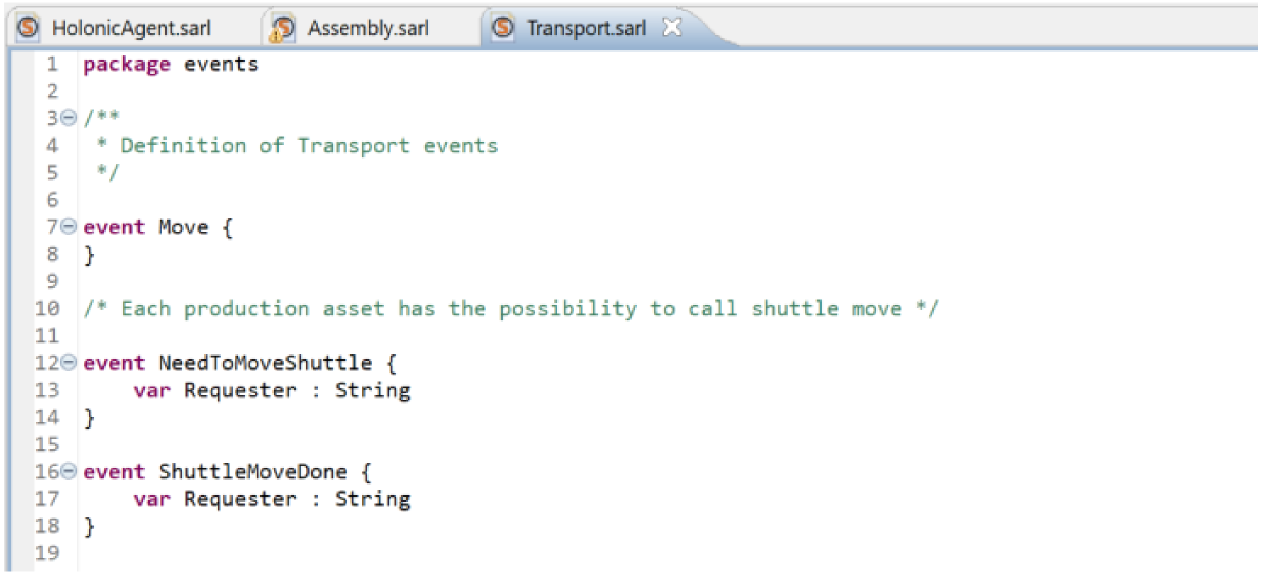
Agent qualified name: agents.HolonicAgent — Search...

Identifier for the root context
- (●) Default identifier predefined in the SRE (--boot-ty
- ( ) Random identifier (--boot-type=random)
- ( ) Identifier computed from the boot agent type (--b

Launch Options
- [✓] Show the logged information messages (--log=info othe

Figure 22: Janus Launch Configuration of holonic agent [3]

Each agent in this context can setup his own behavior and integrate capabilities and skills of the Janus framework. Each of these objects can be defined as own Janus object. The communication between agents can be processed by using the internal Janus message system with predefined events [10].



HolonicAgent.sarl | Assembly.sarl | Transport.sarl

```
package events

/**
 * Definition of Transport events
 */

event Move {
}

/* Each production asset has the possibility to call shuttle move */

event NeedToMoveShuttle {
	var Requester : String
}

event ShuttleMoveDone {
	var Requester : String
}

```

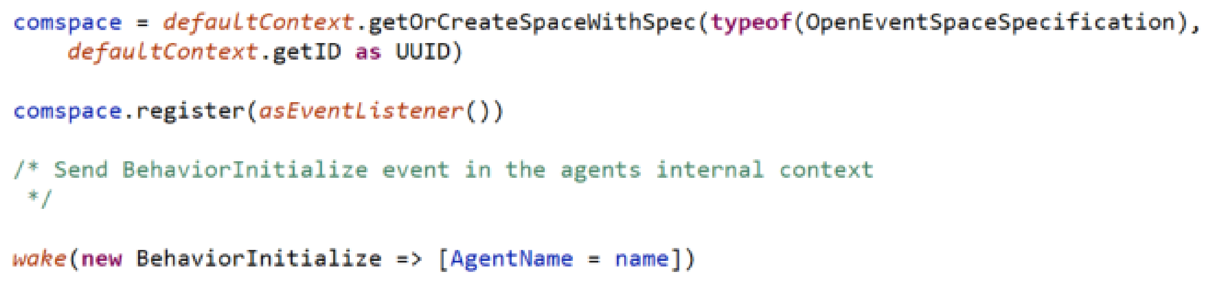
Figure 23: Transport events in SARL [4]

An event can consist of internal syntax and of variables and objects. Events can be sent inside an agent, for example to start internal behavior, but can also be used to communication with

[3] SARL IDE (Janus Launch Configuration)

[4] SARL IDE (Transport events)

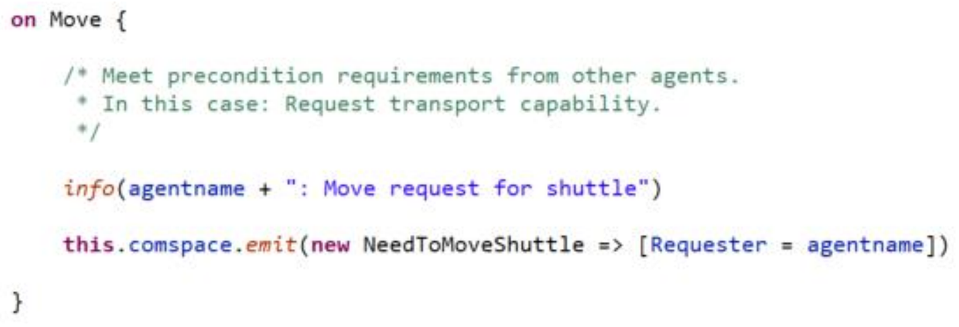
other agents, if they have joined a predefined message channel [10]. In case of distributed MAS4AI agent deployment, the command line for subscription to internal message channels must be replaced with a callback to an integrated external messaging system, like Apache Kafka.



```
comspace = defaultContext.getOrCreateSpaceWithSpec(typeof(OpenEventSpaceSpecification),
    defaultContext.getID as UUID)

comspace.register(asEventListener())

/* Send BehaviorInitialize event in the agents internal context
 */

wake(new BehaviorInitialize => [AgentName = name])
```

Figure 24: Event listener of Janus message channels [5]

An agent furthermore has the possibility to react to events in the scope of the agent definition or inside of active behaviors. Inside the Janus framework, these sorts of events are also needed, if an external messaging system should be used.



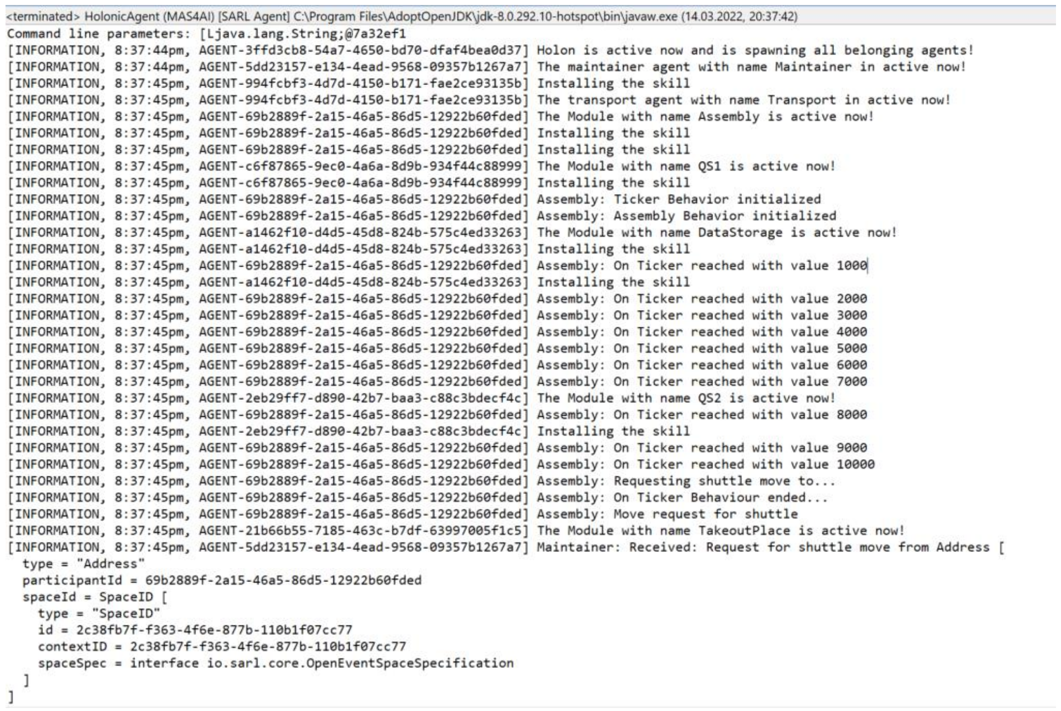
```
on Move {

    /* Meet precondition requirements from other agents.
     * In this case: Request transport capability.
     */

    info(agentname + ": Move request for shuttle")

    this.comspace.emit(new NeedToMoveShuttle => [Requester = agentname])

}
```

Figure 25: Event-Trigger inside of Janus agents [6]

The communication between agents and inside of an agent object is presented in the Figure 26, showing a communication of several agents, that have been started inside an holonic agent.

[5] SARL IDE (Event listener)

[6] SARL IDE (Event-Trigger inside of agents)



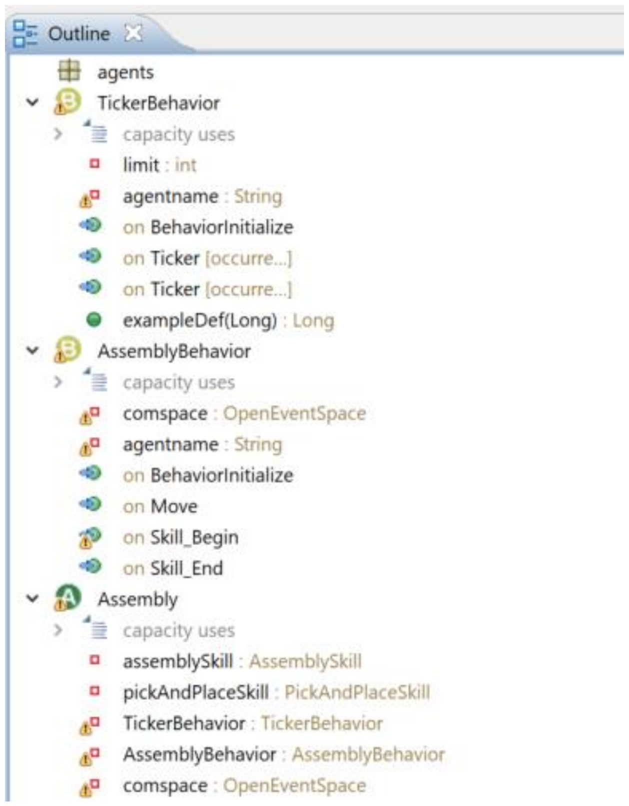
```
<terminated> HolonicAgent (MAS4AI) [SARL Agent] C:\Program Files\AdoptOpenJDK\jdk-8.0.292.10-hotspot\bin\javaw.exe (14.03.2022, 20:37:42)
Command line parameters: [Ljava.lang.String;@7a32ef1
[INFORMATION, 8:37:44pm, AGENT-3ffd3cb8-54a7-4650-bd70-dfaf4bea0d37] Holon is active now and is spawning all belonging agents!
[INFORMATION, 8:37:44pm, AGENT-5dd23157-e134-4ead-9568-09357b1267a7] The maintainer agent with name Maintainer in active now!
[INFORMATION, 8:37:45pm, AGENT-994fcbf3-4d7d-4150-b171-fae2ce93135b] Installing the skill
[INFORMATION, 8:37:45pm, AGENT-994fcbf3-4d7d-4150-b171-fae2ce93135b] The transport agent with name Transport in active now!
[INFORMATION, 8:37:45pm, AGENT-69b2889f-2a15-46a5-86d5-12922b60fded] The Module with name Assembly is active now!
[INFORMATION, 8:37:45pm, AGENT-69b2889f-2a15-46a5-86d5-12922b60fded] Installing the skill
[INFORMATION, 8:37:45pm, AGENT-69b2889f-2a15-46a5-86d5-12922b60fded] Installing the skill
[INFORMATION, 8:37:45pm, AGENT-c6f87865-9ec0-4a6a-8d9b-934f44c88999] The Module with name QS1 is active now!
[INFORMATION, 8:37:45pm, AGENT-c6f87865-9ec0-4a6a-8d9b-934f44c88999] Installing the skill
[INFORMATION, 8:37:45pm, AGENT-69b2889f-2a15-46a5-86d5-12922b60fded] Assembly: Ticker Behavior initialized
[INFORMATION, 8:37:45pm, AGENT-69b2889f-2a15-46a5-86d5-12922b60fded] Assembly: Assembly Behavior initialized
[INFORMATION, 8:37:45pm, AGENT-a1462f10-d4d5-45d8-824b-575c4ed33263] The Module with name DataStorage is active now!
[INFORMATION, 8:37:45pm, AGENT-a1462f10-d4d5-45d8-824b-575c4ed33263] Installing the skill
[INFORMATION, 8:37:45pm, AGENT-69b2889f-2a15-46a5-86d5-12922b60fded] Assembly: On Ticker reached with value 1000
[INFORMATION, 8:37:45pm, AGENT-a1462f10-d4d5-45d8-824b-575c4ed33263] Installing the skill
[INFORMATION, 8:37:45pm, AGENT-69b2889f-2a15-46a5-86d5-12922b60fded] Assembly: On Ticker reached with value 2000
[INFORMATION, 8:37:45pm, AGENT-69b2889f-2a15-46a5-86d5-12922b60fded] Assembly: On Ticker reached with value 3000
[INFORMATION, 8:37:45pm, AGENT-69b2889f-2a15-46a5-86d5-12922b60fded] Assembly: On Ticker reached with value 4000
[INFORMATION, 8:37:45pm, AGENT-69b2889f-2a15-46a5-86d5-12922b60fded] Assembly: On Ticker reached with value 5000
[INFORMATION, 8:37:45pm, AGENT-69b2889f-2a15-46a5-86d5-12922b60fded] Assembly: On Ticker reached with value 6000
[INFORMATION, 8:37:45pm, AGENT-69b2889f-2a15-46a5-86d5-12922b60fded] Assembly: On Ticker reached with value 7000
[INFORMATION, 8:37:45pm, AGENT-2eb29ff7-d890-42b7-baa3-c88c3bdecf4c] The Module with name QS2 is active now!
[INFORMATION, 8:37:45pm, AGENT-69b2889f-2a15-46a5-86d5-12922b60fded] Assembly: On Ticker reached with value 8000
[INFORMATION, 8:37:45pm, AGENT-2eb29ff7-d890-42b7-baa3-c88c3bdecf4c] Installing the skill
[INFORMATION, 8:37:45pm, AGENT-69b2889f-2a15-46a5-86d5-12922b60fded] Assembly: On Ticker reached with value 9000
[INFORMATION, 8:37:45pm, AGENT-69b2889f-2a15-46a5-86d5-12922b60fded] Assembly: On Ticker reached with value 10000
[INFORMATION, 8:37:45pm, AGENT-69b2889f-2a15-46a5-86d5-12922b60fded] Assembly: Requesting shuttle move to...
[INFORMATION, 8:37:45pm, AGENT-69b2889f-2a15-46a5-86d5-12922b60fded] Assembly: On Ticker Behaviour ended...
[INFORMATION, 8:37:45pm, AGENT-69b2889f-2a15-46a5-86d5-12922b60fded] Assembly: Move request for shuttle
[INFORMATION, 8:37:45pm, AGENT-21b66b55-7185-463c-b7df-63997005f1c5] The Module with name TakeoutPlace is active now!
[INFORMATION, 8:37:45pm, AGENT-5dd23157-e134-4ead-9568-09357b1267a7] Maintainer: Received: Request for shuttle move from Address [
  type = "Address"
  participantId = 69b2889f-2a15-46a5-86d5-12922b60fded
  spaceId = SpaceID [
    type = "SpaceID"
    id = 2c38fb7f-f363-4f6e-877b-110b1f07cc77
    contextID = 2c38fb7f-f363-4f6e-877b-110b1f07cc77
    spaceSpec = interface io.sarl.core.OpenEventSpaceSpecification
  ]
]
```

Figure 26: Interaction Example of communication between and inside Janus agents [7]

After spawning the related agents, their initial behavior is started and related skills, for example for assembly, are initialized. The request for transport can be performed from a resource agent, for example from the assembly agent, towards the transport agent.

Agents can be modelled by using capacities with skills [8]. The resource agent for the "Storage and Assembly" production module consists of an assembly behavior that implements an assembly skill and a skill for pick and place to get the product from the shuttle transporter inside the module. The precondition which must be met, before the skill is executed, is to call the transport, resulting in an own behavior trigger inside the agent.

[7] SARL IDE (Console-Log as interaction example for agent communication)

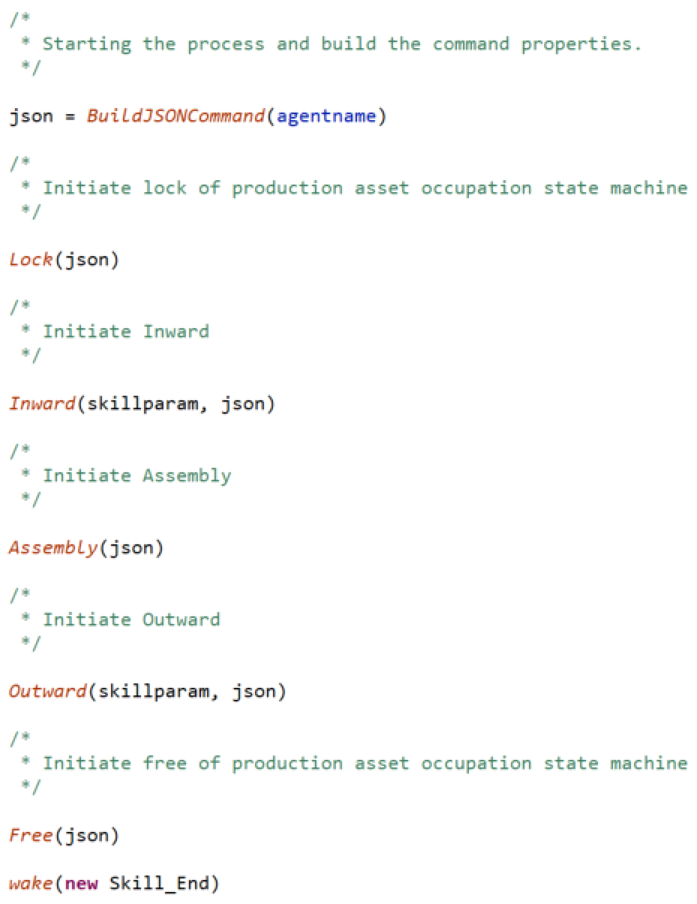

Outline

```
agents
TickerBehavior
    capacity uses
    limit : int
    agentname : String
    on BehaviorInitialize
    on Ticker [occurre...]
    on Ticker [occurre...]
    exampleDef(Long) : Long
AssemblyBehavior
    capacity uses
    comspace : OpenEventSpace
    agentname : String
    on BehaviorInitialize
    on Move
    on Skill_Begin
    on Skill_End
Assembly
    capacity uses
    assemblySkill : AssemblySkill
    pickAndPlaceSkill : PickAndPlaceSkill
    TickerBehavior : TickerBehavior
    AssemblyBehavior : AssemblyBehavior
    comspace : OpenEventSpace
```

Figure 27: Agent behavior and skill implementation in Janus with SARL [8]

The implementation of a skill can be done inside an agent object and be used inside a behavior [11], to execute several commands of this skill in a sequenced manner. For example, the execution of the assembly skill with the shuttle system to get the workpiece carrier for the product must always follow a predefined process. In Figure 28, the assembly skill of the resource agent for the assembly production module is presented.

[8] SARL IDE (SARL Objects for agent behavior and skills)



```
/*
 * Starting the process and build the command properties.
 */

json = BuildJSONCommand(agentname)

/*
 * Initiate lock of production asset occupation state machine
 */

Lock(json)

/*
 * Initiate Inward
 */

Inward(skillparam, json)

/*
 * Initiate Assembly
 */

Assembly(json)

/*
 * Initiate Outward
 */

Outward(skillparam, json)

/*
 * Initiate free of production asset occupation state machine
 */

Free(json)

wake(new Skill_End)
```

Figure 28: Implemented assembly skill functions inside of assembly behavior of a Janus agent [9]

It is possible to submit several parametrizations for the call of the related skills. Furthermore, for agents which interact with assets that are represented with an AAS, it is necessary to build up command messages, for example in JSON, which are sent to the AAS methods as parameters. The commands are executed towards the assets AAS by using the JSON, which can be prepared individually. The communication pattern inside the skill is customizable and changeable, so it is possible to keep a predefined skill sequence inside of an agent's behavior and to change the skill implementation with a modified or a completely new object, without changing the basic agent communication or behavior pattern in the source code in a general way. This allows individual development and customization, without many changes of basic objects like agents or behaviors. The AAS communication inside the skill objects is furthermore part of the MAS4AI framework interface setup, as well as the integration of semantic requests, using an RDF-Store.

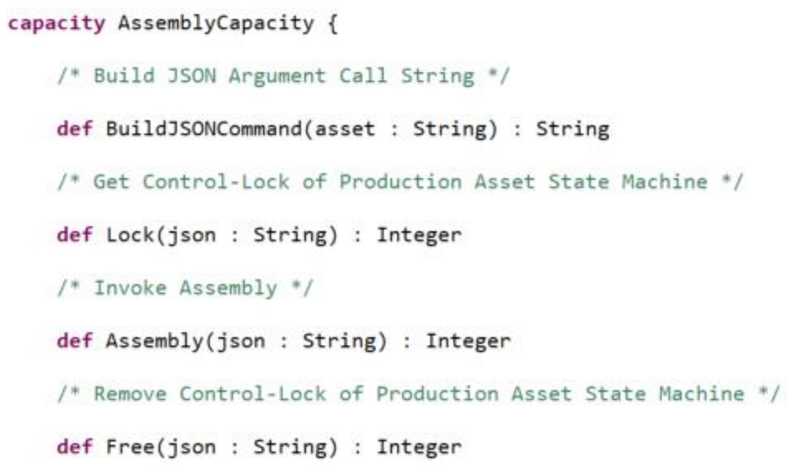
[9] SARL IDE (Skill usage inside of agent behavior)

4.4 Execution and validation with the whole MAS4AI system

Related to the Smart Factory use case, the MAS4AI framework focused in the current state on the integration between the MAS component and the AAS / Data component, which has been built on implementations with the BaSyx-Middleware.

The main concept for interaction of the MAS component with other elements of the MAS4AI framework can be seen in the concept of changeable capacities and skills, following the concept in the Janus framework. The definition of capacities allows an abstract method definition, that can be implemented from various skills in a different way. This allows the definition of software patterns, which can be implemented from agents, without predefinition of the implementation itself, which is integrated very often in an individual manner.

This concept can also be used for the integration with other MAS4AI components, like RDF-Store or the integration of external message channels with Apache Kafka. This allows the setup of various agent templates with a predefined behavior but encapsulates the concrete technology-related interface implementation inside the skill. This approach enables a flexible setup and possibilities for Use Case-specific customization in the MAS component of the MAS4AI framework and can also be realized in other frameworks if this software pattern can be implemented there. Another usage of this pattern can be evaluated in the Jade Framework, by using the standard Java interface pattern, which build the basis of the Janus capacity [12], but without domain-specific skill setup within agents.

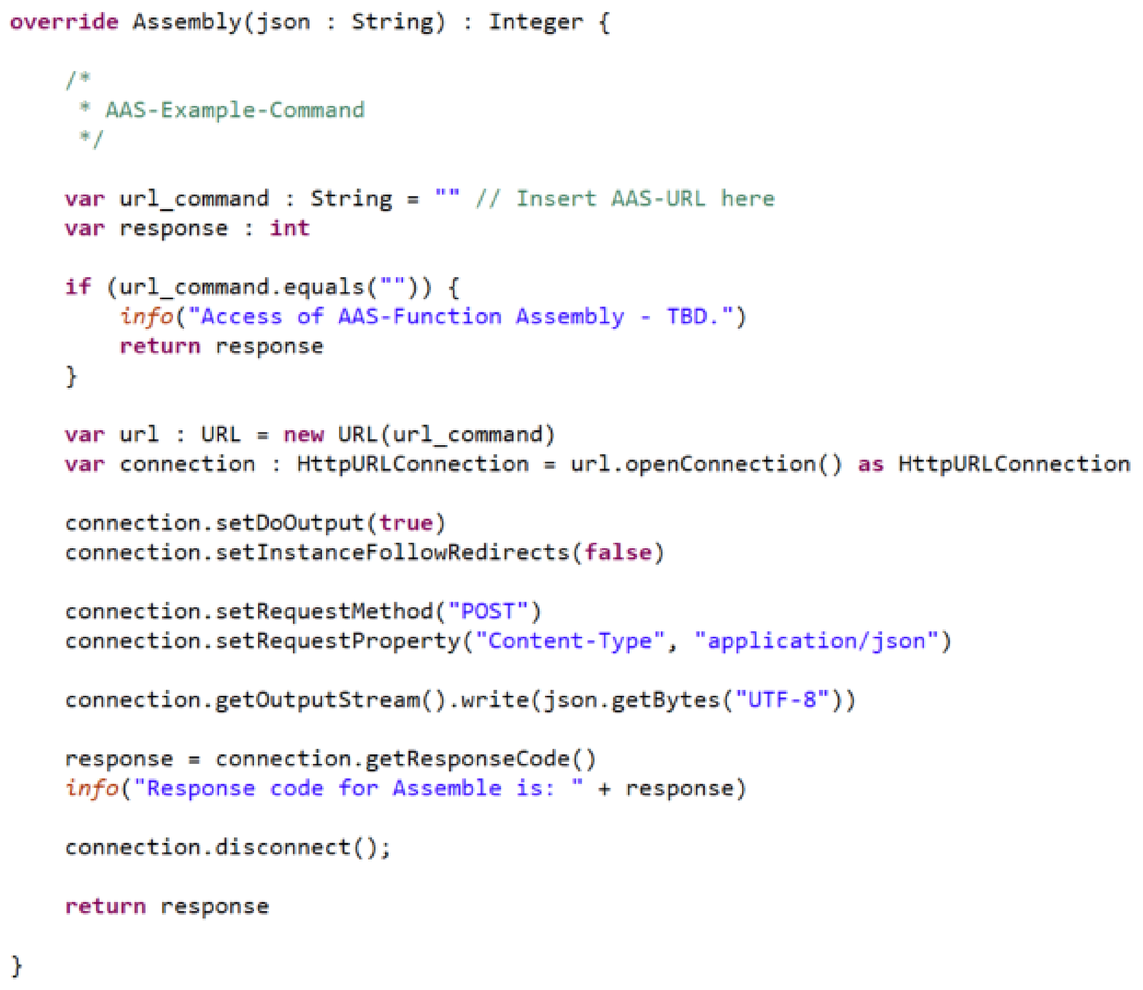


```
capacity AssemblyCapacity {

    /* Build JSON Argument Call String */

    def BuildJSONCommand(asset : String) : String

    /* Get Control-Lock of Production Asset State Machine */

    def Lock(json : String) : Integer

    /* Invoke Assembly */

    def Assembly(json : String) : Integer

    /* Remove Control-Lock of Production Asset State Machine */

    def Free(json : String) : Integer
```

Figure 29: Assembly capacity in SARL [10]

[10] SARL IDE (Capacity Example)

Inside of a skill, the given capacity can be extended with a corresponding implementation. The skill can request the AAS registry of the BaSyx-Middleware to get the related asset and to connect to the AAS submodel, processing a prebuild JSON-command to AAS methods.



```
override Assembly(json : String) : Integer {

    /*
     * AAS-Example-Command
     */

    var url_command : String = "" // Insert AAS-URL here
    var response : int

    if (url_command.equals("")) {
        info("Access of AAS-Function Assembly - TBD.")
        return response
    }

    var url : URL = new URL(url_command)
    var connection : HttpURLConnection = url.openConnection() as HttpURLConnection

    connection.setDoOutput(true)
    connection.setInstanceFollowRedirects(false)

    connection.setRequestMethod("POST")
    connection.setRequestProperty("Content-Type", "application/json")

    connection.getOutputStream().write(json.getBytes("UTF-8"))

    response = connection.getResponseCode()
    info("Response code for Assemble is: " + response)

    connection.disconnect();

    return response

}
```

Figure 30: Assembly Skill in SARL [11]

The communication related part of the MAS framework thus can be built of reusable communication patterns for several protocols, for example to interact with a AAS with a REST-interface or directly by using OPC UA. For further integration of an RDF-Store or Apache Kafka, it is also possible to provide suitable connectors inside a skill.

[11] SARL IDE (Implemented Skill Example)

4.5 Summarized results

For the prototypical implementation of the Smart Factory testbed environment with the PL4 line and the MAS4AI framework evaluation, the methodical approach for the iterative development provided the possibility to reach the defined development goals of each iteration. Thereby, the system setup and evaluation of the overall framework setup, consisting of MAS4AI framework components like the MAS-System itself, as well as the AAS / Data layer as representation of the manufacturing environment could be enhanced with new functionalities.

The status of implemented agent types in the testbed environment, related to the described requirements, are presented in Table 31.

Agent	Input	Output	Status
Production module agent	Raw material	Processed material	Successful integration
Transport agent	Transported good	Transported good	Successful integration
Quality Inspection agent	Product	Product (+Quality assessment)	In Progress (AAS prepared)
Energy agent	Energy	-	In Progress (AAS prepared)

Figure 31: Status of integrated agents in the testbed environment

The preparation of AAS with uniform interfaces to interact with the manufacturing environment has been evaluated as essential for integration and communication with the MAS component of the proposed MAS4AI framework. The interaction itself has been successfully evaluated with AAS and Registry, which is built with the BaSyx-Middleware and hosted with REST-Interface. A point which is very important for future integration is an event-based subscription mechanism if agents execute tasks of assets, because of the asynchronous execution by using the AAS it would be necessary to iterative load the AAS with its properties to get information about successful or failure during the execution. Therefore, topics of Apache Kafka or MQTT could be used.

Another important point of the current evaluation is the usability of agent patterns in the MAS component for several agent types. The resource agent type has been adapted for several agents of production modules as well as for the transportation asset. The holonic agent controls the lifecycle of all related agent types. The skill approach enables the possibility of an abstract integration in agent types and behaviors and is exchangeable, for example to support a communication with an AAS with REST. The integration of AAS for holonic agents and their agent types will be integrated as well as skills for communication with the RDF-Store, the integration of message channel systems like Apache Kafka and HMI-Devices.

5 Conclusion and future research

The implementation of the MAS4AI prototype has shown, that the asset representation as AAS is essential for MAS connection and interaction. The possibility to get data from assets and to interact with them with predefined interfaces, enables a better integration for MAS-Systems. Furthermore, the prototype showed, that the execution of a production flow with components of the MAS4AI framework is possible. The iterative development method thus makes it possible to build up the framework in a modular way and to enhance functionalities.

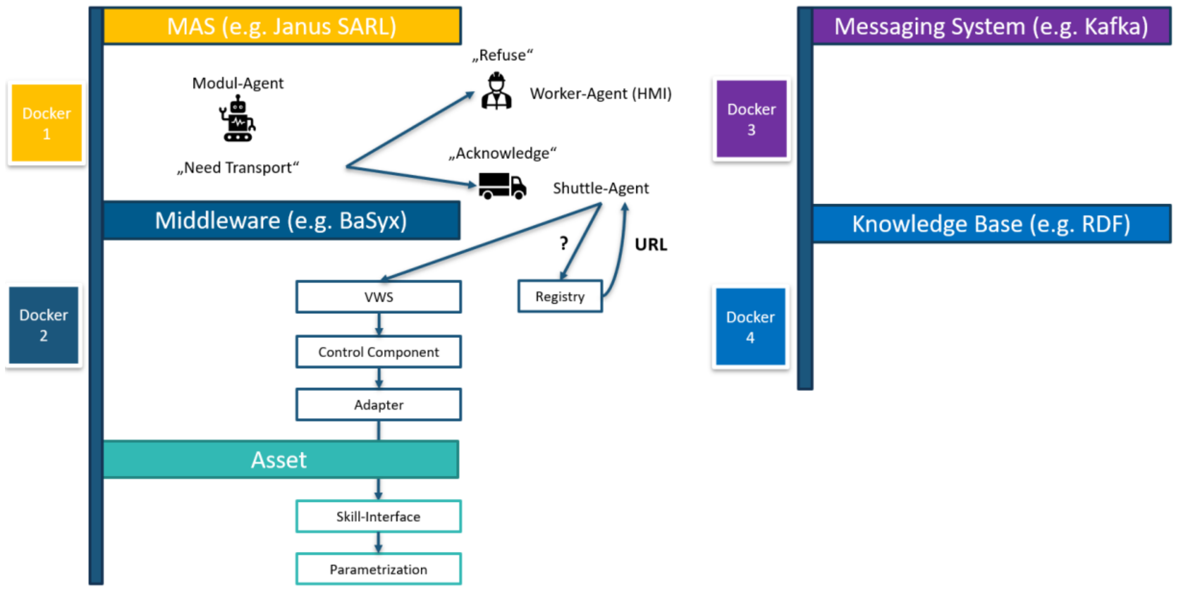


Figure 32: Upcoming iteration built of Smart Factory MAS4AI prototype

The upcoming development iterations will focus more on a distributed deployment of MAS4AI components, by using Docker technology for each component. Therefore, predefined interfaces with API can help to build up single components in a way, that they are independent of the used solution, for example if another technology should be used than the BaSyx-Middleware. In the next steps, the messaging system component and the knowledge base should be integrated and the setup of the already used components more developed in terms of customizable patterns. Furthermore, the integration of OPC UA in skills for direct AAS communication from agents will be considered. The integration and evaluation of more agent types, as described in the use case requirements, are also focused for upcoming prototyping. The integration and usage of AI agents, for example for planning or machine learning, into the shown prototypical implemented MAS4AI framework is also planned for upcoming development iterations.

6 Literature

[1] Ruskowski, Martin, et al. "Production Bots für Production Level 4: Skill-basierte Systeme für die Produktion der Zukunft." atp magazin 62.9 (2020): 62-71.

[2] Eclipse BaSyx™ (2022). Available: https://projects.eclipse.org/projects/technology.basyx, Accessed: 2022-04-13.

[3] Registry Component (2022). Available: https://wiki.eclipse.org/BaSyx_/_Documentation_/_Components_/_Registry, Accessed: 2022-04-13.

[4] AAS Server Component (2022). Available: https://wiki.eclipse.org/BaSyx_/_Documentation_/_Components_/_AAS_Server, Accessed: 2022-04-13.

[5] BaSys 4.0 control and group components (2022). Available: https://wiki.eclipse.org/BaSyx_/_Documentation_/_ControlComponent, Accessed: 2022-04-13.

[6] BaSyx Device integration (2022). Available: https://wiki.eclipse.org/BaSyx_Device_integration, Accessed: 2022-04-13.

[7] Virtual Automation Bus (2022). Available: https://wiki.eclipse.org/BaSyx_/_Documentation_/_VAB, Accessed: 2022-04-13.

[8] Rodriguez, Sebastian, Nicolas Gaud, and Stéphane Galland. "SARL: a general-purpose agent-oriented programming language." 2014 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT). Vol. 3. IEEE, 2014.

[9] Create a SARL Launch Configuration (2022). Available: http://www.sarl.io//docs/official/gettingstarted/RunSARLAgentEclipse.html#1-create-a-sarl-launch-configuration, Accessed: 2022-04-13.

[10] Event Reference (2022). Available: http://www.sarl.io//docs/official/reference/Event.html, Accessed: 2022-04-13.

[11] Skill Reference (2022). Available: http://www.sarl.io//docs/official/reference/Skill.html, Accessed: 2022-04-13.

[12] Capacity Reference (2022). http://www.sarl.io//docs/official/reference/Capacity.html, Accessed: 2022-04-13.