ITEA 3

OPTIMUM

OPTimised Industrial IoT and Distributed Control Platform
for Manufacturing and Material Handling

Deliverable 2.4
Final Implementation of IIoT platform, API and SDK (2nd prototype)

Deliverable type:	Software
Deliverable reference number:	ITEA 16043 | D2.4
Related Work Package:	WP 2
Due date:	2021-04-30
Actual submission date:	2021-05-31
Responsible organisation:	University of Rostock
Editor:	Hannes Raddatz
Dissemination level:	Public
Revision:	Final | Version 1.01

Abstract:	Description of the IIoT-Platform functionality, its purpose in the context of the OPTIMUM project and implementation details with information about the MQTT and OPC UA APIs as well as the data model.

Keywords:	OPTIMUM IIoT Platform, MQTT, OPC UA, M2M

Table_head	Name 1 (partner)	Name 2 (partner)	Approval date (1 / 2)
Approval at WP level	Fabian Hölzke (URO)	Metin Tekkalmaz (ERSTE)	2021-05-05 / 2021-05-07
Veto Review	No Vetos		2021-05-31

Editor
Hannes Raddatz (University of Rostock)

Contributors
Metin Tekkalmaz (ERSTE)
Fabian Hölzke (University of Rostock)

Executive Summary

The Deliverable 2.4 targets the open-source publication of the IIoT-Platform software. This document provides a description of the IIoT-Platform implementation and information about its context in the OPTIMUM project. This includes details about:

- External OPC UA interface
- Internal MQTT interface
- OPTIMUM Data Model
- Implementation Details
- IIoT Platform in OPTIMUM Demonstrators

Table of Content

Figures

Tables

1 Introduction

The Industrial Internet of Things (IIoT)-Platform is one of the central components of the OPTIMUM architecture and enables machine-to-machine (M2M) communication for industry 4.0 applications. This document describes the IIoT-Platform in general, the information data model as well as its internal and external interfaces.

2 OPTIMUM Architecture

The OPTIMUM project targets to equip existing machines with additional functionality to allow for semi-autonomous functionality in the smart material handling and smart manufacturing domains. To achieve such functionalities, enhanced devices require location awareness, distributed control and M2M interfaces. A component-based software approach is used to accomplish this target. Devices like cranes that are capable of semi-autonomous functionality are categorized as Smart Material Handling Systems and contain in the minimal configuration an IIoT-Platform and Distributed Control Platform (DCP). Machines such as forklifts that are only upgraded with a location awareness function are called targets and contain the lightweight IIoT platform as their only component. Therefore, such devices provide only an OPC UA server and no MQTT interface.

3 IIoT Platform

The IIoT-Platform is responsible for non-deterministic [1]communication between machines and between human and machine. The human, called operator in the following, uses a Human-Machine-Interface (HMI) to interact with the machines. This HMI can be a computer, tablet or smartphone.

The platform has three main components: external OPC UA interface, internal MQTT interface and a device information data model. The combination of these components allows the IIoT Platform to forward OPC UA request from outside to other components of the same device via MQTT messages and vice versa. The data model of the platform can be access via OPC UA (read-only) and MQTT (only internal, write access). Further information on the components is presented in the following.

3.1 OPC UA Interface

The IIoT-Platform contains an OPC UA server that can be accessed by other devices in the same network that fulfill the security requirements (see section 3.4.1). Both versions of the IIoT-Platform (complete and lightweight) provide an OPC UA server that contains the data model described in section 3.3 which provides read-only access to the variable values. The complete IIoT-Platform for smart material handling systems and smart manufacturing contains additional device-specific method calls to invoke semi-autonomous and corresponding functions. These method calls can be triggered by either an operator's HMI or another machine that is equipped with an IIoT-Platform. Also, a conventional OPC UA client can be used but manual handling of the response node mechanism is required.

Response Node Mechanism

Every method invocation provokes an MQTT message to another component of the device. Since the response time of the component remains unknown, the IIoT-Platform responds to a method invocation immediately after checking the consistency of the request. The response contains information about a new variable in the data model of the IIoT-Platform that is unique for every method invocation. If the requesting device is interested in intermediate and final status information of its request, it needs to subscribe to this new variable. The IIoT-Platform will notify the requesting device about a status changes. The target component uses the MQTT interface to update the status information.

The MQTT interface can trigger the instantiation of an OPC UA client in a separate thread to retrieve information from another OPTIMUM device or any other OPC UA server. The OPC UA client will either read the variables in the data model of the targeted OPC UA server or invoke methods on the IIoT-Platform that will use the response node mechanism to provide intermediate and final request status information. See also section 3.3.2.2.

3.2 MQTT Interface

The MQTT interface maps internal to external requests and vice versa. It forwards requests that are received from other devices via OPC UA method invocations to components of the same device and transmits requests or answer payloads of internal components to external devices using OPC UA. The payload of the specific MQTT topics that contain the request and answer data is encoded using JSON. The following chapters provide a detailed description of

[1] The term non-deterministic relates to the assumption that in case of a delayed message no critical or safety issues are expected

the available topics, the mechanisms that use these interfaces and the corresponding JSON encoding schemas.

3.2.1 MQTT Topic iiot-common

A component of the same device can request data of another device in the network by sending a JSON encoded MQTT message to the MQTT interface of the IIoT-Platform using the MQTT topic "iiot-common". The OPC UA interface will be used to fetch this information by using a separate OPC UA client thread. A device's component can instruct the IIoT-Platform of the same device to read the requested data once or on a subscription basis until the component decides to cancel the subscription.

The same topic can be used to send requests or commands to components of other OPTIMUM devices. In this case, the IIoT-Plaforms of the requesting and target device are using a response mechanism developed in OPTIMUM project to send intermediate and the final response to the requesting component. Further information on this response mechanism can be found in sections 3.1, 3.3.2.1 and 3.3.2.2.

Table 1: MQTT Topic iiot-common

Comm. Protocol	MQTT
Topic Name	iiot-common
Published By	Device internal component
Subscribed By	IIoT-Platform
Description	Request data or send commands of/to another device by instructing IIoT-Platform to forward request via OPC UA
Message Payload with JSON data encoding **only in case of *read* command**	{"answer_topic" : <string>, "req_id" : <string>, "command" : <string>, "device" : <string>, "component" : <string>, "data" : [<string>,…], "subscription" : <string>}
Parameter Details	**answer_topic:** The response of the request is send to this MQTT topic **req_id:** arbitrary but unique identifier of the request **command:** • *"read":* one-time read or subscription of/to data of another device's IIoT-Platform • *"methodcall":* send a request or command to a component (e.g. DCP) of another device via the IIoT-Platforms of both devices **device:** URL of target device's IIoT-Platform or OPC UA server component: Name of target component on other device (e.g. *"iiot"*, *"dcp"*) **data:** depending on command

***Further details on DCP request structure are described in section 3.3.2.4**	• *"read":* Array of OPC UA variable NodeIDs • *"methodcall":* structure depended on target component, structure of **DCP*** request: {"dev_id":<string>, "cmd":<string>, "arguments":{key:value,…}}

Examples

The following data request via MQTT requests location coordinates (X,Y,Z) with a one-time read request from another device's IIoT-Platform, issued by DCP component (with escape characters for string parsing). The flow of data and sequence of invocations between components and devices is visualized in Figure 1:

MQTT Topic: "iiot-common"

Payload:
```
"{\"answer_topic\":\"dcp-common\",
  \"req_id\":\"dcp1\",
  \"command\":\"read\",
  \"device\":\"opc.tcp://iiot_platform-2:4840\",
  \"component\":\"iiot\",
  \"data\":[\"1042\",\"1043\",\"1056\"],
  \"subscription\":\"false\"
 }"
```

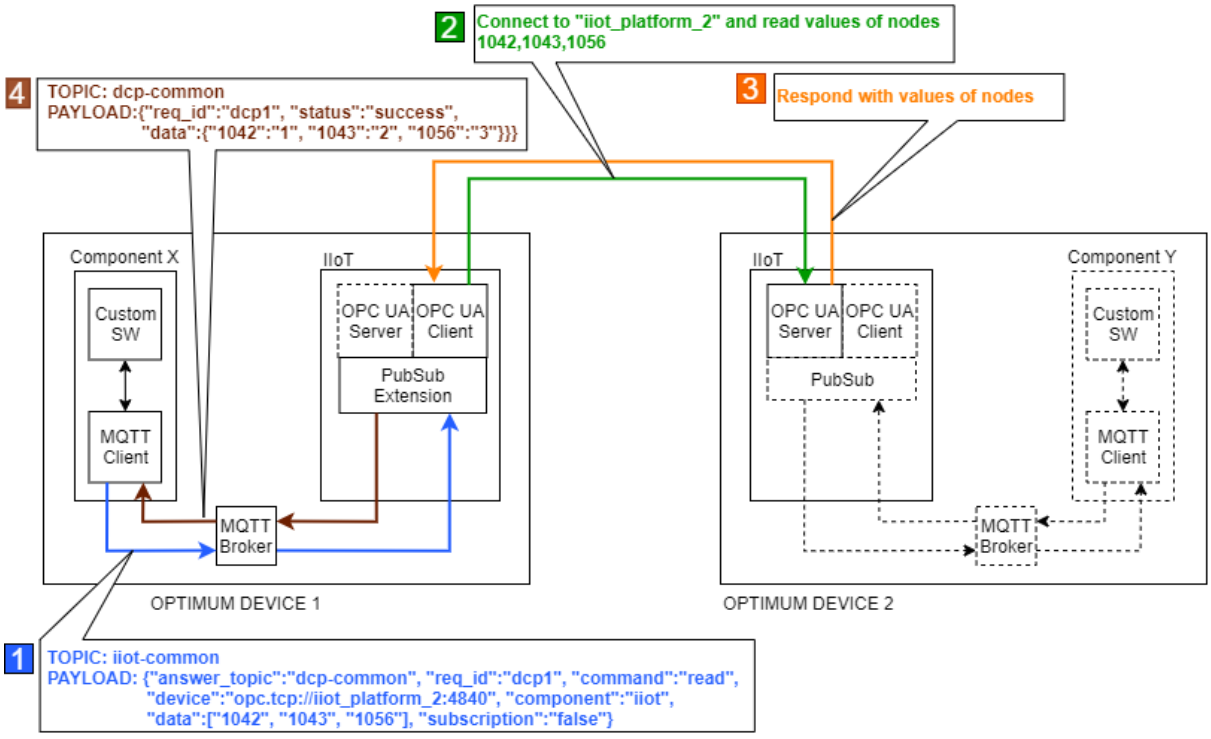

Figure 1: Message flow of read request

The next example presents a reservation request (see section 3.3.2.4.1) send from an internal DCP component to the DCP of another device via the IIoT-Platforms of both devices (with escape characters for parsing):

MQTT Topic: “iiot-common”

Payload:
```
"{\"answer_topic\":\"dcp-common\",
  \"req_id\":\"dcp2\",
  \"command\":\"methodcall\",
  \"device\":\"opc.tcp://iiot_platform-2:4840\",
  \"component\":\"dcp\",
  \"data\":{\"dev_id\":\"hmi1\",\"cmd\":\"reservation\"}
}"
```

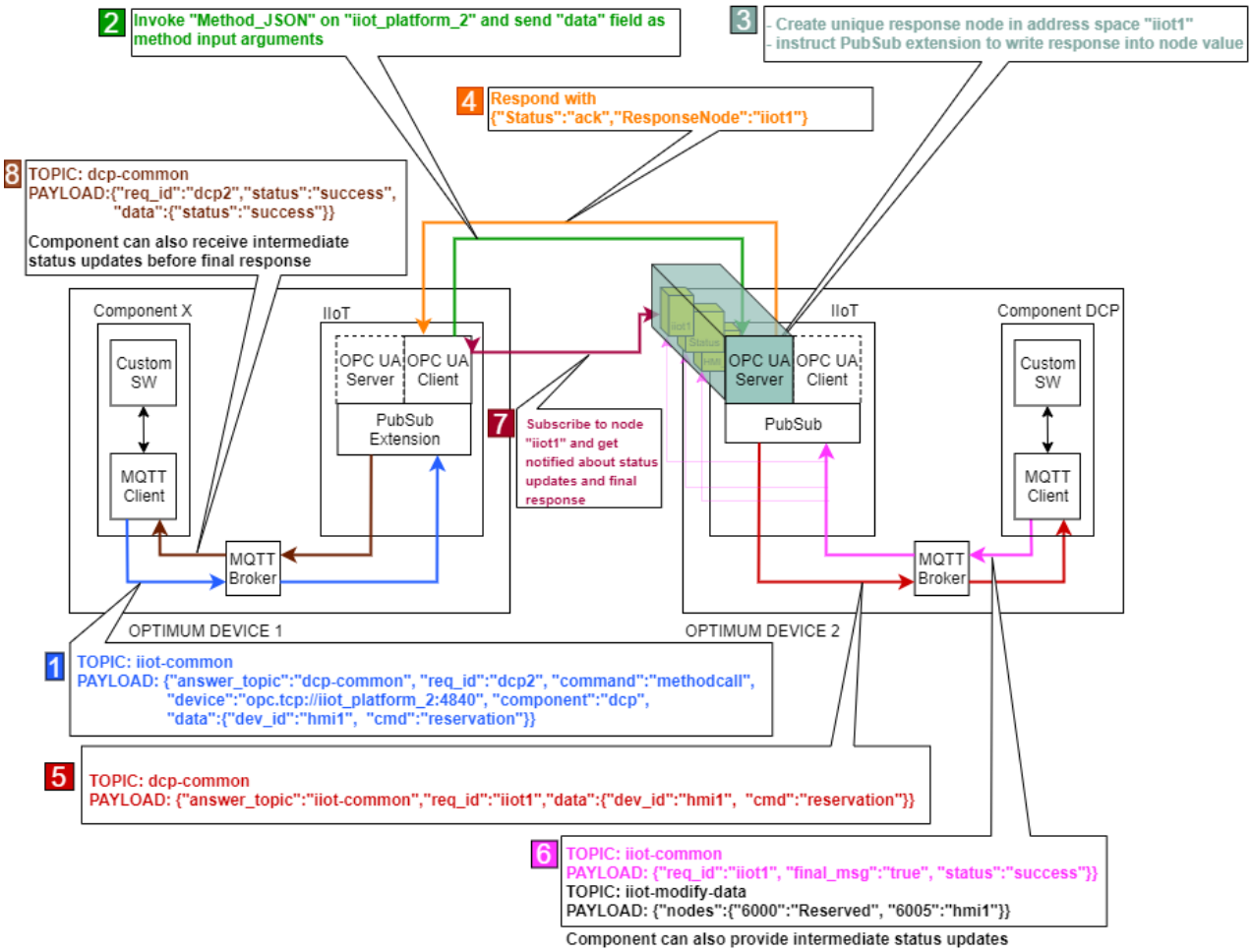

Figure 2: Message flow of method call

3.2.2 MQTT Topic iiot-modify-data

The information in the data model of the IIoT-Platform can be altered by the device components. The IIoT-Platform listens on the MQTT topic “iiot-modify-data” for this purpose and requires MQTT messages with the following syntax:

Table 2: MQTT topic iiot-modify-data

Comm. Protocol	MQTT
Topic Name	iiot-modify-data
Published By	Device internal component
Subscribed By	IIoT-Platform
Description	Modify variable values in the OPC UA data model of the IIoT-Platform
Payload	{"nodes":{"<NodeID>":<value>,…}}
Parameter Details	**<NodeID>:** OPC UA NodeID of variable to be modified **<value>:** new value of the variable, requires data type of target variable

Example

The following MQTT message will set new values for the coordinates X (NodeID 1042), Y (NodeID 1043) and Z (NodeID 1056) of the device:

MQTT Topic: `"iiot-modify-data"`

Payload: `"{\"nodes\":{\"1042\":2,\"1043\":20,\"1056\":200}}"`

3.3 Data Model

There was no common OPC UA data model for the material handling domain when the OPTIMUM project started. As a consequence, a data model has been developed to fulfill the needs of the OPTIMUM platform. As soon as there is an official Companion Specification for the material handling domain available, the data model can be modified to fulfill the requirements of such an official specification. However, the developed data model incorporates the OPC UA for Machinery profile, which can be seen as a digital nameplate for machines, to allow basic communication with other OPC UA devices outside of the OPTIMUM project. The defined object types of the data model are available as an XML-file. In the following, the OPTIMUM data model is described without the additional elements of the OPC UA for Machinery specification.

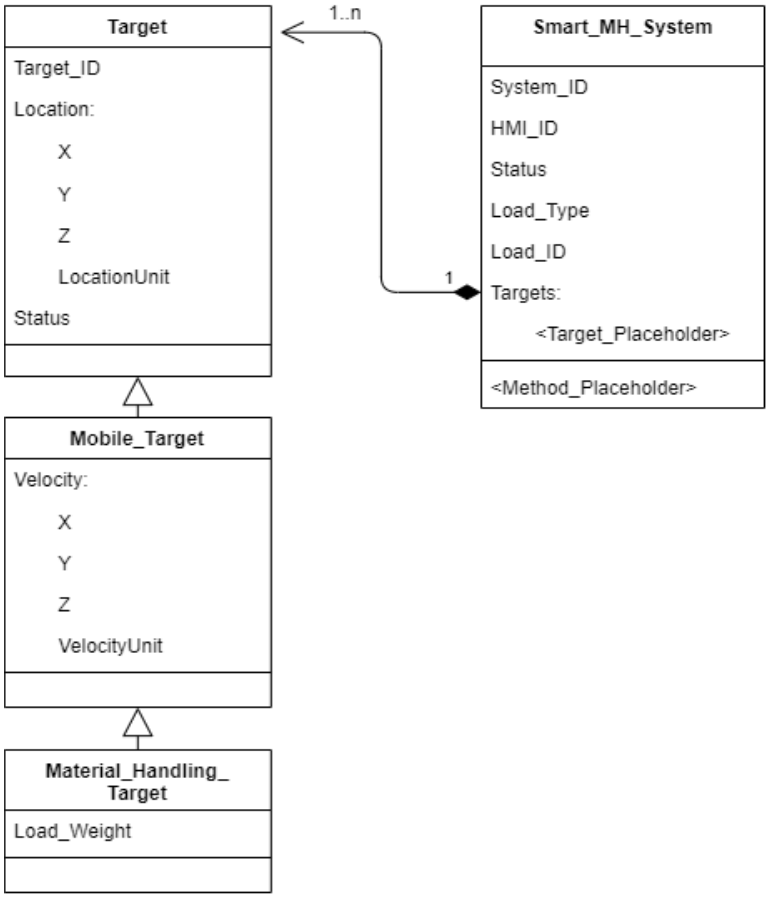

Figure 4: OPTIMUM Data Model - Object Relations

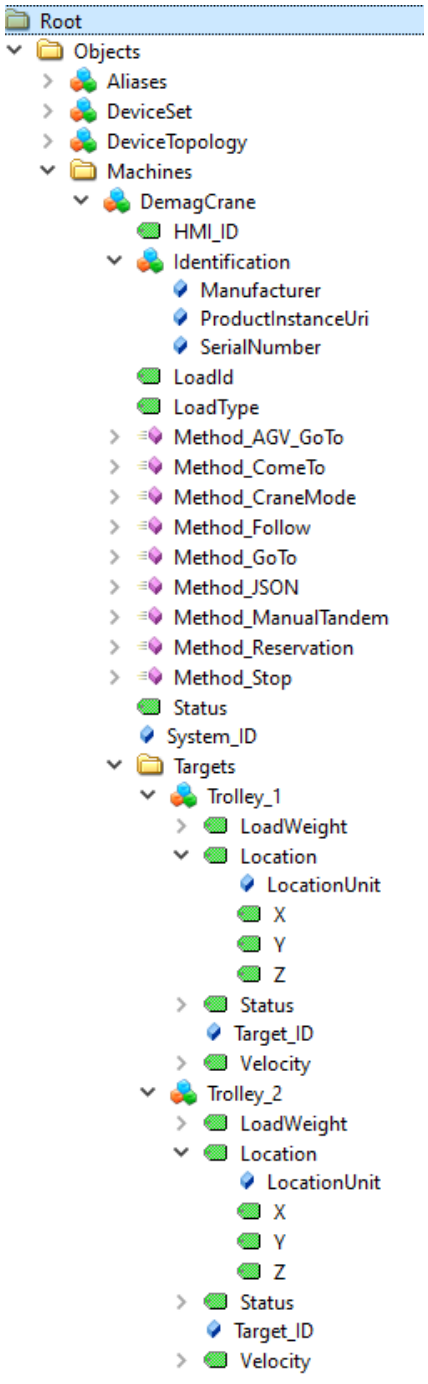

Figure 3: Data Model Example Crane in UAExpert

Table 3: OPC UA Object Type TargetType

OPC UA Object Type	**TargetType**
Namespace	http://optimum-itea3.eu/MaterialHandling/
Description	The Target Type is the basic element of the data model and defines a device that is aware of its own location in relation to a common coordinate system.
Object Children	▪ Location ○ X : Double ○ Y : Double ○ Z : Double ○ LocationUnit : PropertyType ▪ Status : String ▪ Target_ID : String
Children Description	**Location :** aggregates X, Y and Z coordinate of the device **LocationUnit:** reflects the metric unit used for X, Y and Z **Status:** Contains the current status of the device or component: *Available, Reserved, Maintenance, Error* **Target_ID:** contains the name of the device or component

Table 4: OPC UA Object Type MobileTargetType

OPC UA Object Type	**MobileTargetType**
Namespace	http://optimum-itea3.eu/MaterialHandling/
Description	The MobileTargetType inherits the object structure from the TargetType object and extends it with information about the device's velocity.
Object Children	▪ <TargetType Object Children> ▪ Velocity ○ X : Double ○ Y : Double ○ Z : Double ○ VelocityUnit : PropertyType
Children Description	**Velocity :** aggragates X_Speed, Y_Speed and Z_Speed velocity parameters of the device **VelocityUnit:** reflects the metric unit used for X, Y and Z

Table 5: OPC UA Object Type MaterialHandlingTarget

OPC UA Object Type	**MaterialHandlingTargetType**
Namespace	http://optimum-itea3.eu/MaterialHandling/
Description	The MaterialHandlingTargetType inherits the object structure from the MobileTargetType object and extends it with information about the device's current load weight (e.g. for cranes).
Object Children	▪ <MobileTargetType Object Children> ▪ LoadWeight : Double ○ WeightUnit : PropertyType
Children Description	**Load_Weight :** contains the weight of the current load transported by the device **WeightUnit:** reflects the metric unit used for LoadWeight value

Table 6: OPC UA Object Type Smart_MH_SystemType

OPC UA Object Type	**Smart_MH_SystemType**
Namespace	http://optimum-itea3.eu/MaterialHandling/
Description	The Smart_MH_SystemType aggregates multiple targets, contains additional information objects and methods that allow for a smart control functionality.
Object Children	▪ System_ID : String ▪ HMI_ID : String ▪ Status : String ▪ Load_Type : String ▪ Load_ID : String ▪ Targets : Folder ○ <Target_Placeholder> : TargetType ▪ <Method_Placeholder> : OPC UA method
Children Description	**System_ID:** contains the name of the device, can be set via command line argument (-s) during start of IIoT-Platform **HMI_ID:** contains the ID of the current user/HMI that reserved the device to use smart functions **Status:** Contains the current status of the device: *Available, Reserved, Maintenance, Error* **Load_Type:** contains the type of current load attached to the device **Load_ID:** contains the unique identifier of the transported load **<Target_Placeholder>:** placeholder for one or multiple objects of the type TargetType or derived types (e.g. MobileTargetType) **<Method_Placeholder>:** placeholder for one or multiple OPC UA methods that enable smart control of the device (see 3.3.2.4), methods are currently added via IIoT-Platform source code.

Access Variables in Data Model

If the same variable of one or several devices needs to be read repeatedly by an OPC UA client, the easiest way of retrieving this data via OPC UA is to address a specific node with its NodeID and NamespaceID. Alternatively, the name of the node can be used together with the NamespaceID. However, the name could not be unique, as in case of X, Y and Z coordinates. For example, the following constant NodeIDs hold true for the demonstrators of DEMAG and NXP using the OPTIMUM data model:

Table 7: Common NodeIDs for OPTIMUM demonstrators

Node	NodeID	Namespace ID
trolley_1/ Fork Lift/AGV/Operator X	1042	3
trolley_1/ Fork Lift/AGV/Operator Y	1043	3
trolley_1/ Fork Lift/AGV/Operator Z	1056	3
Trolley_1 Target_ID	1040	3
trolley_2 X	1052	3
trolley_2 Y	1053	3
trolley_2 Z	1061	3
Trolley_2 Target_ID	1050	3
DemagCrane/Fork Lift/AGV System_ID	6001	3
DemagCrane/AGV HMI_ID	6005	3

3.3.1 OPC UA for Machinery

The Companion Specification OPC UA for Machinery is a rather new specification and targets several application. However, only part 1 of this specification is published during the period of the OPTIMUM project which targets a common identification and nameplate of a device. Furthermore, the specification provides a common entry point to find all machines on an OPC UA server. Future goals of the Companion Specification workgroup are identification and nameplate of device components and an overall machine state. These elements could replace variables in the OPTIMUM data model with the same purpose in the future. The following sections give examples on the usage of this data model for machine discovery and identification.

Find All Machines of a Server

The specification defines a folder *Machines* as part of `Root-->Objects` that contains a link to all machines represented on an OPC UA server as shown in Figure 5.

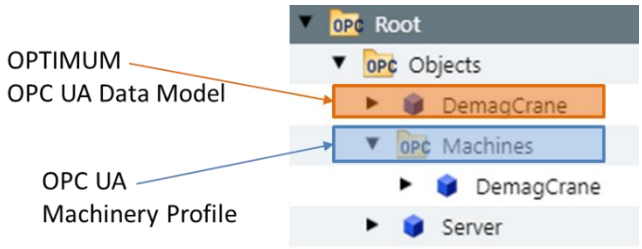

Figure 5: Machine Folder of OPC UA for Machinery CS

Identification

The digital nameplate used to identify a machine is provided by the specification as a data structure that is instanciated as a child of the machine object (see Figure 7). Figure 6 shows the elements of the Identification structure. If one or multiple elements of the identification structure exist already in the data model of the machine, they will be linked to the identification object. Further information on the details of the identification structure is provided in [1].

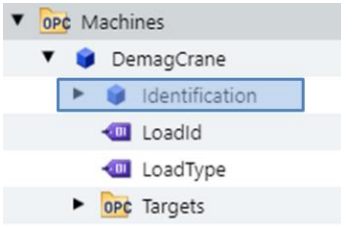

Figure 7: Identification object of OPC UA for Machinery CS

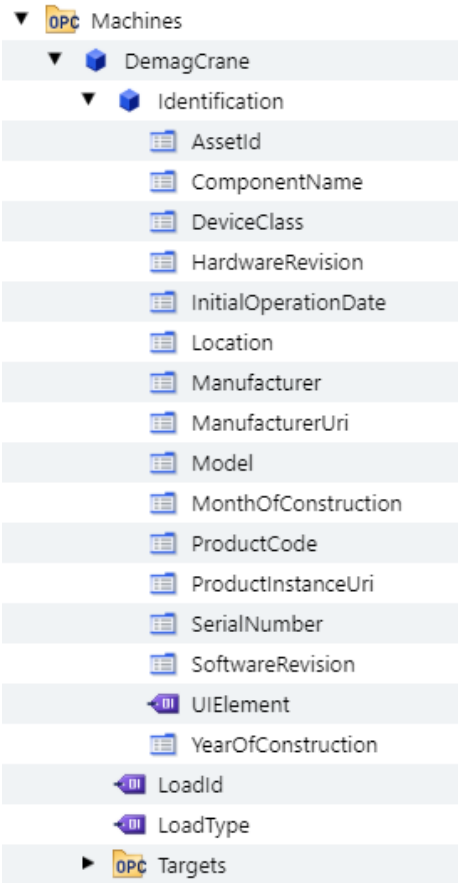

Figure 6: Identification data structure of OPC UA for Machinery CS

3.3.2 Semi-Autonomous Functions

The IIoT Platform provides OPC UA methods to activate semi-autonomous functions on OPTIMUM devices. An overview about the method calls is provided in the following. The methods are added as children of the OPTIMUM device object as shown in Figure 3. If not explicitly stated, all input arguments are mandatory. The Method output arguments always have the same structure.

3.3.2.1 Method Output Arguments

All methods rely on the same communication principle. If the syntax of the input arguments of a method call is valid, the IIoT of the target device responds with “request status: ack” and information about a *Response Node*. This information is called method output arguments. The device that called the method is required to create an OPC UA subscription (monitored Item) to the newly created *Response Node* of the targeted device to receive further status information about the request. See section 3.3.2.2 for more information.

In case of OPC UA method invocation over MQTT API, the IIoT platform takes care about this mechanism automatically and forwards all request status updates the requesting component.

Table 8: OPC UA Method Output Arguments

Output Arguments	• Request Status : String
	  ○ "ack" : method call input argument syntax valid
	  ○ "fail" : syntax of input arguments invalid
	• Information : String, human-readable information text
	• Response Node : String, NodeID of variable to be subscribed
	• Response Node Namespace : OPC UA Namespace of Response Node variable

3.3.2.2 Response Node

The value of the response node contains JSON encoded status information. The key-value pair „final_msg": „true/false" can be used to identify the final status and cancel the subscription.

Table 9: Response Node Mechanism

Value Syntax	**JSON encoded string with key-value pairs:**
	`{`
	`"status":"PLACEHOLDER",`
	`"final_msg":"PLACEHOLDER",`
	`"data":"PLACEHOLDER"`
	`}`
	
	`Example:`
	`{`
	`"status":"finish",`
	`"final_msg":"true",`
	`"data":"Target location reached"`
	`}`

Arguments	• status : String,
	  ○ "pending" : Gathering all required data
	  ○ "ongoing" : device realizes requested function, in general physical movement of the device
	  ○ "finish" : functionality successfully completed
	  ○ "success" : replaces "finish" in case of Method_Stop and Method_Reservation
	  ○ "fail" : execution of requested functionality failed
	• final_msg : String
	  ○ "false" : current message is only intermediate information, further status updates are expected
	  ○ "true" : final status update, no more messages expected, teardown of subscription on server side
	• data : String, component depended human-readable that contains specific information about the current status

3.3.2.3 FollowMachine Parameters

The FollowMachine functionality enables the user to control multiple devices simultaneously. The function is combined with semi-autonomous functions like GoTo or ComeTo. Therefore, it is not implemented as a separate method call but rather a set of additional input parameters of methods that support following machines. These additional input parameters are listed in the table below.

Table 10: FollowMachine Parameter Set

Function	**FollowMachine**
Purpose	Control multiple OPTIMUM cranes simultaneously
Description	Additional input parameters are used to configure this functionality when invoking a method. The function can currently address multiple cranes with multiple trolleys. Only specific configurations are possible. The FollowMachine arguments *Master Device* and *Master Trolley* are required to be always set (mandatory). Even if no simultaneous movement of devices is intended. FollowMachine is activated when devices are added to the arguments *Slave Devices* and *Slave Trolleys*. The location information of the master trolley is used to navigate the group of machines. Examples: • crane_1 moves its trolley_1 and trolley_2 simultaneous • crane_1 and crane_2 move together with only trolley_1 of each crane activated
Additional Input Arguments	• Master Device : String, System_ID of master device • Slave Devices : String array, optional, System_IDs of devices, e.g., “[“crane2”,”crane3”]” • Master Trolley : String, Target_ID of master trolley • Slave Trolleys : String array, optional, Target_IDs of trolleys, e.g., “[“trolley1”,”trolley2”]

3.3.2.4 OPC UA Methods

Some of the following OPC UA Methods are simulated in the testbed *OPTIMUM_HMI_Development*. This testbed is part of the IIoT-Platform project and is intended to be used during HMI testing. Further information is provided in section 0.

3.3.2.4.1 Method Reservation

Table 11: OPC UA Method Reservation

Comm. Protocol	OPC UA
Method Name	**Method_Reservation**
Description	Before any other control method (DCP) of an OPTIMUM device can be invoked, the device needs to be reserved by the operator/HMI. When the operator finishes the controlling of the crane, the reservation needs to be cancelled.
Information Flow	Request is forwarded to DCP. See 3.3.2.1 and 3.3.2.2

Input Arguments	• Client ID : String, User or HMI ID that reserved the device
	• Reservation status : String
	  ○ "on" = reserve device
	  ○ "off" = cancel reservation

Output Arguments	See 3.3.2.1
Simulated in Testbed	yes

3.3.2.4.2 Method Stop

Table 12: OPC UA Method Stop

Comm. Protocol	OPC UA
Method Name	**Method_Stop**
Description	OPTIMUM device shall stop its currently active function
Information Flow	Request is forwarded to DCP. See 3.3.2.1 and 3.3.2.2 If the method stops a previously started semi-autonomous function like Method_ComeTo, the cancelation of this semi-autonomous function is only indicated via the *Response Node* of the Method_Stop function.
Input Arguments	• Client ID : String, User or HMI ID that reserved the device • Request ID : String, name of *Response Node* of formerly invoked Method
Output Arguments	See 3.3.2.1
Simulated in Testbed	yes

3.3.2.4.3 Method GoTo

Table 13: OPC UA Method GoTo

Comm. Protocol	OPC UA
Method Name	**Method_GoTo**
Description	Move crane to a target position
Information Flow	Request is forwarded to DCP. See 3.3.2.1 and 3.3.2.2
Input Arguments	• Client ID : String, User or HMI ID that reserved the device • Coordinates : Comma separated string (x,y,z) • **<FollowMachine_Parameters>**
Output Arguments	See 3.3.2.1
Simulated in Testbed	yes

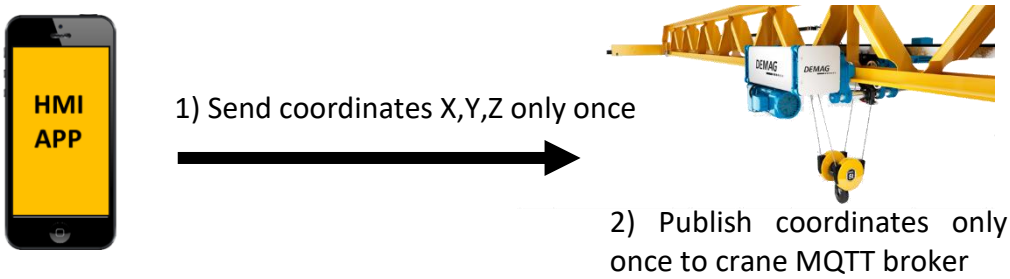

Figure 8: Visualization of Method GoTo

3.3.2.4.4 Method ComeTo

Table 14: OPC UA Method ComeTo

Comm. Protocol	OPC UA
Method Name	**Method_ComeTo**
Description	Move crane to position of an IIoT enabled target
Information Flow	Request is forwarded to DCP. See 3.3.2.1 and 3.3.2.2
Input Arguments	• Client ID : String, User or HMI ID that reserved the device • Target Device : URL (e.g. opc.tcp://operatorHW:4840) • **<FollowMachine_Parameters>**
Output Arguments	See 3.3.2.1
Simulated in Testbed	yes

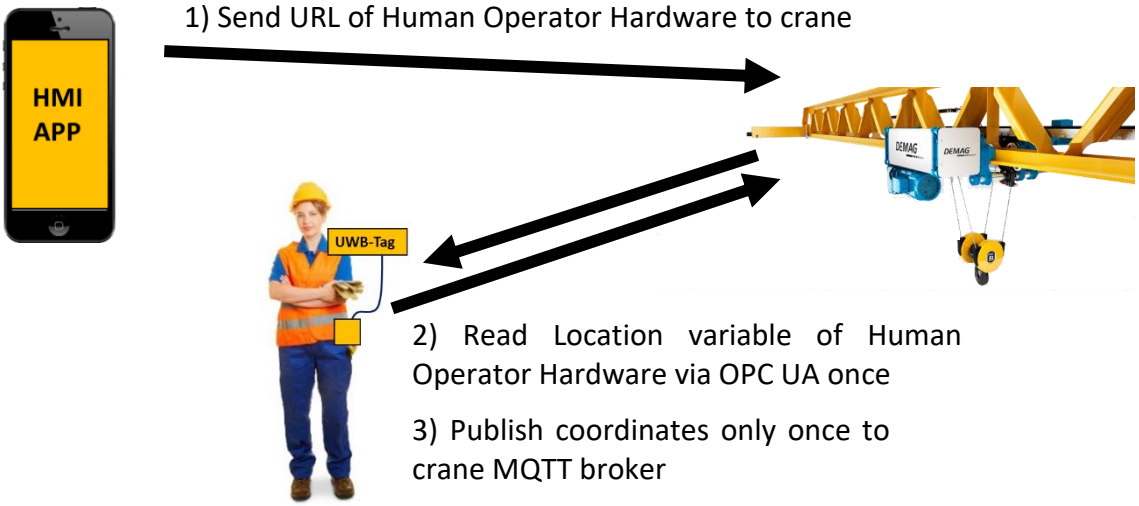

Figure 9: Visualization of Method_ComeTo

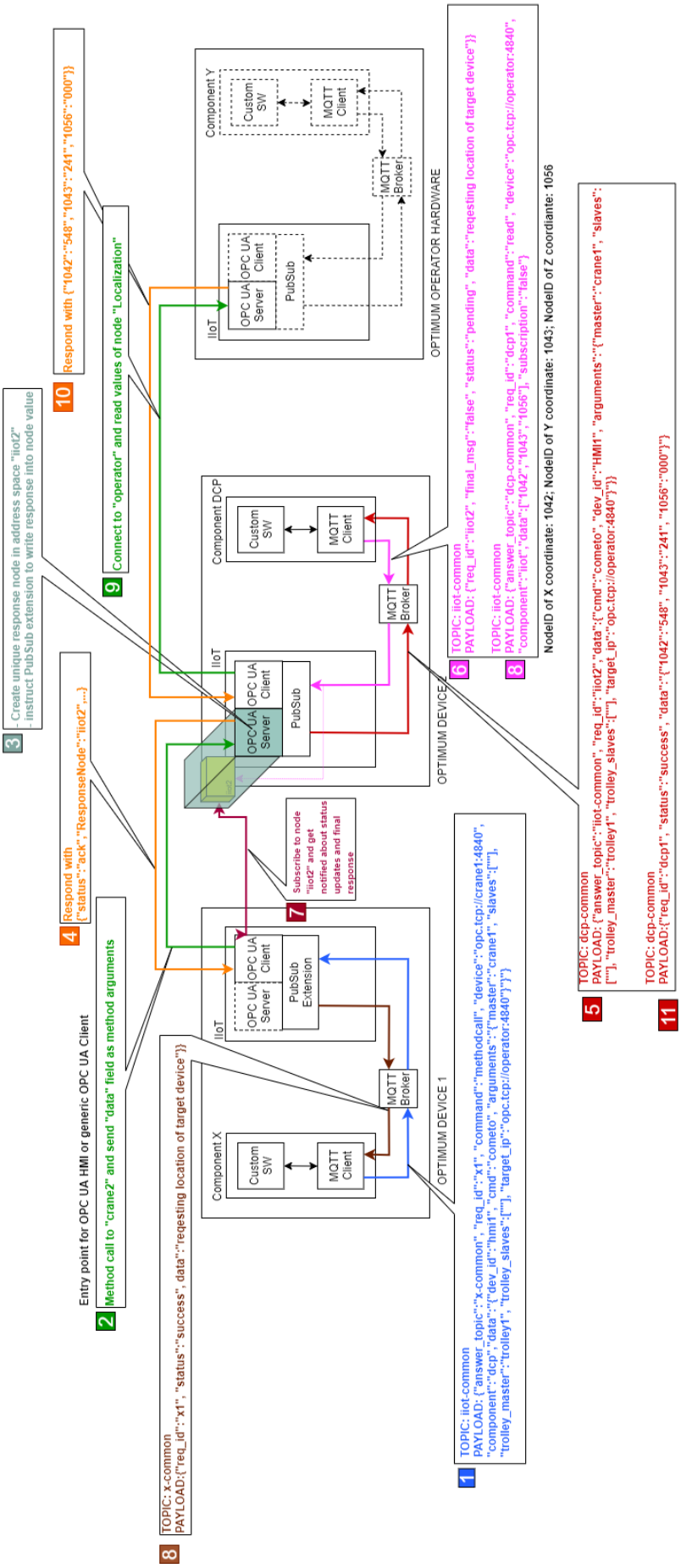

Figure 10: Exemplary message flow of Method_ComeTo, requested by component via MQTT, in case of OPC UA HMI/client step 2 is entry point

3.3.2.4.5 Method Follow

Table 15: Method Follow

Comm. Protocol	OPC UA
Method Name	**Method_Follow**
Description	Crane shall follow the a target
Information Flow	Request is forwarded to DCP. See 3.3.2.1 and 3.3.2.2 This function finishes only when **Method_Stop** is invoked
Input Arguments	• Client ID : String, User or HMI ID that reserved the device • Target Device : URL (e.g. opc.tcp://operatorHW:4840) • **<FollowMachine_Parameters>**
Output Arguments	See 3.3.2.1
Simulated in Testbed	yes

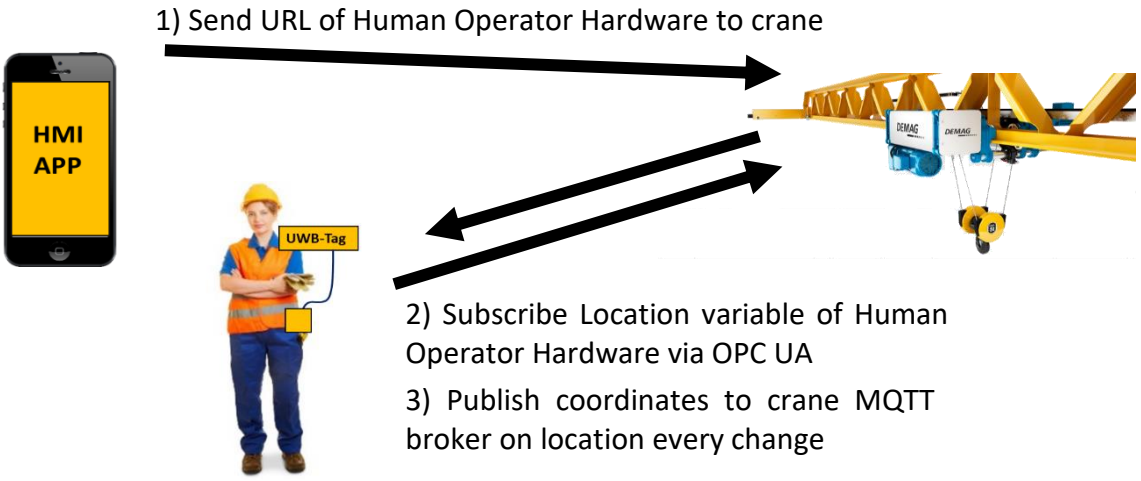

Figure 11: Visualization of Method_Follow

3.3.2.4.6 Method JSON

Table 16: OPC UA Method JSON

Comm. Protocol	OPC UA
Method Name	**Method_JSON**
Description	Used for general IIoT-to-IIoT communication and requires a specific JSON encoded syntax. This method implements a generic mechanism for communication to components of remote devices that are interfaced using an IIoT-Platform. It can replace all above mentioned simplified methods.
Information Flow	Request is forwarded to DCP. See 3.3.2.1 and 3.3.2.2

Input Argument	**JSON encoded string with key-value pairs, syntax:**
	{
	"component":"PLACEHOLDER",
	"data":{
	"CMD":"PLACEHOLDER",
	"Args":"PLACEHOLDER"
	}}
	
	Example:
	{
	"component":"dcp",
	"data":{
	"dev_id":"hmi1",
	"cmd":"gotoposition,
	"arguments":{
	"master":"crane1",
	"slaves":[""],
	"trolley_master":"trolley1",
	"trolley_slaves":[""],
	"position":{
	"x":1,
	"y":2,
	"z":3
	}}}}

Output Arguments	See 3.3.2.1
Simulated in Testbed	no

3.3.2.4.7 Method ManualTandem

Table 17: OPC UA Method ManualTandem

Comm. Protocol	OPC UA
Method Name	**Method_ManualTandem**
Description	Enable manual tandem functionality of master remote control to move two cranes simultaneously. Only one remote control of the involved cranes can be used at a time. The Tandem is a function provided by DEMAG cranes that allows the simultaneous movement of up to two cranes. Further information can be found in the DEMAG manuals.
Information Flow	Request is forwarded to DCP. See 3.3.2.1 and 3.3.2.2

Input Arguments	• Client ID : String, User or HMI ID that reserved the device
	• Master Device : String, System_ID of master device
	• Enable: String
	  ○ “on” = enable
	  ○ “off” = disable

Output Arguments	See 3.3.2.1
Simulated in Testbed	no

3.3.2.4.8 Method CraneMode

Table 18: OPC UA CraneMode

Comm. Protocol	OPC UA
Method Name	**Method_CraneMode**
Description	Swap between legacy control mode (no OPTIMUM functionality) and OPTIMUM mode with smart functionality
Information Flow	Request is forwarded to DCP. See 3.3.2.1 and 3.3.2.2

Input Arguments	• Client ID : String, User or HMI ID that reserved the device
	• Mode: String
	  ○ “legacy” = conventional device control mode
	  ○ “optimum” = OPTIMUM control mode (default)

Output Arguments	See 3.3.2.1
Simulated in Testbed	no

3.3.2.4.9 Method AGV GoTo

Table 19: OPC UA Method AGV GoTo

Comm. Protocol	OPC UA
Method Name	**Method_AGV_GoTo**
Description	Custom function to activate “go to position” functionality for automated guided vehicle (AGV)
Information Flow	Request is forwarded to DCP. See 3.3.2.1 and 3.3.2.2
Input Arguments	• Client ID : String, User or HMI ID that reserved the device • Start Node • End Node
Output Arguments	See 3.3.2.1
Simulated in Testbed	no

3.4 Additional Features under Development

3.4.1 Security

The IIoT-Platform contains in-build security features provided by the applied OPC UA library open62541 [2]. This includes encryption and integrity protection due to the use of an abstracted security layer that can integrate openSSL [3] or mbed TLS [4] security libraries.

Additionally, a certificate handling is implemented by the OPC UA library that allows to use basic authentication features.

A fine-grained certificate-based authentication and authorization concept has been developed in the OPTIMUM project.

The project partner NXP provides with the secure element SE050 a Trusted Platform Module (TPM) that stores certificates and keys. Furthermore, it provides hardware encryption capabilities to speed up communication via a secured channel.

The integration of the TPM functionality into the IIoT-Platform is currently under development and not yet part of the IIoT release version 0.2 that is described in this document.

3.4.2 OPC UA Discovery

The specification of OPC UA contains a discovery service to find devices in the network. A Local Discovery Server (LDS) that provides a list of all servers in a local network that registered themselves to the LDS. The Global Discovery Server (GDS) being a central database that holds connection information of OPC UA servers in the local or other networks of the same administration domain, e.g., company network that has additional capabilities for certificate issue and handling. The LDS fulfills the requirements in the OPTIMUM project and prototypical implementations are currently evaluated. This includes the modification of the LDS to provide additional device status information about the registered OPC Servers (IIoT-Platform). This modification has been developed for the unicast-based LDS as well as for the UDP multicast-based LDS-Multicast Extension (LDS-ME). The integration of a discovery service into the IIoT-Platform is targeted for the next release of the IIoT-Platform.

3.5 IIoT-Platform in Demonstrators

This section presents a short overview about the implementation status of the IIoT-Platform in the different demonstrators in the OPTIMUM project.

3.5.1 DEMAG

The IIoT-Platform is implemented as described in this document. Either running as native application on the embedded boards IMX6UL or IMX8 or as a Docker container.

3.5.2 NXP

The IIoT-Platform is implemented as described in this document. It is executed as a Docker Container on the development board IMX8. The Location information source is modified. Usually the data is modified via MQTT in the OPC UA data model. In case of NXP demonstrator, the location information is fetched from an Inter-Process-Communication message queue on every read or subscription interval.

3.5.3 IFAK

The demonstrator uses exclusively the DCP solution *DOME* from ifak [5] to control the cranes.

3.5.4 ETRI

Since the project ended earlier for this partner, a previous version of the IIoT-Platform and Data Model is used.

3.5.5 ERMETAL

Figure 12 depicts a simplified view of the ERMETAL (Turkish) demonstrator. As shown in the figure, it complies with the OPTIMUM architecture and therefore the IIoT component is an integral part of it. IIoT provides the M2M communication interface between the internal components of the industrial device and the other components of the system for non-time-critical data exchange.

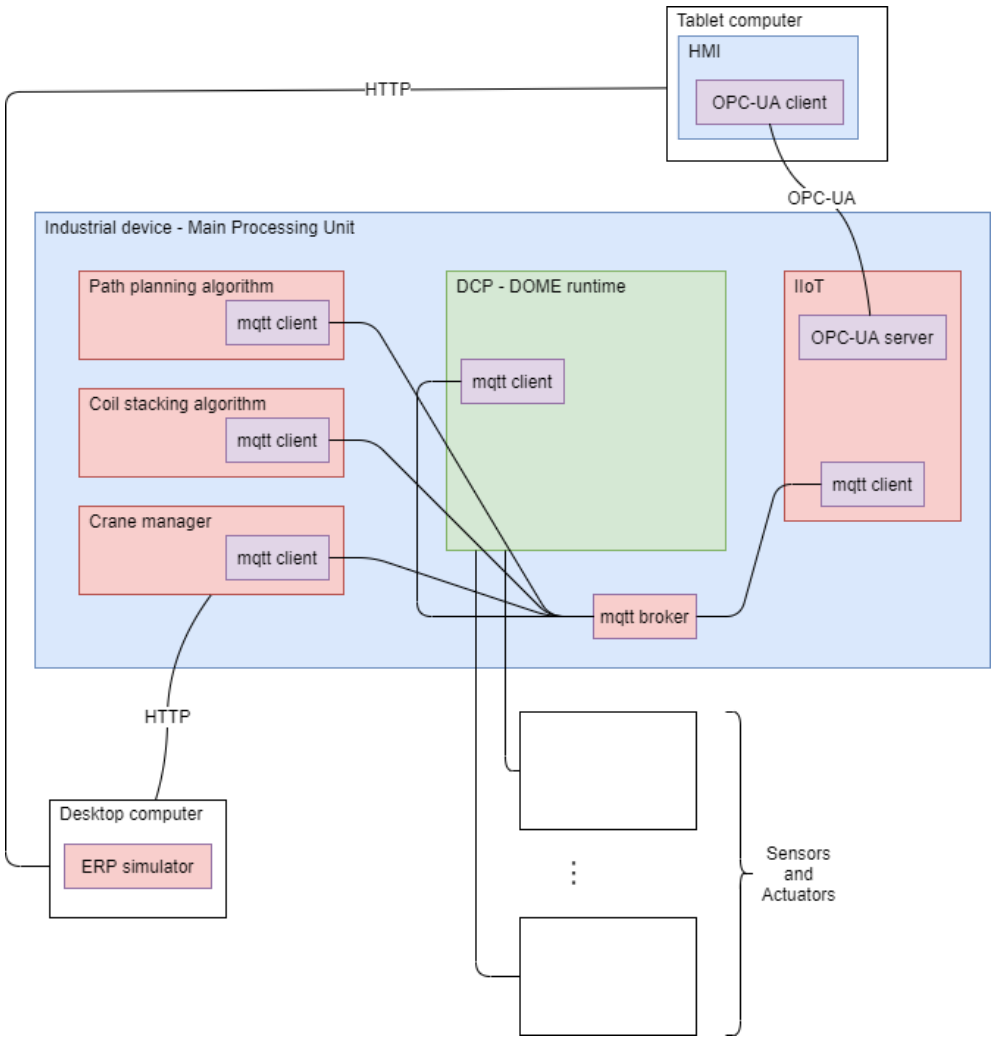

Figure 12: Final implementation of ERMETAL demonstrator

As of the publication date of this deliverable, a customised/specialised IIoT implementation (rather than common OPTIMUM IIoT-platform) is used for the realization of ERMETAL demonstrator. In the initial ERMETAL demonstrator (i.e., desktop demonstrator) it was not possible to use the common OPTIMUM IIoT-platform, since it was not available at the time (Fall 2019). In the final ERMETAL demonstrator (i.e., larger scale Turkish demonstrator), the common OPTIMUM IIoT-platform could not be integrated yet, due to time and resource limitations. It is expected to integrate the common OPTIMUM IIoT-platform into ERMETAL demonstrator with minor adaptation activities for 2 main reasons: 1. ERMETAL demonstrator is built using the OPTIMUM architecture, 2. The data model of the common OPTIMUM IIoT-

platform is developed as a generic data model taking all the demonstrator requirements into account. By migrating to the common OPTIMUM IIoT-platform, ERMETAL demonstrator can benefit from all the known advantages of using a common reusable software component.

3.5.6 MAGTEL

The Spanish demonstrator uses the DCP component and the IIoT-Platform to orchestrate the movement of collaborative robots (cobots) in a production line. The cobots expose an HTTP interface to exchange data and control commands with DCP and IIoT.

4 Implementation Details

The IIoT-Platform uses the open-source OPC UA implementation *open62541* [2] that is written in C/C++ for its core functionality. Some parts of the library are modified and extended to fulfill the requirements of the IIoT-Platform. This includes modification of the JSON data encoding and the integration of an MQTT subscriber for the Publish-Subscribe layer of open62541 that is standardized in OPC UA specification part 14 of the OPC Foundation. In future, the modifications are may aggregated into a patch that can be applied to all *open62541* release versions in theory.

Table 20 provides an overview of all external libraries on which the IIoT platform depends. These dependencies are linked via Git submodule mechanism.

Table 20: Overview of IIoT-Platform External Library Dependencies

Library	Description
open62541 [2]	The library provides OPC UA client, server and partial Publish-Subscribe functionality.
json-c [6]	This project provides common JSON data encoding capabilities besides the custom JSON encoding built into *open62541*.
uthash [7]	This project provides a lightweight implementation of a hash table that is used to maintain the data structures of request and their attached OPC UA client threads.
ua-nodeset [8]	This project reflects the official OPC UA standard Nodesets and is required by *open62541*.

A so called multi-stage Dockerfile can be used to build the IIoT-Platform with all its dependencies. This process creates a Docker image of the IIoT-Platform with minimal footprint. The IIoT-Platform project contains also yaml-file that defines a Continuous Integration flow to build the Docker Image for different platforms, e. g., amd64 and armV7.

Further information about the build process and start arguments can be found in the README.md in the project root directory.

Testbeds

Virtual test environments were created to verify the functionality of the IIoT platform and its interaction with the HMI smartphone application (OPC UA client) and the DCP. These testbeds are created with the Docker Compose application to create several instances of the IIoT-Platform, the MQTT broker and a mockup of the DCP in an isolated environment with a virtual network between them. The DCP mockup uses mqtt-spy [9] and a JavaScript script to respond to messages with requests from the IIoT-Platform.

5 References

[1] "OPC 40001-1 UA for Machinery Part 1 Basic Building Blocks," [Online]. Available: https://reference.opcfoundation.org/v104/Machinery/v100/docs/. [Accessed 04 05 2021].

[2] "open62541," [Online]. Available: https://open62541.org/. [Accessed 04 05 2021].

[3] "openSSL," [Online]. Available: https://www.openssl.org/. [Accessed 04 05 2021].

[4] "mbed TLS," [Online]. Available: https://tls.mbed.org/. [Accessed 04 05 2021].

[5] "Ifak DOME," ifak, [Online]. Available: https://www.ifak.eu/de/produkte/dome. [Accessed 04 05 2021].

[6] E. Haszlakiewicz, "GitHub json-c project," [Online]. Available: https://github.com/json-c/json-c. [Accessed 05 05 2021].

[7] T. D. Hanson, "GitHub uthash project," [Online]. Available: https://github.com/troydhanson/uthash. [Accessed 05 05 2021].

[8] OPC Foundation, "GItHub OPF Foudnation UA-Nodeset project," [Online]. Available: https://github.com/OPCFoundation/UA-Nodeset. [Accessed 05 05 2021].

[9] Eclipse Foundation, "Eclipse Paho MQTT Spy," [Online]. Available: https://www.eclipse.org/paho/index.php?page=components/mqtt-spy/index.php. [Accessed 05 05 2021].

6 Abbreviations

AGV ..... Automated Guided Vehicle
cobots ..... collaborative robots
DCP ..... Distributed Control Platform
GDS ..... Global Discovery Server
HMI ..... Human-Machine-Interface
IIoT ..... Industrial Internet of Things
LDS ..... Local Discovery Server
LDS-ME ..... Local Discovery Server Multicast Extension
M2M ..... Machine-To-Machine
TPM ..... Trusted Platform Module