Technical Report

Application and Extension of the Module Type Package Concept for Production-Related Logistics

Author:
Michelle Blumenstein, *Helmut Schmidt University Hamburg*

Reviewers:
Andreas Stutz, *Siemens AG*
Mathias Maurmaier, *Siemens AG*
Alexander Fay, *Helmut Schmidt University Hamburg*

Version 2, 06 November 2023

Version History of the Document

Version	Changes	Date
[v1]	Initial contents of Sections 1 to 4	2022-09-20
[v2]	First complete version of the document	2023-11-06

List of Contents

List of Abbreviations

AGV	Automated Guided Vehicles
AT	Attribute Type
ATL	Attribute Type Library
CES	Cyclic Execution Service
FFS	Form Fill Seal
HMI	Human Machine Interface
ID	Identifier
LA	Logistics Area
LEA	Logistics Equipment Assembly
LL	Logistics Line
LO	Logistics Object
LOL	Logistics Orchestration Layer
MLS	Modular Logistics System
MTP	Module Type Package
POL	Process Orchestration Layer
P&ID	Piping and Instrumentation Diagram
SES	Single Execution Service
SUC	System Unit Class
TCS	Transport Coordination System
WQC	Worst Quality Code

Typographical Notes

Typographic Formatting	Meaning	Example
Italic	MTP model terms	*StructuredServParam*

1 Purpose of this Document

Modular plants are gaining more and more importance in the process and manufacturing industries for creating flexible and adaptable production systems. In order not to restrict this flexibility, an equally modular and thus flexible and adaptable production-related logistics system is required. Here, the Module Type Package (MTP) concept, which is already known in the area of modular production, can be applied to the area of production-related logistics [1, 2].

In this context, this document describes interpretations and necessary extensions of the Module Type Package concept for the field of modular production-related logistics facilities. It serves as a technical specification to describe the basic concepts of Modular Logistics Systems and to specify necessary new MTP interface and model definitions. Its structure is oriented along the structure of the MTP standard with its different specification aspects.

In Section 2 a short introduction into Modular Logistics Systems as application context of this document is given. Section 3 introduces a service-based automation concept for logistics modules following the VDI/VDE/NAMUR 2658-4 [3] specifications. A concept for a further development of the vendor-neutral Human Machine Interface description of VDI/VDE/NAMUR 2658-2 [4] are described in Section 4. Based on VDI/VDE/NAMUR 2658-3 [5] and VDI/VDE/NAMUR 2658-4 [3] Section 5 proposes blueprints to reduce the complexity of base interfaces in the context of machines in discrete applications like logistics. In Sections 6 and 7 a cross-module coordination mechanism for the automation of Logistics Lines and an automation approach for Logistics Areas are introduced. Functions and architectural approaches for Logistics Orchestration Layers as overlaying control systems of Modular Logistics Systems are described in Section 8.

Based on all those concepts, Section 9 gives an overview of the necessary extensions to the Module Type Package specification in the context of production-related logistics. Finally, Sections 10, 11 and 12 define necessary semantical enriching model definitions and corresponding new interface definitions which can be used in logistics modules and their Module Type Packages.

Since the work on the concepts for modular logistic systems is still ongoing, there may still be changes to this document.

2 Modular Logistics Systems

Figure 2.1 shows an exemplary Modular Logistic System (MLS), which is used for filling and palletizing bags and octabins. This has been investigated and the findings have been published in [6].

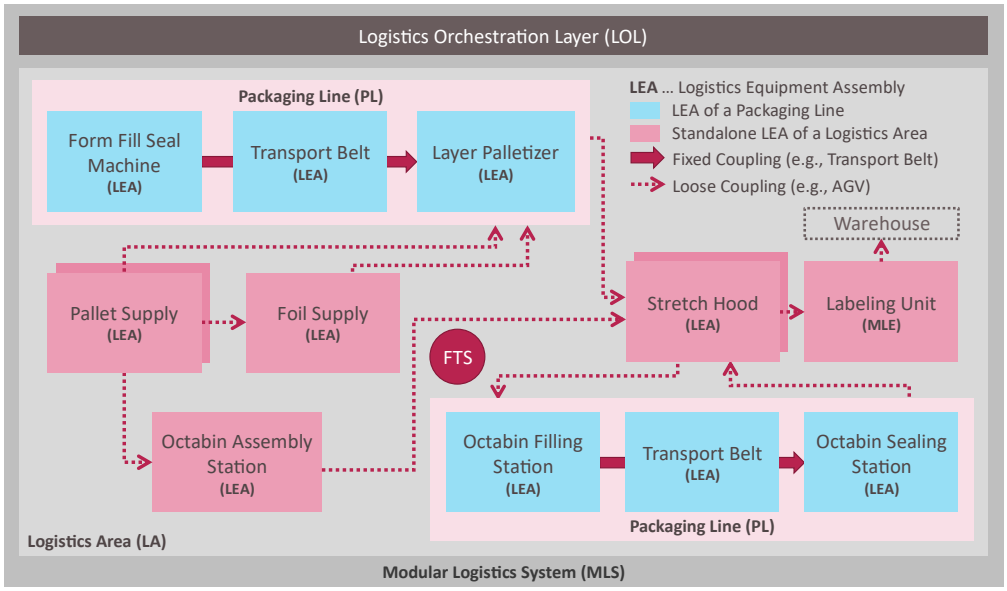

Figure 2.1: Structure of a Modular Logistics System

This system consists of Logistics Equipment Assemblies (LEAs) that can pack and process various Logistics Objects (LOs), such as bags and pallets. The LEAs can be integrated into fixed Logistics Lines (LLs) (see Figure 2.1, blue rectangles), in which the path an LO takes through the line is predefined. Necessary transports are executed e.g., by conveyor belts. In addition, LEAs and Logistics Lines can be arranged so-called Logistics Areas (LA) where they are loosely coupled with each other (see Figure 2.1, red rectangles). The path of an LO through a Logistics Area is only determined at runtime. Flexible transport systems, such as Automated Guided Vehicles (AGVs), are used in this case. A higher-level system, the Logistics Orchestration Layer (LOL), is provided for orchestrating the Modular Logistics System. The LOL takes over functions for order and parameter management, central control and monitoring or track & trace, although not all these functions are always necessary and/or available.

3 Automation Services for Logistics Equipment Assemblies

3.1 State-based Automation

Since LEAs usually implement only one specific logistical function, such as filling, transporting, or palletizing, it is reasonable to equip each LEA with only one service in the sense of the MTP concept. According to [6], this can be operated in two execution types – the order-oriented Cyclic Execution Service (CES) and the demand-oriented Single Execution Service (SES).

For some LEA types, it is useful to offer their logistics functionality as both CES and SES operations [6]. Since CES and SES cannot be executed simultaneously, they are implemented as different procedures of the MTP service of a LEA. CES and SES procedures conform to the existing MTP concept. However, they are based on special interpretations of the MTP state machine, which are described in more detail in Sections 3.1.1 and 3.1.2 and are published in [7].

For the unique identification of CES and SES procedures, corresponding *FunctionClassificationAttributes* according to VDI/VDE/NAMUR 2658-4 [3] are added to each procedure of a logistics service. In Table 3.1 and Table 3.2, *FunctionClassificationAttributes* for CES and SES procedures are proposed to be discussed in future MTP standardization work. "10" in this case stands for version 1.0 in major-minor format and can be incremented accordingly when changes are made to the *FunctionClassificationAttributes*.

Table 3.1: FunctionClassificationAttribute for a CES procedure

FunctionClassificationAttribute for CES	
Standard	ModuleTypePackage:Logistics
Level	Machine
Type	CES
IRDI	ModuleTypePackage-Logistics#CES#10 (following ISO/IEC 11179-6)

Table 3.2: FunctionClassificationAttribute for a SES procedure

FunctionClassificationAttribute for SES	
Standard	ModuleTypePackage:Logistics
Level	Machine
Type	SES
IRDI	ModuleTypePackage-Logistics#SES#10 (following ISO/IEC 11179-6)

3.1.1 Cyclic Execution Service

The Cyclic Execution Service (CES) is used to automate LEAs of a Logistics Line. It is designed to accept an order and then process all LOs belonging to this order identically. For example, in the case of the order "pack 500 bags on 10 pallets", the service of a Form Fill Seal machine (see Figure 2.1) would cyclically fill 500 bags in one service run. A characteristic feature of CES operation is that the service is parameterized once at the start of a service run according to the order data and then cyclically processes a specific or unspecified number of identical LOs. The necessary interpretation of the MTP state machine for a normal service run (without exception handling) is depicted in Figure 3.1.

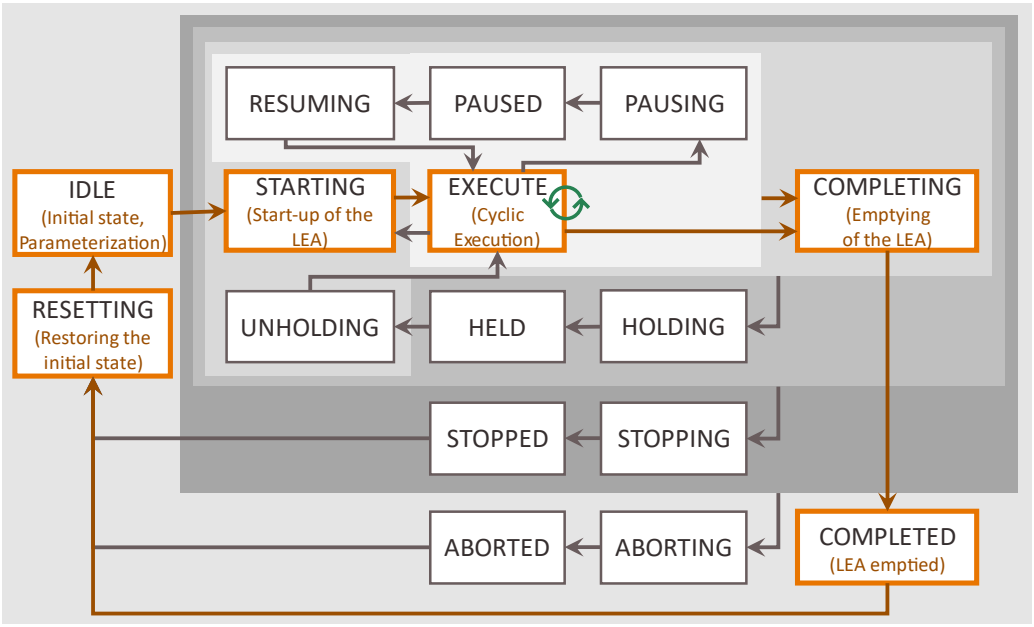

Figure 3.1: Operation of a Cyclic Execution Service

Like every MTP service, a logistics service in CES mode initially is in IDLE state. Since a CES procedure works in an order-oriented manner, all order data required for execution must be transferred to it before it can be started. For this purpose, a corresponding parameterization must be carried out according to Section 3.2. After starting the procedure in the STARTING state, LOs are processed cyclically with the same previously set order data in the EXECUTE state. CES procedures can be self-completing or continuous. Accordingly, the processing can be terminated by a Complete command or after a defined number of processed LOs. The LO currently being processed may be finished in COMPLETING state and afterwards the completion is signalled with COMPLETED state. Finally, a Reset command sets the procedure back to IDLE state.

The orange-marked states in Figure 3.1 represent the state of the LEA and not the state of the LO processing, in contrast to MTP applications in the process industry. The unmarked states, in particular the pause, hold, stop and abort loops, follow the semantics described in VDI/VDE/NAMUR 2658-4 [3].

3.1.2 Single Execution Service

The Single Execution Service (SES) is used to automate stand-alone LEAs in a Logistics Area. It is designed to process individual LOs on demand according to their individual order data. For example, the stretch hood machine shown in Figure 2.1 must be able to stretch both pallets with bags and pallets with an octabin with different parameters. For this purpose, a SES is parameterized individually for each LO. In this way, LOs from different orders can be processed according to their different order data. The number and sequence of LOs that are processed within a service run is undefined when the service is started and is determined on demand at runtime. The necessary operation of an SES is illustrated in Figure 3.2.

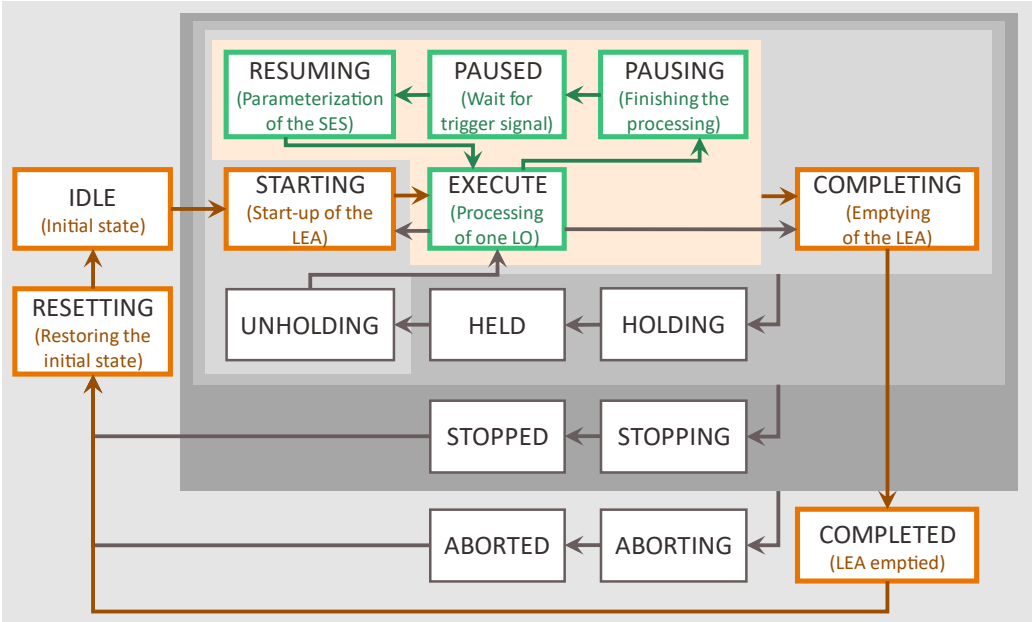

Figure 3.2: Operation of a Single Execution Service

At the beginning, a logistics service in SES mode is in the initial state IDLE. In this state, all parameters that are independent of the type of LO to be processed can be passed to the service using the parameterization mechanisms described in Section 3.2. Subsequently, the SES procedure is started independently of any order and changes through the STARTING, EXECUTE and PAUSING states to PAUSED state. Now the SES waits for an external trigger that indicates the demand to process a LO. Such a trigger could be, for example, an incoming AGV that intends to pick up a LO or to transfer it to the LEA. If such a trigger occurs, the service state changes to RESUMING and the SES is parameterized for the individual processing of the respective LO. In the following EXECUTE state, the processing of the LO is executed as required. After processing has been completed, the SES switches back to PAUSED state via PAUSING state and waits for the next trigger. If no further LOs need to be processed, the SES can be terminated by means of a Complete command. If necessary, the LO currently being processed is completed in the COMPLETING state. SES procedures are always executed continuously, since at the beginning of a service run it is not known how many LOs must be processed in which order.

The states marked in orange in Figure 3.2 reflect the state of the LEA, like in the CES case. The states marked in green, on the other hand, reflect the current processing state of an LO. The unmarked states, in particular the hold, stop and abort loops, follow the semantics described in VDI/VDE/NAMUR 2658-4 [3].

3.2 Parameters

This section specifies all relevant topics regarding parameterization of logistic services. Therefore, logistic-specific parameter types and the applicable parameterization mechanism are introduced as well as defined parameter requests in the context of production-related logistics. Parts of this concept are also published in [7].

3.2.1 Parameter Types

To adapt logistics functions to order-, product- and machine-specific conditions, appropriate parameterization is required. Accordingly, three types of parameters can be differentiated.

Order-specific parameters are used to transfer order data to a service. They result from customer orders and therefore change with each order. Essentially, they specify the organizational data of an order (e.g., the order number), the product to be packed and its quantity (e.g., the number of bags or pallets).

Product-specific parameters result from the LO to be packaged including its customer- and country-specific characteristics (hereinafter referred to as “product”). These parameters must be adapted if a different product needs to be packed. Examples are stretch parameters or packing patterns.

Construction-specific parameters are dependent on the physical structure of the LEA. They change when the LEA is physically modified or equipped. Essentially, they specify which functional equipment assemblies (FEAs) are assigned to the LEA (e.g., which filling nozzle is connected) or which supplies (e.g., pallet type) the LEA is equipped with.

3.2.2 Parameter Transfer

Parameters in modular plants can be transferred to the modules (here: LEAs) by different parameterization mechanisms. In particular, the following mechanisms can be distinguished.

Mechanisms 1 - Transfer of Individual Variables

This mechanism is based on transferring all parameters to the service via separate parameter interfaces.

- **Advantages:** Metainformation (e.g., minimum/maximum value or unit) can be provided for each parameter.
- **Disadvantages:** A large number of parameters may be required for parameterizing a LEA, making the service interface extensive and the parameterization time-consuming. In addition, each parameter is transferred individually to the LEA service. Thus, it must always be ensured that a consistent, valid data set is available at the service across all parameters.
- **MTP implementation:** This mechanism corresponds to the parameterization envisaged in the previous MTP concepts. Thus, corresponding parameter interfaces are already available in VDI/VDE/NAMUR 2658-4 [3].

Mechanisms 2 - Transfer of Parameter Sets

This mechanism envisages that parameters are not transferred to the service as individual variables but as a parameter set with an LEA-specific structured data type.

- **Advantages:** Especially for LEAs with large parameter sets, the service interface is simplified, and the effort required for parameterization is reduced. In addition, consistent writing and applying of the complete parameter set is possible.
- **Disadvantages:** No meta information can be given to the individual parameters of the parameter set. This would require read and write access to individual variables in the parameter set, which is not possible in complex data types according to VDI/VDE/NAMUR 2658-1 [8]. In addition, the

entire parameter set must always be transferred for parameterization, which can lead to a high network load.

- **MTP implementation:** So far, no parameter interfaces for structured data types are provided in the MTP concept. However, VDI/VDE/NAMUR 2658-1 [8] describes the possibility of modelling complex data types. Based on this, a *StructServParam* interface is presented in Section 10.8.1.

Mechanisms 3 - Selection of Parameter Sets

This mechanism combines the use of structured data types with the possibility of parameterization via a single variable. Parameter sets for different products are stored in the LEA in the form of an array. These parameter sets can be downloaded into the LEA at any time. An ID can then be used to select which parameter set should be applied for the current packaging process. This principle has already been implemented in many logistics systems (proprietarily).

- **Advantages:** A quick and easy selection of the parameter set to be used is possible. In addition, the consistency of the parameter sets is always ensured.
- **Disadvantages:** Two interfaces are necessary for this – one for loading the parameter sets into the LEA and one for the ID-based selection of one parameter set. Currently, there is no way to model the relationship between these two interfaces.
- **MTP implementation:** For the MTP-based implementation of the interface for ID selection, the *DIntServParam* interface specified in VDI/VDE/NAMUR 2658-4 [3] can be used. For loading the parameter sets, a configuration parameter is required to access an array located in the LEA. A corresponding interface is currently not provided in the MTP concept and is therefore specified in Section 10.8.1 as *ArrayServParam*. The individual parameter sets of the array can have a basic MTP data type (BOOL, DINT, REAL, STRING) or a LEA-specific structured data type, as described in mechanism 2.

Based on the parameterization mechanisms described above as well as their advantages and disadvantages, the following parameterization concept is recommended in the context of production-related logistics.

For the transmission of **order-specific** and **construction-specific parameters**, the parameterization method currently intended in the MTP concept is suitable. It uses a parameterization by individual variables, which is initiated from the LOL or carried out by an operator directly on the local panel of the LEA. In the case of more complex order interfaces, the transfer of a parameter set in the form of a structured data type (via *StructServParam*) can also be useful in order to reduce the parameterization effort. Due to their order-specific character, order-specific parameters are always to be implemented as procedure parameters according to VDI/VDE/NAMUR 2658-4 [3]. Construction-specific parameters are set at the time of commissioning of a LEA and must therefore be implemented by means of configuration parameters according to VDI/VDE/NAMUR 2658-4 [3].

For the transmission of **product-specific parameters**, the mechanism "selection of parameter sets" is recommended, since it enables parameterization with minimal communication effort and is also used in a similar way in today's logistics systems. Parameter sets for several products are stored in a LEA-internal data management, e.g., an LEA-internal data block. This data management can be loaded before the start of an order by the LOL via the *ArrayServParam* interface (initiative from the LOL), or the LEA can request

the data sets from the LOL (initiative from the LEA). Thereupon, the LOL transfers the data sets to the LEA via the *ArrayServParam* interface. Which variant is the more meaningful, depends on the specific use case. Since the loading of the array takes place independently of a concrete packaging order, the array interfaces shall be implemented as configuration parameters in the sense of VDI/VDE/NAMUR 2658-4 [3].

Using a unique product ID, the LEA can select and apply a corresponding data set for an order. In the case of a CES, the product ID is entered by means of a procedure parameter of type DINT as defined in VDI/VDE/NAMUR 2658-4 [3]. In the case of an SES, the product ID is transmitted to the LEA as part of the transport order data and does not require a separate parameter interface at the service of a LEA (see also Section 7).

Where applicable, packaging-specific parameters that are part of the product-specific parameters can be stored in a separate array and selected by a Packaging ID accordingly.

To implement this mechanism, a LOL must know which *ArrayServParam* interfaces are used to manage the product and packaging data sets, and which *DIntServParam* interfaces are intended to enter product and packaging IDs. For this purpose, semantic information at service parameter level are required. According to VDI/VDE/NAMUR 2658-4 [3], such information is currently only available at service and procedure level in the form of *FunctionClassificationAttributes*. Table 3.3 to Table 3.6 suggest the necessary *FunctionClassificationAttributes* as a basis for discussions in MTP standardization work. "10" in this case stands for version 1.0 in major-minor format and can be incremented accordingly when changes are made to the *FunctionClassificationAttributes*.

Table 3.3: FunctionClassificationAttribute for Product ID

FunctionClassificationAttribute for Product ID	
Standard	ModuleTypePackage:Logistics
Level	Service Parameter
Type	Product ID Procedure Parameter
IRDI	ModuleTypePackage-Logistics#ProductId#10 (following ISO/IEC 11179-6)

Table 3.4: FunctionClassificationAttribute for a Product Data Set

FunctionClassificationAttribute for a Product Data Set	
Standard	ModuleTypePackage:Logistics
Level	Service Parameter
Type	Product Data Set Configuration Parameter
IRDI	ModuleTypePackage-Logistics#ProductDataSet#10 (following ISO/IEC 11179-6)

Table 3.5: FunctionClassificationAttribute for Packaging ID

FunctionClassificationAttribute for Packaging ID	
Standard	ModuleTypePackage:Logistics
Level	Service Parameter
Type	Packaging ID Procedure Parameter
IRDI	ModuleTypePackage-Logistics#PackagingId#10 (following ISO/IEC 11179-6)

Table 3.6: FunctionClassificationAttribute for a Packaging Data Set

FunctionClassificationAttribute for a Packaging Data Set	
Standard	ModuleTypePackage:Logistics
Level	Service Parameter
Type	Packaging Data Set Configuration Parameter
IRDI	ModuleTypePackage-Logistics#PackagingDataSet#10 (following ISO/IEC 11179-6)

3.2.3 Initiation of Parameterization

In addition to the mechanisms for transferring parameters, the parameterization of LEAs can be distinguished by whether it is initiated by the LOL or by the LEA.

Parameterization by the LOL is equivalent to the variant currently provided in the MTP environment. Here, the LOL knows when which parameters are to be transferred to the LEAs and initiates the parameterization accordingly.

Since in logistics systems there is sometimes no continuous control from the LOL, but the LEAs operate largely autonomously, a **request for parameters by the LEA** is also useful in some cases. Such a variant is not yet foreseen in the MTP specification, but can be implemented similar to the service interaction mechanism described in VDI/VDE/NAMUR 2658-4 [3].

It became apparent that in the context of parameterization context, three defined requests from the LEAs to a LOL are useful:

- Request of parameter sets (**ParameterRequest**): If a LEA shall process a LO with a ProductID that is not available in the LEA-internal database, the LEA can request the corresponding data set from the LOL by means of a *ParameterRequest*.

- Information about a parameter update (**ParameterUpdatedInfo**): The LEA-internal data management of product-specific parameter sets may be changed not only by the parameter management of the LOL, but also at the local panel of the LEA. In this case, the LEA informs the LOL via a *ParameterUpdatedInfo* interaction that parameter values have changed.

- Request for the next transport node to be approached (**TransportNodeRequest**): In the course of coordinating flexible transports in a Logistics Area (see also Section 7), it may be necessary to request the next transport node to be approached from the LOL. This request is referred to as a *TransportNodeRequest*.

ParameterRequest

To query a parameter set by a LEA, the mechanism shown in Figure 3.3 is used.

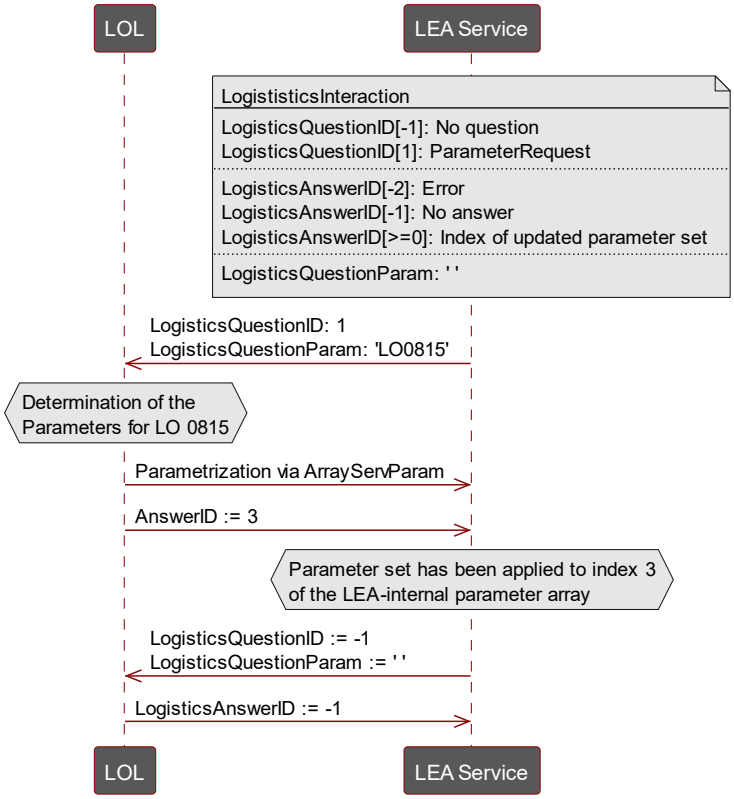

Figure 3.3: Sequence of the logistics interaction of a ParameterRequest

The LEA sends a *ParameterRequest* to the LOL by means of a corresponding *LogisticsQuestionID* (here: *LogisticsQuestionID* = 1) and transmits the ID of the LO currently being processed as *LogisticsQuestion-Param*. The LOL then determines the necessary parameters for the desired LO and parameterizes the LEA service accordingly via its *ArrayServParm* interface. If the parameterization is successful, the LOL returns a *LogisticsAnswerID* >= 0 (here: *LogisticsAnswerID* = 3) to the LEA, which reflects the index in the LEA-internal data management, to which the new parameter set has been written. A faulty parameterization is signalled with *LogisticsAnswerID* = -2. Subsequently, the *LogisticsQuestionID* and the *LogisticsAnswerID* are set to -1, which signals that there is currently no question or answer. In contrast to the existing concept of the *Questions*, the *ParameterRequest* does not have any answers defined in the MTP, but values of the number space >=0 that can be represented with DINT (maximum in the size of the LEA-internal array) as well as the values -1 and -2 can be returned as *LogisticsAnswerIDs*.

ParameterUpdatedInfo

To inform the parameter management of the LOL about the change of parameter values in the LEA, the mechanism shown in Figure 3.4 is used.

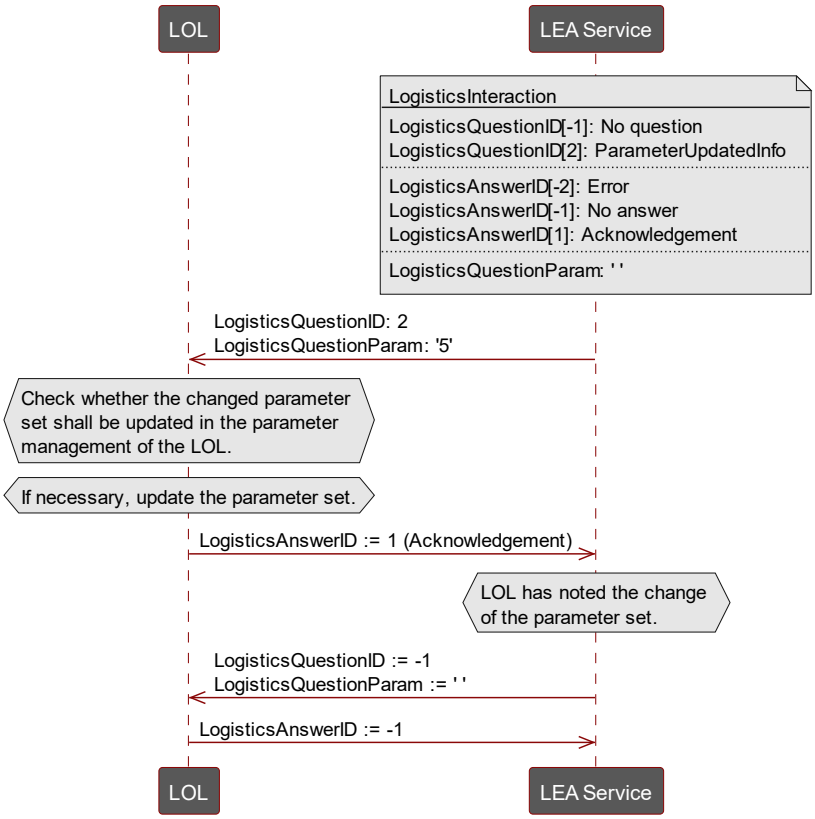

Figure 3.4: Sequence of the logistics interaction of a ParameterUpdatedInfo

The LEA sends a *ParameterUpdatedInfo* to the LOL by means of a corresponding *LogisticsQuestionID* (here: *LogisticsQuestionnID* = 2) and transmits the array index (here: array index = 5) of the changed parameter values as *LogisticsQuestionParam* to the LOL. The LOL's parameter management then determines whether the corresponding parameter data set should also be adjusted in the LOL (if necessary, by user query) and updates the parameter set as required. The LOL then acknowledges the parameter change by sending *LogisticsAnswerID* = 1 to the LEA. If an error has occurred, this is signalled by *LogisticsAnswerID* = -2. Subsequently, *LogisticsQuestionID* and *LogisticsAnswerID* are set to -1, which signals that there is currently no question or answer. In contrast to the existing concept of *Questions*, the *ParameterUpdatedInfo* does not have any answers defined in the MTP, but the defined values -2, -1 and 1 can be returned.

TransportNodeRequest

To query the next transport node to be approached in the Logistics Area, the sequence shown in Figure 3.5 is used.

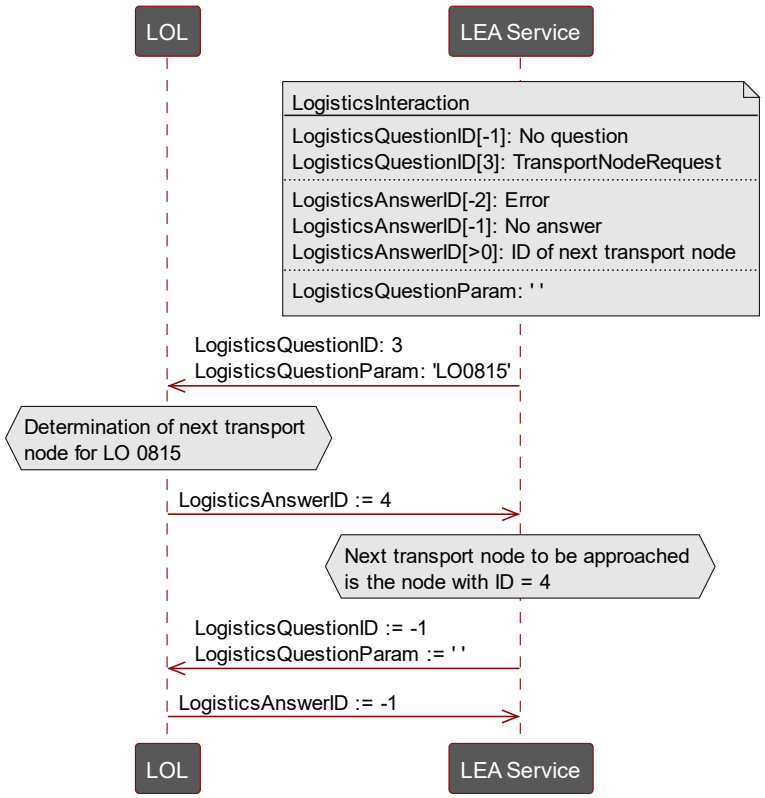

Figure 3.5: Sequence of the logistics interaction of a TransportNodeRequest

The LEA reports an information demand for the next transport node to be approached to the LOL by means of a corresponding *LogisticsQuestionID* (here: *LogisticsQuestionID* = 3) and transmits the ID of the LO currently being processed as *LogisticsQuestionParam*. In order not to require a separate parameter for setting the next node, the *LogisticsAnswerID* is directly interpreted as the ID of the next node to be approached. In Figure 3.5, the response of *LogisticsAnswerID* = 4 thus means that the transport node with ID = 4 should be approached next. If an error has occurred, this is signalled by *LogisticsAnswerID* = -2. Subsequently, the *LogisticsQuestionID* and the *LogisticsAnswerID* are set to -1, which signals that there is currently no question or answer. In contrast to the existing concept of *Questions*, the *TransportNodeRequest* has no answers defined in the MTP, but values of the number space >0 that can be represented with DINT (and correspond to a transport node ID in the logistics system) as well as the values -1 and -2 can be returned as *LogisticsAnswerIDs*.

For all three logistics-specific interactions – *ParameterRequest, ParameterUpdatedInfo* and *TransportNodeRequest* – the necessary *Questions* and possible *Answers* can be predefined. They do not differ for different LEAs. Thus, in contrast to the existing service interaction specified in VDI/VDE/NAMUR 2658-4 [3], the structure of the logistics interactions can be standardized and does not have to be modelled in the MTPs of the individual LEAs. This results in new model and interface definitions that enable those interactions. These are specified in detail in Sections 10.7 and 10.8.

3.3 Report Values

According to current knowledge, the already specified mechanism for report values is also suitable for the use in the area of production-related logistics. Similar to the parameter interfaces, the possibility of processing structures and arrays should also be foreseen in the context of report values. Accordingly, Sections 10.8.9 and 10.8.10 specify corresponding interface definitions.

3.4 Process Values

According to current knowledge, the already specified mechanism for process values is also suitable for the use in the area of production-related logistics. Similar to the parameter interfaces, the possibility of processing structures and arrays should also be foreseen in the context of process values. Accordingly, Sections 10.8.11 and 10.8.12 specify corresponding interface definitions.

4 Machine-oriented Human Machine Interfaces

While VDI/VDE/NAMUR 2658-2 [4] is designed for P&ID[1]-like Human Machine Interfaces (HMIs), for the field of logistics machine-oriented HMIs, as shown in Figure 4.1 for a pallet supplier, are more appropriate.

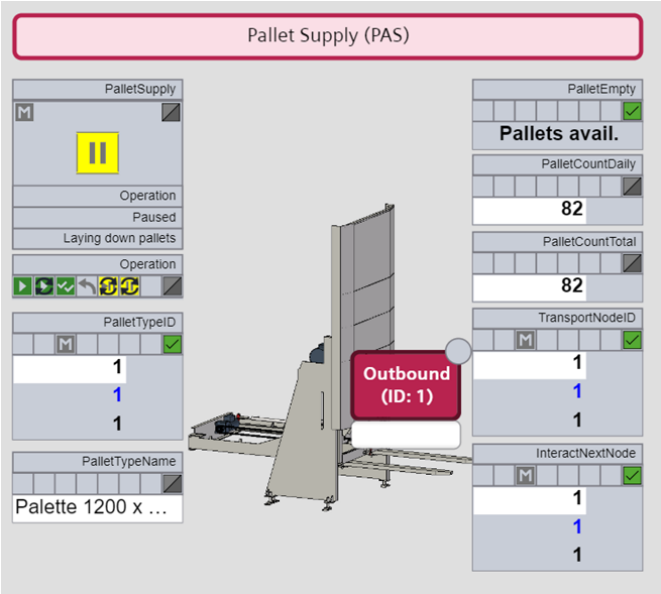

Figure 4.1: Human Machine Interface of a pallet supplier based on a custom ECLASS element

This HMI contains an image of the LEA as a static HMI object and several dynamic objects for parameters and report values. The latter can be implemented with the mechanisms from VDI/VDE/NAMUR 2658-2 [4].

Static objects are positioned as *VisualObjects* in the HMI according to VDI/VDE/NAMUR 2658-2 [4] and are provided with an ECLASS reference. The integrating system (here: LOL) must have this reference available in a graphics library to be able to display the static object accordingly. The used graphics should be as similar as possible to the appearance of the real machine in order to achieve a visual relationship between the operator screen and the real machine.

Especially with very special machines, the LOL may not have a suitable graphic in its library to represent the machine. To avoid having to keep images of many different machines in the LOL, it makes sense to include them in the MTPs of the LEAs as attachments according to VDI/VDE/NAMUR 2658-1 [8]. To identify the graphics in the appendix, the file names of those attachments should correspond to the ECLASS references, which are used for the HMI modelling. Here two cases are to be distinguished.

Case 1 – No suitable ECLASS exists

If no suitable ECLASS reference exists for a specific machine, numbers in the number range 90-90-XX-YY are to be selected, since these are not occupied with official coding. The graphics must be stored in the attachments folder in a separate HMI folder and the filenames of the graphic must correspond with the

[1] Piping and Instrumentation Diagram

selected ECLASS reference. If a visual object with an ECLASS reference starting with 90-90-* is then placed in the HMI image, the LOL knows that this object must be obtained from the MTP.

Case 2 – A suitable ECLASS exists

If a more or less suitable ECLASS reference exists for a specific machine, the module vendor nevertheless may want to provide a graphic for their specific machine. In this case the ECLASS reference best suited for the machine has to be used as file name for the graphics and for HMI modelling. If a LOL does not contain a graphic with the given ECLASS in its graphics library, it can obtain it from the HMI folder in the attachment of the MTP. If for a given ECLASS a graphic exists in the LOL graphics library as well as in the MTP attachment, the LOL must decide which one to use.

5 Systematic Complexity Reduction of Interfaces

The interfaces specified in VDI/VDE/NAMUR 2658-3 [5] and VDI/VDE/NAMUR 2658-4 [3] are defined for a wide range of use cases in the process industry – from laboratory to production. Although these interfaces are necessary for process engineering applications, they are too comprehensive for many use cases in discrete industries like logistics or manufacturing. Therefore, this section is intended to introduce some blueprints to reduce the complexity of MTP interfaces. This section might be extended in a future version of this document.

Since no adaptation of existing MTP interfaces shall be done, it is useful to reduce the complexity of the interfaces by setting reasonable default values. Figure 5.1 shows this principle using the example of the *ParameterElement* interface from VDI/VDE/NAMUR 2658-4 [3], which is part of every MTP parameter interface.

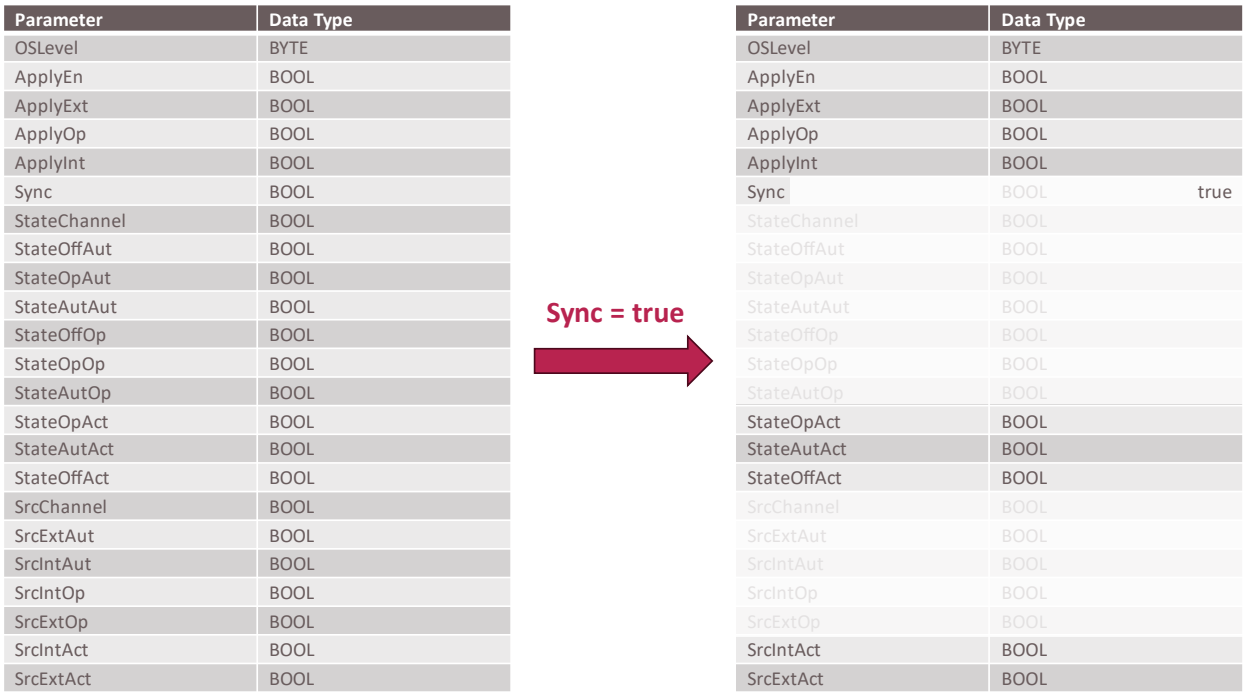

Figure 5.1: Complexity reduction of the parameter element interface of VDI/VDE/NAMUR 2658-4 [3]

While in process industry parameters can be operated in an operation mode that differs from the operation mode of the service, in the field of production-related logistics the assumption can be made that parameters always have the same operation mode as the superimposed service. In this case, the variable "Sync" can be set to "true" by default. As a result, many other variables of the interface become irrelevant, and the number of interface variables is reduced from 23 to 10 variables.

In this way, not only a reduction in the complexity of the interface but also a considerable saving of memory in the LEA controller can be achieved. All values greyed out in Figure 5.1 (right) no longer have to be provided in the OPC UA server of the controller but are set to constant values in the MTP. The saving of one Boolean value already corresponds to a saving of more than 100 bytes.

6 Coordination of Logistics Lines

Logistics Lines are built up from LEAs that are operated in CES mode. They operate on an order-oriented basis and thus according to a plan defined at the time of engineering. It is clearly defined which LEAs follow one another in which order, i.e., the material flow is not adaptable at runtime. When automating Logistics Lines, the challenge is to synchronize the different LEAs of the line. For example, a LEA must stop when its downstream LEA stops. Otherwise, there would be a congestion of LOs. In this context, the requirement exists that Logistics Lines should be coordinated by direct LEA-to-LEA communication. In this way, a higher-level entity controlling the LEAs is not permanently required, if at all, and faster response times can be achieved.

To meet this requirement, the concept of Automation Services Choreographies [9], which is currently being researched in the field of modular automation, has been taken up. This concept promises a systematic implementation of direct module-to-module communication without the need for a central coordination instance. The basic idea is that procedural, regulatory, interlocking, and parameterizing information is exchanged between different services of automated units. In addition, each service has an execution shell in which internal rules can be configured about how the corresponding service should react to certain information from other services. These rules are configured in the OPC UA server of the modules and can be adjusted without reloading the corresponding controller. In this way, it is possible, for example, to start a service as soon as another service is running or, in the event of an error in one service, to set all other services to an error state.

It has been shown that this approach is also well suited for automating modular Logistics Lines [10]. Figure 6.1 shows an example of a simple Logistics Line consisting of a Form Fill Seal (FFS) machine, a conveyor belt, and a layer palletizer.

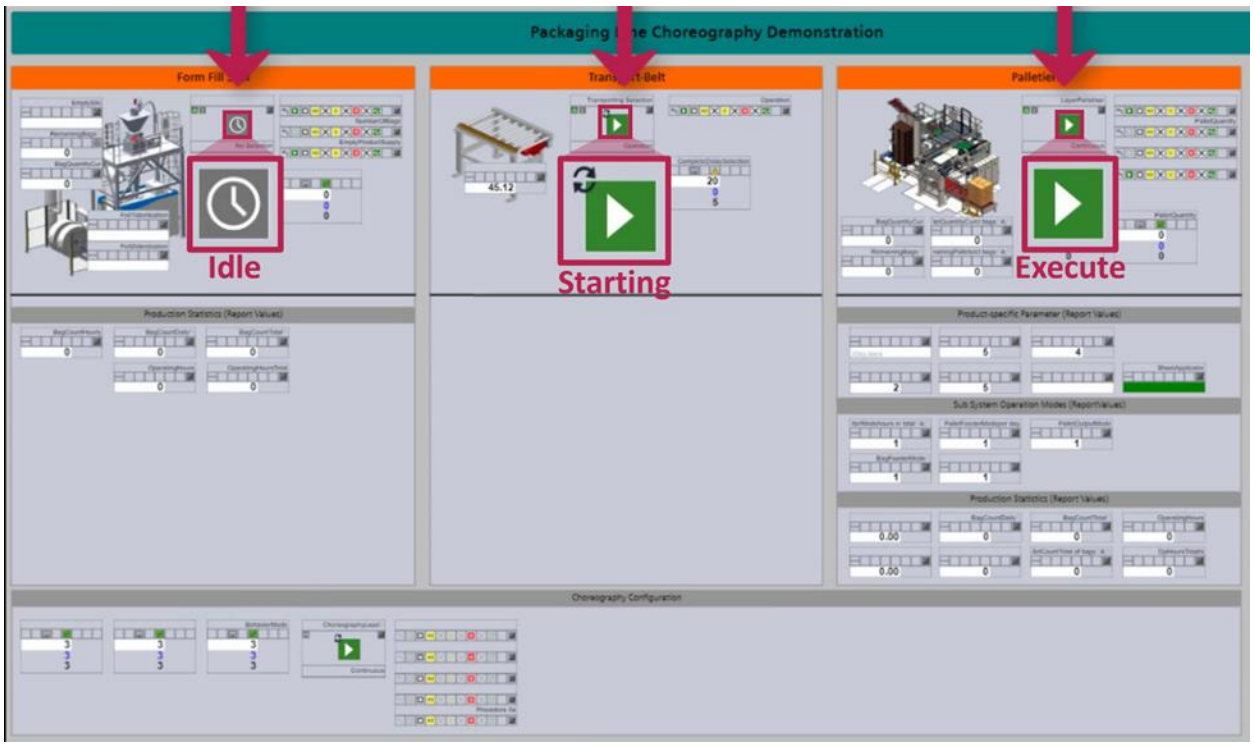

Figure 6.1: Operator screen of a packaging line controlled by an automation service choreography

Specifically, the above figure shows the start-up process of the line, which is carried out in an orderly manner from the last LEA (= layer palletizer) to the first LEA (= FFS machine). This orderly start-up is defined by the following two rules in the conveyor and the FFS machine:

- ConveyorBelt.Start := LayerPalletizer.EXECUTE [rule within the conveyor belt]
- FFS.Start := ConveyorBelt. EXECUTE [rule within the FFS machine]

If the layer palletizer is then started manually or by a higher-level system, its state changes from IDLE to STARTING to EXECUTE according to the MTP state machine. The conveyor belt monitors the state of the layer palletizer. As soon as the palletizer is in EXECUTE state, the internal rule of the conveyor belt takes effect, and the conveyor belt is started. This status of the line is shown in Figure 6.1. The FFS machine monitors the state of the conveyor belt. If the conveyor belt is then in the EXECUTE state, the internal rule of the FFS machine takes effect and also this machine is started. If the FFS machine is also in the EXECUTE state, the line is completely started up. From the point of view of an operator or a higher-level system only the Start command at the palletizer is necessary to trigger the start-up of the line. Everything else is done via direct LEA-to-LEA communication and LEA-internal rules.

In addition to the start-up of the line, the following other scenarios are considered reasonable in this example:

- After a Complete command, the line runs empty from front to back in an orderly manner.
- If an error occurs in one of the LEAs (in the form of Hold, Stop or Abort commands), this shall be propagated to the other LEAs and these also change to an error state.
- After a non-critical error state (HELD), the line can be restarted in an orderly manner from back to front.
- After the line has run empty or after a critical error state (STOPPED or ABORTED) has occurred, the line can be reset to the initial state IDLE.

A choreography design for these scenarios has been published in [10]. This design contains the necessary internal rules of the LEAs as well as the variables necessary at the interface of the line.

For the configuration of choreographies and the exchange of information between the LEAs of a line, new model and interface definitions are necessary. These are specified in Section 11.

7 Coordination of Logistics Areas

Logistics Lines (as a whole) and free-standing LEAs can be loosely coupled with each other in a Logistics Area. The free-standing LEAs are operated in SES mode and can be approached and used flexibly on demand. Thus, it is necessary to define the material flow through the system only at runtime. Necessary transports are performed by a flexible transport system, e.g., consisting of Automated Guided Vehicles (AGVs). The challenge in the automation of Logistics Areas is the integration and coordination of such a flexible transport system. There is no fixed predecessor-successor relationship between LEAs, the material flow is defined at runtime. The flexible transport system is to be configured according to the pending transport orders. A corresponding concept for the coordination of Logistics Areas has been published in [11].

7.1 Transport Processes in a Logistics Area

Figure 7.1 shows a typical structure of a Logistics Area and the transport processes therein. The transport of Logistics Objects (LOs) is carried out by an AGV, which is coordinated by a Transport Coordination System (TCS). The next LEA to be approached can be specified statically or can be selected dynamically at runtime by an external optimization system. Packaging orders are provided, e.g., by a manufacturing execution system (MES)[2].

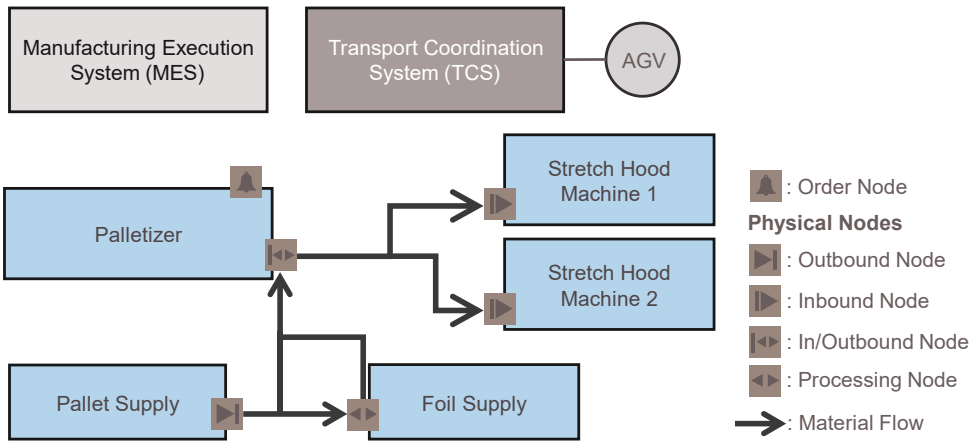

Figure 7.1: Transport processes within a typical Logistics Area

The LEAs of this system can interact with the AGV in different ways. Some LEAs have physical transfer points to transfer LOs to an AGV. For example, the pallet supplier can transfer empty pallets to the AGV. These transfer points are described as **outbound nodes** in the above figure. On the other hand, there are also **inbound nodes**, e.g., at the stretch hood machines, where LOs can be transferred from the AGV to the corresponding LEA. In addition, there are also **in/outbound nodes**, e.g., at the palletizer, which can perform the function of an inbound or an outbound node as required. Furthermore, it is possible that an LO is processed by a LEA while being on the AGV. For example, the foil supplier may apply a foil to an empty pallet that is on the AGV. For this case, **processing nodes** are foreseen.

[2] The Manufacturing Execution System should be understood as an example at this point. Other systems such as a material flow controller are also conceivable here.

Between those transport nodes, transport processes are performed based on **transport orders**. A transport process is the transport of an LO from an outbound to an inbound node, including the transfers at the start and target nodes. On the way from the start to the target, any number of processing nodes can be approached.

In order to plan the transport operations meaningfully, transport demands must be indicated in the system. For this purpose, so-called **order nodes** are available in the system, through which a LEA can report a transport demand. In the case of the shown system, only the palletizer has an order node, since only this LEA can report transport demands.

When reporting transport demands, a distinction must be made between **push and pull demands**. In the case of push transport demands, a LEA reports that it has completed an LO and wants it to be transported away. This occurs, for example, when the palletizer has finished palletizing a pallet. In a pull transport demand, a LEA has a material demand and wants to obtain a LO from another LEA. For example, the palletizer needs empty pallets from the pallet supplier on which it can palletize bags afterwards. A push transport demand is consequently initiated by the start LEA, a pull transport demand by the target LEA.

7.2 Transport Services for Encapsulating Transport Orders

According to the above definition of a transport order, the goal of transport coordination is to coordinate LO transports from an outbound to an inbound node and, while doing so, to approach any number of processing nodes. This coordination of transport processes takes place in the TCS. Each active transport order is represented as an MTP service in the TCS. This makes the TCS a provider of MTP services in the same way as the LEAs. However, a special characteristic is that any number of active transport orders can exist in the logistics system and thus the TCS can also contain any number of transport services. The transport services are made available via an OPC UA server of the TCS, each in connection with an individual transport ID (see Figure 7.2). This enables identification and dynamic connection of a transport node of an LEA with these services at runtime.

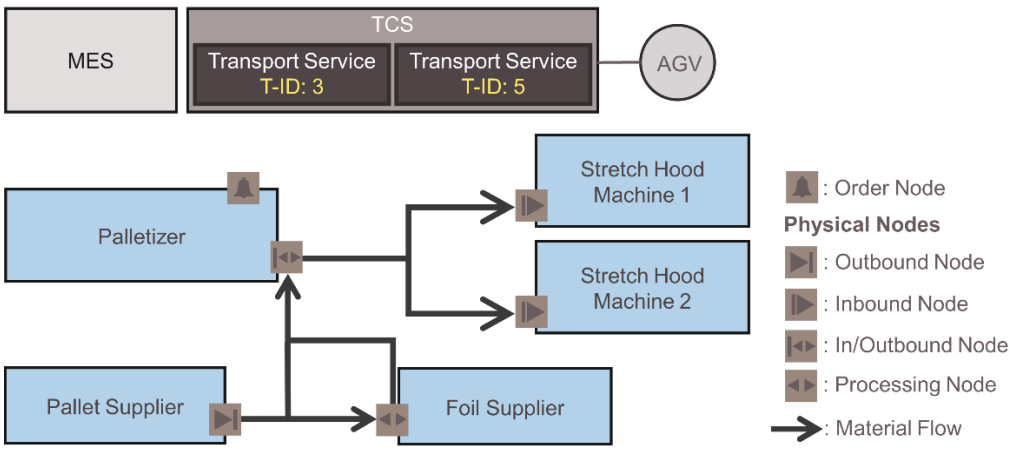

Figure 7.2: Representation of transport orders as MTP transport services in the Transport Coordination System

7.3 Procedures for Representing the Status of Transport Orders

A run of a transport service represents the sequence of a transport process from the initiation of the transport to the transfer of the LO from the transport system to the LEA at the target node. During this process, the transport service passes through various phases in which different coordination tasks must be performed. These are shown in Figure 7.3 in the form of a process model.

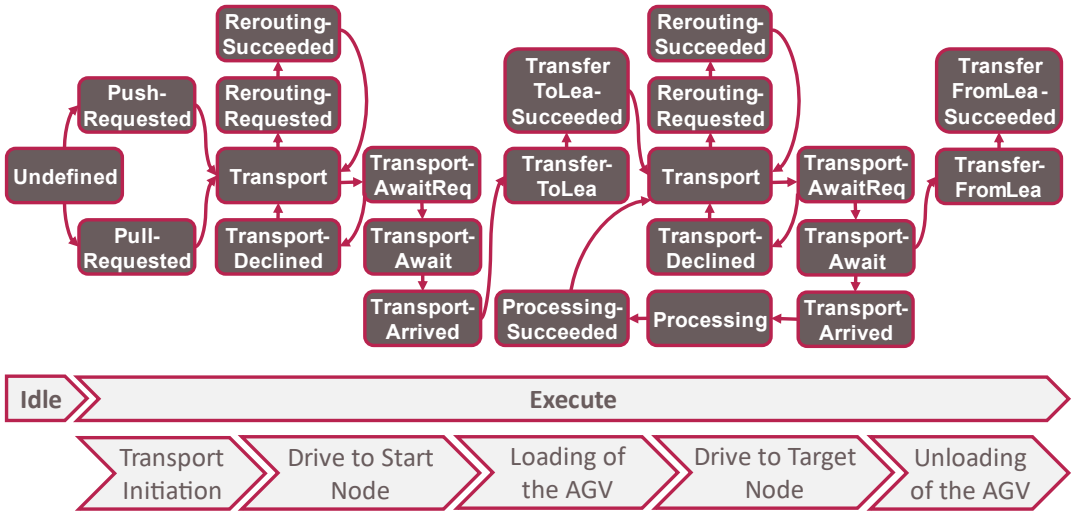

Figure 7.3: Process model of a MTP transport service

Independent of transport demands in the system, the TCS continuously ensures that transport services are created for all existing order nodes in the system and assigned to the corresponding order nodes. If a transport service is no longer assigned to an order node, a new blank transport order is automatically created in the TCS and assigned to this node.

If a LEA has a transport demand, it starts the transport service assigned to its order node as part of the **transport order initiation** and thus signals its transport demand to the TCS. At the same time, the information whether a push or pull transport is required and which is the start node of the transport is also transmitted.

The TCS then coordinates the **empty drive** of an AGV to the start node of the transport, where the **loading of the AGV** takes place. In the course of loading, an exchange of information is performed between the TCS and the LEA containing the start node. This synchronizes the transfer of the LO in the form of a handshake mechanism and sets the next node to be approached at the transport service. The TCS then coordinates the AGV to the next node.

If the next node is a **Processing** Node, the LO remains on the AGV and is processed by the LEA. If it is an inbound node, the **unloading** of the AGV is performed, whereby the transport order is completed after successful transfer of the LO to the target LEA.

These phases of a transport order are further subdivided into 14 different statuses that a transport order can have, as shown in Figure 7.3 (brown boxes). The states of the MTP state machine are not suitable to represent these statuses. Instead, the transport process is executed entirely in the EXECUTE state of the transport service, and the different transport statuses are mapped as different procedures of the

transport service. By restarting the transport service, it is possible to switch between the different procedures, i.e., between the different transport statuses according to the process model from Figure 7.3.

The resulting procedures of a transport service are shown in Table 7.1.

Table 7.1: Procedures of a MTP transport service

Procedure ID	Procedure Name	Procedure Description
16#1	PushRequested	A LEA indicates a transport demand due to a completed Logistics Object.
16#2	PullRequested	An LEA indicates a material demand.
16#3	Transport	An AGV has been scheduled and commissioned for a transport order. It travels to the intended next node.
16#4	TransportAwaitRequested	The AGV is close to the next node and requests permission to approach the node.
16#5	TransportAwaited	A transport service has been successfully coupled to the proxy interface of a LEA. The LEA confirms the transport order assigned to its node.
16#6	TransportDeclined	A transport service has been coupled to the proxy interface of a LEA. The LEA rejects the transport order assigned to its node.
16#7	TransportArrived	An AGV has arrived at a target node and is ready to interact with the LEA.
16#8	TransferFromLea	An LO is being transferred from a LEA to an AGV.
16#9	TransferFromLeaSucceeded	A LO has been successfully transferred from a LEA to an AGV.
16#A	Processing	A LEA is performing a transformation process on a LO that is on an AGV.
16#B	ProcessingSucceeded	A transformation process has been successfully performed on a LO.
16#C	TransferToLea	An LO is being transferred from an AGV to a LEA.
16#D	TransferToLeaSucceeded	A LO has been successfully transferred from an AGV to a LEA.
16#E	ReroutingRequested	A LEA intended for an active transport order is no longer operational. Rerouting is required.
6#F	ReroutingSucceeded	The MES has successfully assigned an alternative target node to a transport order as part of the rerouting process.

7.4 Proxy Interfaces for Flexible Interaction between Transport Services and LEAs

From the phases of a transport order, it becomes clear that a transport service must interact with various LEAs during its execution. Figure 7.4 shows this using the example of the palletizing LEA, which has completed a pallet that shall now be transported to a stretch hood machine.

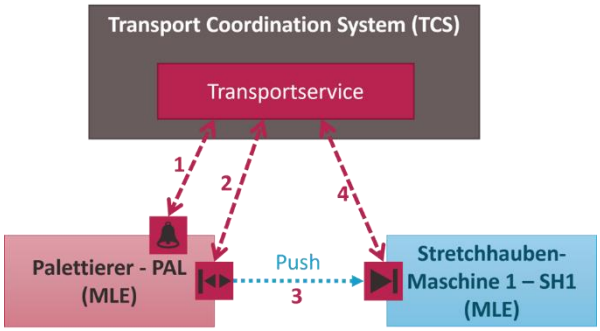

Figure 7.4: Interaction between a transport service and different LEAs in the course of a transport process

To initiate the transport, the transport service in the TCS interacts with the order node of the palletizer (1). During the transfer of the pallet to the AGV, the transport service interacts with the in/outbound node of the palletizer (2). During the transport, the transport service does not interact with any LEA (3). During the transfer of the pallet from the AGV to the stretch hood machine, the transport service interacts with the inbound node of the stretch hood machine (4).

To implement this flexible interaction between the transport service and the various nodes of the LEAs, the principle of decentralized orchestration has been applied, which was developed in the MTP environment [12]. For this purpose, proxy interfaces are provided in the LEAs, each of which is a software implementation of a node. They enable a transport service to be connected via an OPC UA connection configurable at runtime. Via the proxy interfaces, the interface of the transport service is mirrored into an LEA. Thus, although the transport service runs in the TCS, the information is equally accessible to the LEA and allows interaction with it. In principle, three different proxy interfaces are used in the developed concept (see Figure 7.5).

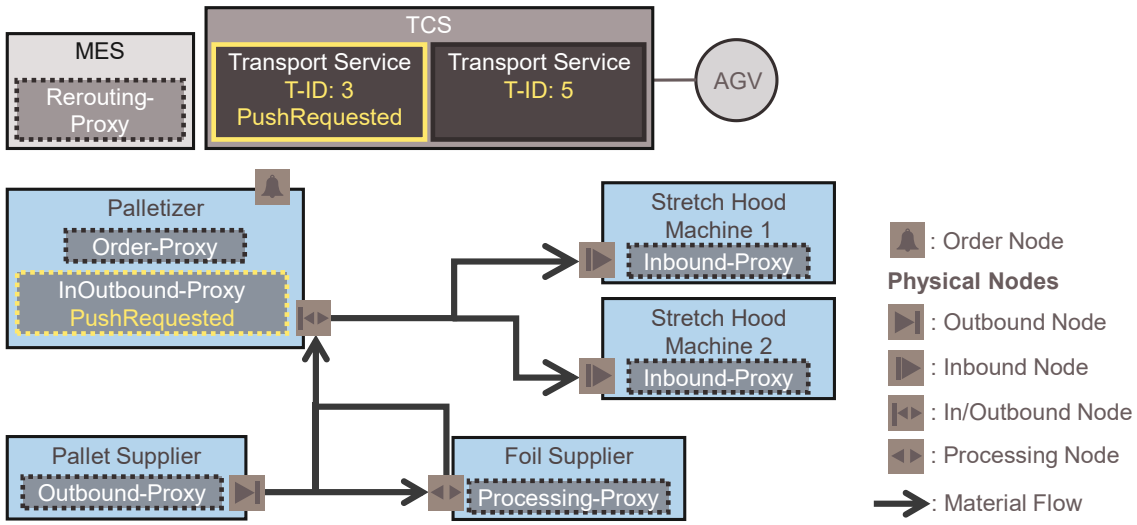

Figure 7.5: Proxy interfaces for flexible connection of MTP transport services

Transport proxy interfaces correspond to the introduced physical nodes for the transfers from and to LEAs and processing of LOs and can be further subdivided into inbound proxies, outbound proxies, in/outbound proxies and processing proxies.

Order proxy interfaces are intended for order nodes and thus for transport order initiation. A characteristic feature of these proxies is that they are automatically assigned a blank transport order by the TCS as soon as they are no longer connected to a transport order.

A **rerouting proxy** interface is provided by the MES. Based on this interface, it is possible to establish a temporary OPC UA connection between a transport service and the MES and thus allow an update of the transport order from the MES. This is necessary if a LEA that is currently being approached by an AGV is no longer operational. This is detected by the TCS, which then connects the transport service to the rerouting proxy interface.

8 Logistics Orchestration Layer

Modular Logistics Systems require overarching management, coordination and monitoring functions that are handled by a Logistics Orchestration Layer (LOL). This section describes a first set of LOL functions as well as hints for their implementation.

The LOL functions shown in Table 8.1 have been derived based on functions of today's logistics control systems (mainly based on VDI 5600 [13, 14]) and the functions of today's Process Orchestration Layers (POLs) and have also been published in [15].

Table 8.1: Functions of a Logistics Orchestration Layer

1) Order Management
The management of internally and externally initiated orders is an essential part of today's logistics control systems and is therefore also envisaged in LOLs in the same way. The MTP-based standardized LEA interfaces simplify the transmission of order information to the LEAs compared to today's logistics machines. This supports the idea of "digital customer order management" in the sense of VDI 5600-7 [14].
2) Asset Management
The asset management of today's logistics control systems is also adopted for LOLs, however, this is further developed in the direction of the asset management of POLs. In addition to the pure management of LEA types and instances, there is also simple integration and disintegration of LEAs by means of their MTPs and comprehensive monitoring, alarm management and diagnostics via standardized MTP interfaces. This supports an "adaptive machine and plant connection" in the sense of VDI 5600-7 [14].
3) Orchestration Configuration
In the context of Modular Logistics Systems, classical detailed planning and fine control, which is designed for fixed planning and non-modular logistics systems, no longer appears to be effective. Instead, the approach from POLs is used to create orchestration configurations, which can then be activated by configuration management. In the sense of VDI 5600-7 [14], this serves "dynamic detailed planning in production". Since LEAs often operate quite autonomously and largely independently of other LEAs, however, the focus of the orchestration configuration for LOLs is not on the creation of procedural sequences that are later executed by the LOL, or on cross-module regulations and interlocks. Instead, it is necessary to configure the dependencies of LEAs in Logistics Lines according to Section 6. For this purpose, a line configuration functionality is useful, which is described below and can be used by orchestration configuration. In addition, the correct parameterization of LEAs is essential. Due to the large number of parameters, this should be automated by a parameter management function described below. Parameter management supports the "assisted master data management" described in VDI 5600-7 [14].
4) Configuration Management
To manage, activate and deactivate the created orchestration configurations, configuration management is provided for LOLs similar to POLs. As with POLs, when a configuration is activated, specific LEA instances are assigned to the topology configuration placeholders. However, due to the loose

chaining of LEAs in Logistics Areas, they can be flexibly assigned to the placeholders of the process configuration at runtime [16]. In this case, the assignment of a specific LEA instance to a process step is performed by operation management. The LEA instances assigned in the topology configuration are available for this purpose.

5) Operation Management

The LOL's operation management provides similar functionality to that of POLs, supplemented by the previously described function of assigning LEA instances to process steps at runtime. Operation management implements the activated orchestration configuration and thus processes orders provided by order management. The LEAs operate largely autonomously and in a decentral manner once the order has been transferred and do not have to be controlled individually by the LOL, as in the case of POLs. In the course of executing the logistics functionality, data is collected and stored. This is similar to the data acquisition and information management in today's logistics control systems. However, standardized MTP interfaces can be used for data acquisition. On the basis of this data, further tasks can be performed in line with the functions of today's logistics control systems, such as track and trace, performance analysis or quality management. Depending on the scope of these tasks in the specific application, they can be regarded as separate LOL functionalities used by the operation management or implemented directly in the operation management. Standardized MTP interfaces simplify data transfer between different applications, even across manufacturer boundaries. In this way, for example, "cross-company traceability" in the sense of VDI 5600-7 [14] can be implemented.

6) Material Management

The material management functionality of a LOL essentially comprises the same tasks as the function of the same name in today's logistics control systems. One challenge here is the MTP-based integration and coordination of flexible transport processes (e.g., based on Automated Guided Vehicles). For this purpose, the LOL provides a transport management functionality, which is described below. This includes, among other things, demand-driven ordering of material by the LEAs in the sense of "dynamic material management and transports" according to VDI 5600-7 [14].

7) Staff Management

The staff management foreseen in today's logistics control systems is also relevant for LOLs in the same way. No changes to this function are expected in the context of Modular Logistics Systems.

8) Energy Management

This LOL function is based on the energy management of today's logistics control systems. In addition, solutions for energy management in modular plants are currently being developed in the MTP environment. For example, a uniform information model for managing energy data is being developed [17] and its implementation is being investigated in the context of the MTP concept [18]. It is expected that these solutions will essentially also be suitable for LOLs. A uniform MTP-based model, which contains both process-relevant and energy-relevant information, can be used to implement integrated energy management in the sense of VDI 5600-7 [14].

9) Systems Management
In the context of LOLs, similar to POLs, a platform concept is pursued. This requires certain overarching system functions, such as user and rights management, security functions or version management of the LOL functions. These features are summarized as systems management here.
10) Line Configuration
The choreography-based Logistics Line coordination mechanism described in Section 6 requires a configuration of internal rules in each LEA that describe how the LEA should respond to changes in process variables of other LEAs. In order to integrate the choreographed Logistics Line into the LOL like a single MTP service, it is furthermore necessary to combine the MTPs of the individual LEAs into a "Composed-MTP" that describes the interface of the Logistics Line as a whole. In this context, the line configuration functionality of a LOL enables the configuration of the choreography rules and the subsequent creation of a "Composed-MTP".
11) Transport Management
The concept for Logistics Areas described in Section 7 envisages an MTP-based abstraction layer for flexible transport systems, which is provided by the LOL's transport management. This allows proprietary transport systems to be connected, aggregated, and provided to LEAs via a uniform interface. LEAs can report transport demands directly (in a decentral manner) to a transport system, which then executes them. Alternatively, it is also possible to schedule the transports in the orchestration configuration. In this case, the transports are commissioned centrally from operation management. In this case, operation management must be able to handle a variable number of MTP transport services at runtime.
12) Parameter Management
LEAs, like existing logistics machines, have a large number of parameters to adapt them to the specific application context and different products to be processed. Due to the expected fast set-up and reconfiguration of Modular Logistics Systems, the adjustment effort of these parameters increases significantly. Therefore, a parameter management functionality is provided in the LOL to perform the LEA parameterization in an automated manner as far as possible. In this way, rapid modification and updating of the parameter data sets in the LEAs and thus simple product changes and replacement of LEAs become possible. In addition to LEA parameters, management of layout data for labels also appears to make sense in the context of logistics.

The above table describes a list of possible LOL functions. Not all these functions are necessary and available in every LOL. When implementing a LOL, it is therefore always necessary to check which functions are necessary for the given use case.

Once the functional scope of a LOL has been defined, it is still necessary to check which of the functions to be implemented need to be newly developed and which functions can be based on existing solutions. Functions 10 - 12 in the above list are specific functions of a LOL. Therefore, these functions have to be newly developed in any case, since no solutions exist for them yet. All other functions are derived from current MES or POL functions. In those cases, it must be checked whether an adaptation of the existing solutions to the special features of modular logistics described in Table 8.1 is economically possible.

A LOL should have a modular structure so that the scope of functions can be defined specifically for a particular customer or use case and existing solutions can also be integrated. A microservice architecture is a suitable solution for this purpose. Each LOL function is then implemented as one or more microservices with open interfaces, enabling information to be exchanged between the different LOL functions.

9 Enhancements of the Module Type Package Concept

The Module Type Package concept has originally been developed for the process industry. For a purposeful application in the context of production-related logistics, a further development is necessary, which is presented in the following sections. In course of this development, care has been taken not to change the state machine or any other existing constructs (interface definitions, model definitions and mechanisms) of the MTP concept. However, reinterpretations and reasonable specifications of new MTP constructs have been developed. This section summarizes the identified and partly already implemented reinterpretations and further developments structured according to the different sheets of VDI/VDE/NAMUR 2658 [19].

Necessary new model and interface definitions are specified in Sections 10, 11 and 12. For this, the notations used in the MTP guideline are applied. The modelling of the contents of an MTP follows IEC 62714-1:2018 [20] (AutomationML standard). The newly specified model and interface definitions are marked with a red border in the corresponding figures.

9.1 VDI/VDE/NAMUR 2658-1

In the VDI/VDE/NAMUR 2658-1 [8] (also referred to as Sheet 1) an overview is given of the basic concepts of the Module Type Package, on which all other sheets are based. In the context of production-related logistics, the new parameter types *StructServParam* and *ArrayServParam* (see Section 3.2.2) have revealed the need for user-defined complex data types. This has been recognized not only in logistics but also in other industries. Therefore, a possibility for modelling such user-defined data types has already been incorporated and published in a revision of Sheet 1. In addition, in the context of the HMI modelling (see Section 4), content from Sheet 1 is used, specifically the possibility of adding attachments to an MTP.

9.2 VDI/VDE/NAMUR 2658-2

VDI/VDE/NAMUR 2658-2 [4] (also referred to as Sheet 2) presents a concept for the vendor-independent modelling of HMIs in a MTP. Because of the origin of the MTP concept from the process industry, this concept has so far been designed for P&ID-like HMIs. However, in production-related logistics machine-oriented HMIs are better suited. Section 4 therefore introduces a concept for MTP-based modelling of such HMIs.

9.3 VDI/VDE/NAMUR 2658-3

VDI/VDE/NAMUR 2658-3 [5] (also referred to as Sheet 3) describes a library for data objects. This refers to typical data objects of the process industry, such as valves or drives, but also general objects for displaying, monitoring, and operating values of different data types and their metadata. With regard to this sheet, two needs for action have been identified in the context of production-related logistics.

On the one hand, data objects necessary for logistics or general unit load processes need to be added. Thereby, new interface definitions for *IndicatorElements* and *OperationElements* of Array and Struct types seem possible and reasonable. These are specified in Section 10.8. In addition, further interfaces are conceivable, but are not yet specified in this document. These include piece counts (without unit), resettable counters, time displays, drag indicator displays, etc.

On the other hand, the interfaces specified so far mainly in Sheets 3 and 4 are too extensive for most unit load processes as they occur in logistics. Therefore, approaches to simplify these interfaces are presented in Section 5.

9.4 VDI/VDE/NAMUR 2658-4

VDI/VDE/NAMUR 2658-4 [3] (also referred to as Sheet 4) describes a service concept for process equipment assemblies (PEAs). The automation of LEAs shall also be done using MTP-compliant services. This is based on two logistics-specific interpretations of the existing MTP state machine – an order-oriented Cyclic Execution Service (CES) and a demand-oriented Single Execution Service (SES). Thereby, the CES and SES execution types are implemented as different procedures of the logistics services within a LEA. This does not result in any adaptations of the MTP concept regarding state-based process control, only more specific interpretations of the state machine and *FunctionClassificationAttributes* for the unique identification of CES and SES procedures.

The parameterization of LEAs can basically be implemented with procedure or configuration parameters in the sense of the MTP concept. However, the parameterization of logistics services with *StructServParam* and *ArrayServParam* requires two new MTP parameter interfaces (see Section 10.8). In addition, semantic information in the form of *FunctionClassificationAttributes* should be added to all parameter interfaces (similar to those already existing at the service and procedure level). Besides the new parameter interfaces, also *ProcessValuesIns*, *ProcessValueOuts* and *ReportValues* of struct and array types seem meaningful and are specified in Section 10.8. Beyond this, for logistic applications, an interface for the transmission of layout data to labelling LEAs, e.g., in the form of a binary large object, is also required. However, this has not yet been specified in detail and will be added to this document in due course. The need for a method interface for LEAs is envisaged in the future as well for writing consistent parameter sets. The definition of such an interface is already in progress in the MTP standardization committees. In addition to parameterization on the initiative of the orchestration layer (here: LOL), as envisaged by previous MTP concepts, three defined requests from the LEAs to the LOL are useful in the context of production-related logistics (see Section 3.2.3). For these, similar to the MTP service interaction in VDI/VDE/NAMUR 2658-4 [3], a logistics interaction is provided that implements these logistics-specific requests from an LEA to the LOL.

The concepts for the enhancement of VDI/VDE/NAMUR 2658-4 [3] are presented in detail in Section 3. Associated new model definitions and interface definitions are specified in Section 10.

9.5 VDI/VDE/NAMUR 2658-5

VDI/VDE/NAMUR 2658-5 [21] (also referred to as Sheet 5) describes the communication of PEAs. This currently refers to the communication between PEAs and a POL. As shown in Section 6, the automation of Logistics Lines shall be implemented by the concept of Automation Services Choreographies. This concept is based on direct LEA-to-LEA communication without a LOL as a broker. For this reason, Sheet 5 must be enhanced to include the necessary specification content for direct module-to-module communication. However, these have not been specified yet.

9.6 VDI/VDE/NAMUR 2658-X: ChoreographySet

For the coordination of physically coupled Logistics Lines, the concept of Automation Services Choreographies is presented in Section 6. This is not yet included in the MTP standard. Accordingly, model and interface definitions for configuring choreographies do not yet exist. Section 11 therefore introduces proposals for corresponding model and interface definitions of a *ChoreographySet*. These are intended to serve as a basis for standardizing Automatization Services Choreographies in the MTP standard.

9.7 VDI/VDE/NAMUR 2658-X: TransportSet

Section 7 introduces a concept for the coordination of flexible transport in Logistics Areas. This is currently not yet included in the MTP standard. Accordingly, there are not yet any model and interface definitions for configuring transports and transport nodes. Section 12 presents proposals for corresponding model and interface definitions of such a transport set. These are intended to serve as a basis for standardization of flexible transports in the MTP standard.

9.8 Further Enhancements

Logistics-specific enhancements may also arise in other sheets of the MTP standard. For example, it may be necessary for alarms to be provided at LEA level rather than at service or control module level. In addition, logistics-specific diagnostic functions may become necessary. However, these aspects have not yet been examined in detail and may be added in future versions of this document.

10 Specification of the Logistics Aspect

10.1 Extension of Service Parameters

For the parameterization of logistics services, two new interface definitions are necessary in addition to the existing interface definitions for service parameters – *StructServParam* and *ArrayServParam* (see Figure 10.1 and Figure 10.2).

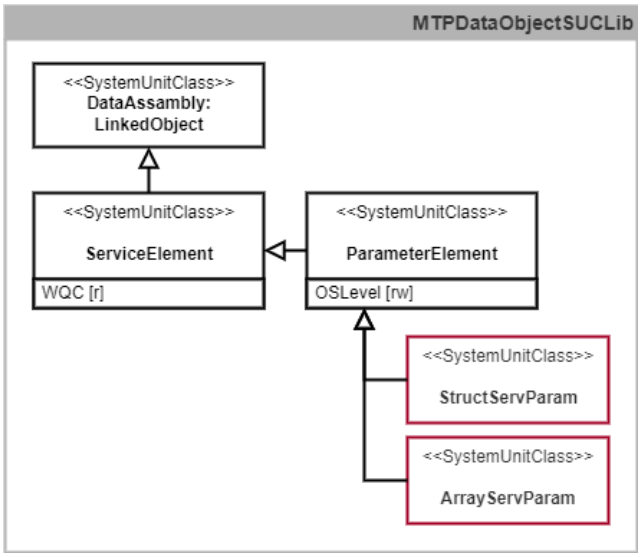

Figure 10.1: SUCs of the logistics aspect for extending the service parameters

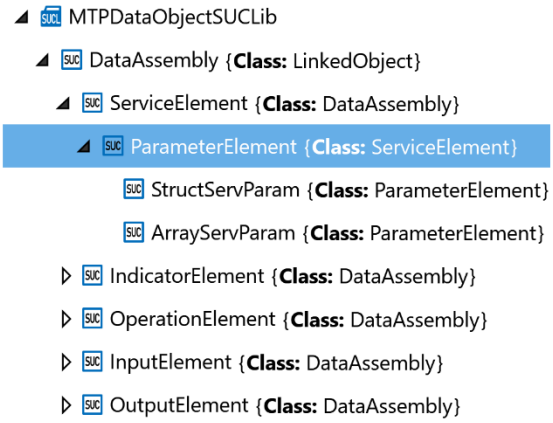

Figure 10.2: Interface definitions of the logistics aspect to extend the services parameters in the MTPDataObjectSUCLib

Like all other MTP service parameters, these are defined in the MTPDataObjectSUCLib and derived from the *ParameterElement* specified in VDI/VDE/NAMUR 2658-4 [3]. The new interface definitions are described in Sections 10.8.1 and 10.8.2.

In addition, an extension of the *ServiceParameter* model definition with semantic information (in the form of *FunctionClassfinationAttributes*) is proposed. This extension is described in Section 10.7.1.

10.2 Extension of Indicator Elements

StructView and *ArrayView* are conceivable, similar to the indicator element interfaces for all existing MTP data types (see Figure 10.3 and Figure 10.4).

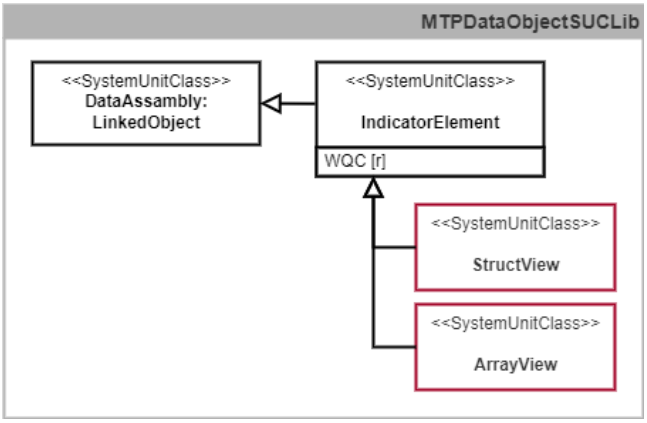

Figure 10.3: SUCs of the logistics aspect for extending the indicator elements

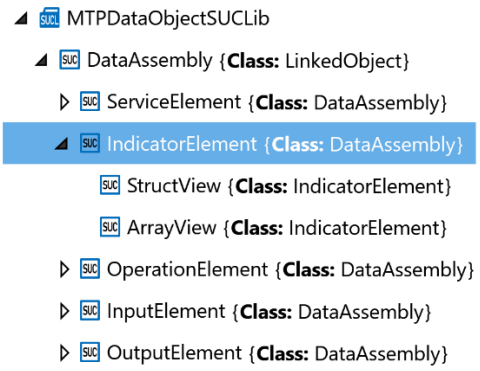

Figure 10.4: Interface definitions of the logistics aspect to extend the indicator elements in the MTPDataObjectSUCLib

Like all other MTP indicator elements, these are defined in the MTPDataObjectSUCLib and derived from the *IndicatorElement* specified in VDI/VDE/NAMUR 2658-4 [3]. The two new interface definitions are described in Sections 10.8.3 and 10.8.4. The interfaces can partly be also used for the corresponding process value outputs and report values.

10.3 Extension of Operation Elements

StructMan, *StructManInt*, *ArrayMan* and *ArrayManInt* are conceivable, similar to the operation element interfaces for all existing MTP data types (see Figure 10.5 and Figure 10.6).

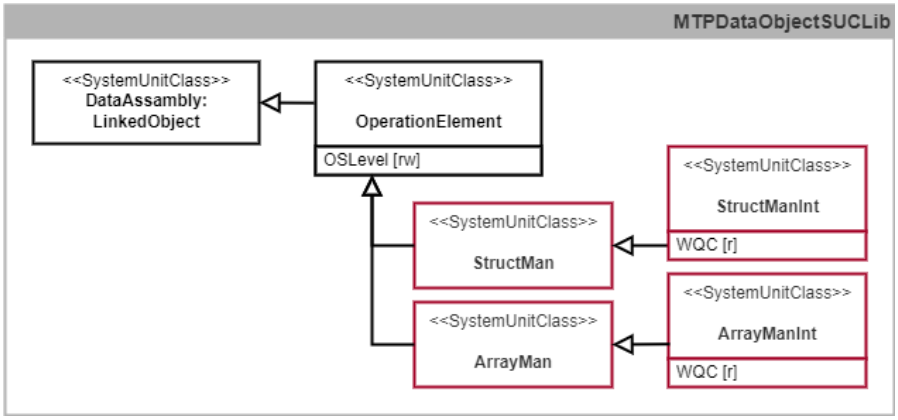

Figure 10.5: SUCs of the logistics aspect for extending the operation elements

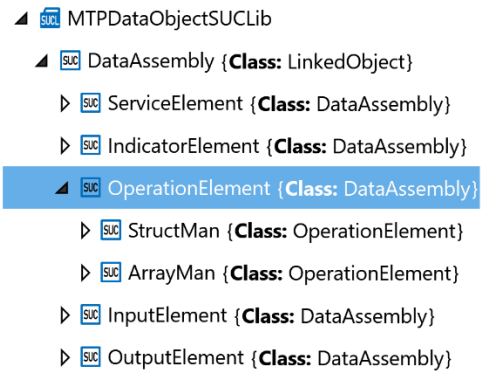

Figure 10.6: Interface definitions of the logistics aspect to extend the operation elements in the MTPDataObjectSUCLib

Like all other MTP operation elements, these are defined in the MTPDataObjectSUCLib. *StructMan* and *ArrayMan* are derived from the *OperationElement* specified in VDI/VDE/NAMUR 2658-4 [3]. *StructManInt* and *ArrayManInt* are derived from *StructMan* and *ArrayMan* respectively. The new interface definitions are described in Sections 10.8.5 to 10.8.8.

10.4 Extension of Process Value Inputs

StructProcessValueInputs and *ArrayProcessValueInputs* are conceivable, similar to the process value input interfaces for all existing MTP data types (see Figure 10.7 and Figure 10.8).

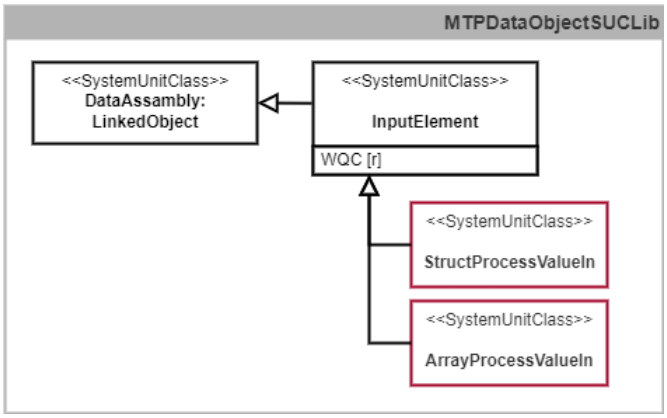

Figure 10.7: SUCs of the logistics aspect for extending the process value inputs

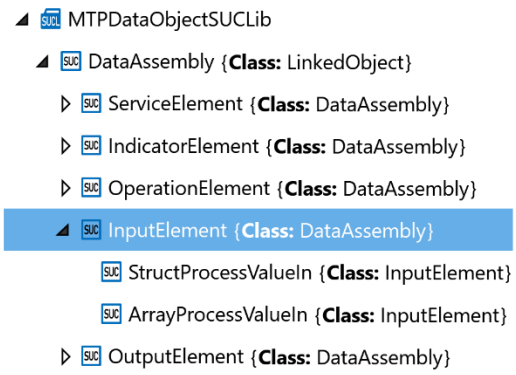

Figure 10.8: Interface definitions of the logistics aspect to extend the process value inputs in the MTPDataObjectSUCLib

Like all other MTP process value inputs, these are defined in the MTPDataObjectSUCLib and derived from the *InputElement* specified in VDI/VDE/NAMUR 2658-4 [3]. The two new interface definitions are described in Sections 10.8.11 and 10.8.13.

10.5 Extension of Process Value Outputs

For the process value outputs of structured data types, the associated *IndicatorElement* (*StructView*, see Section 10.2) can be used as for all other MTP data types. For process value outputs of the array data type, a separate *ArrayProcessValueOutput* must be modelled since this is different from the associated *IndicatorElement* (*ArrayView*, see Section 10.2).

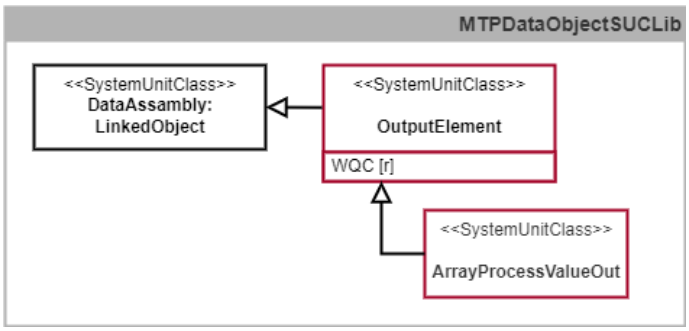

Figure 10.9: SUCs of the logistics aspect for introducing the *ArrayProcessValueOutput*

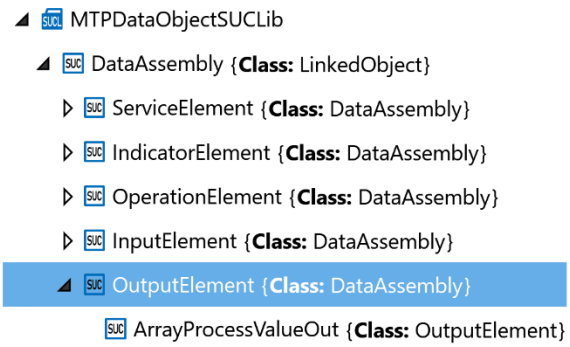

Figure 10.10: Interface definitions of the logistics aspect to introducing the *ArrayProcessValueOutputs* in the MTPDataObjectSUCLib

The *ArrayProcessValueOutput* is defined in the MTPDataObjectSUCLib and derived from the newly defined *OutputElement* which in turn is derived from *DataAssembly* following VDI/VDE/NAMUR 2658-1 [8]. For more semantic clarity and with regard to possible further developments, it should be considered in the context of MTP standardization to explicitly model the process value outputs of all data types and to derive them from the *OutputElement* as well. The two new interface definition are described in Sections 10.8.14 and 10.8.15.

10.6 Specification of Logistics Interaction

The logistics interactions between a LEA and a LOL described in Section 3.2.3 are based on the principle of service interaction described in VDI/VDE/NAMUR 2658-4 [3]. However, new SUCs and RCs are required to represent the logistics interactions in the IH of an MTP, which are shown in Figure 10.11.

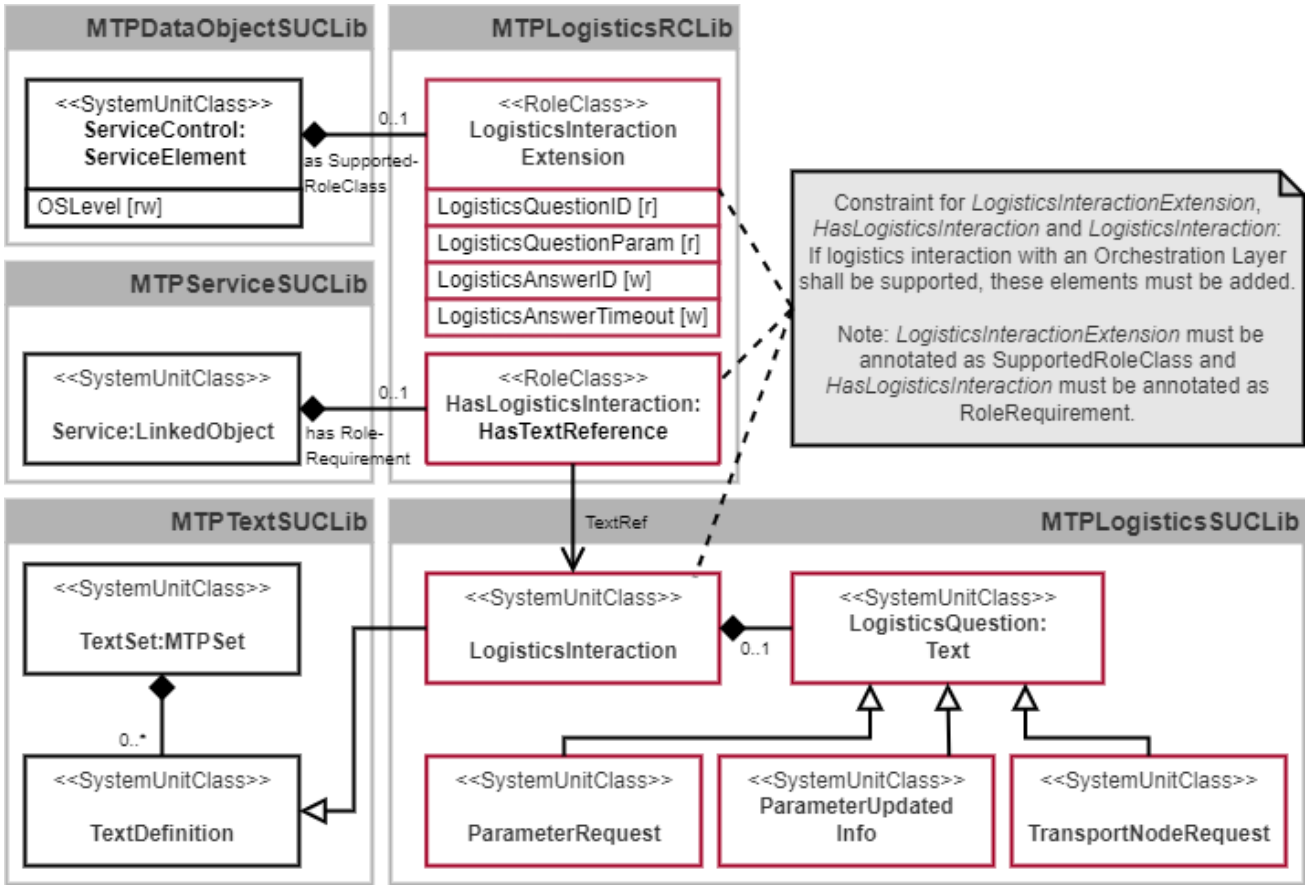

Figure 10.11: SUCs of the logistics aspect for the implementation of the logistics interaction

The SUC *LogisticsInteraction* derived from the SUC *TextDefinition* specified in VDI/VDE/ NAMUR 2658-4 [3] organizes all necessary new model definitions. This primarily comprises a generic *LogisticsQuestion* derived from the SUC *Text* from VDI/VDE/NAMUR 2658-4. Three concrete logistics-specific questions are derived from the abstract *LogisticsQuestion* – *ParameterRequest*, *ParameterUpdatedInfo* and *TransportNodeRequest*. A LEA can provide each of these *LogisticsQuestions* either not at all or exactly once subordinated to its SUC *LogisticsInteraction*.

These model definitions are based on the types of MTPLogisticsSUCLib shown in Figure 10.12.

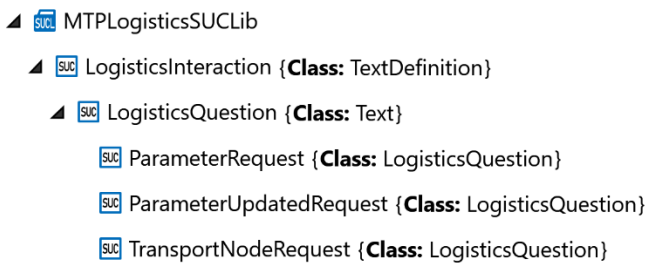

Figure 10.12: Model definitions of the logistics aspect for the implementation of the logistics interaction in the MTPLogisticsSUCLib

In addition to these new model definitions, extensions to existing model and interface definitions are also necessary and must be inserted if logistics interaction is foreseen in the LEA. Such optional extensions are to be implemented in the MTP specification by means of RCs. In the present case, an extension of the *ServiceControl* interface definition (specified in VDI/VDE/NAMUR 2658-4 [3]) is necessary, which is implemented by the RC *LogisticsInteractionExtension*. In addition, it must be signalled at the model definition *Service* (specified in VDI/VDE/NAMUR 2658-4 [3]) by means of the RC *HasLogisticsInteraction* that a logistics interaction is assigned to the service. The RC *HasLogisticsInteraction* is derived from the RC *HasTextReference* from VDI/VDE/NAMUR 2658-4. A concrete logistics interaction is assigned to the service by means of a text reference to the model definition of the *LogisticsInteraction*.

These extensions are defined in the MTPLogisticsRCLib shown in Figure 10.13.

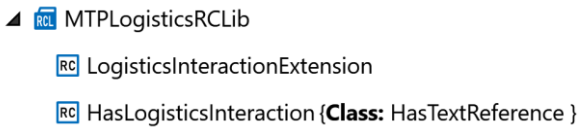

Figure 10.13: Model and interface extensions of the logistics aspect for the implementation of the logistics interaction in the MTPLogisticsRCLib

All model and interface definitions required for the logistics interaction are specified in Sections 10.7 and 10.8.

10.7 Model Definitions

10.7.1 Extension of the ServiceParameter

The SUC *ServiceParameter* (see Table 10.1) defines the base class for MTP service parameters of all data types. This model definition is already specified in VDI/VDE/NAMUR 2658-4 [3] and is extended here by semantic information in form of a *FunctionClassificationAttribute*.

Table 10.1: Model definition of ServiceParameter

Name	**ServiceParameter**
Type	SystemUnitClass
Description	base model definition of service parameter
Hierarchy	MTPServiceSUCLib
Parent	MTPSUCLib/LinkedObject
RoleClasses	

Version	ModuleTypePackage:Logistics (V0.0.1)		
Properties			
Name	**Type**	**Description**	
-	-	-	
Attributes			
Name	**Type**	**Description**	***AttributeType Reference***
Classification	**<empty>**	**list of child attributes of AttributeType FunctionClassificationAttribute**	**OrderedListType**
Comment			
-			

10.7.2 LogisticsInteraction

The SUC *LogisticsInteraction* (see Table 10.2) organizes all necessary model definition for logistics interaction between a LEA and a LOL. This primarily includes the *LogisticsQuestions* available in the LEA. The *LogisticsInteraction* is derived from the *TextDefinition* specified in VDI/VDE/NAMUR 2658-4 [3]. This model definition is linked to the model definition *HasLogisticsInteraction* via a TextRef. The *LogisticsInteraction* follows a similar principle as the service interaction specified in the VDI/VDE/NAMUR 2658-4 with adaptations that are described in Section 3.2.3.

Table 10.2: Model definition of LogisticsInteraction

Name	**LogisticsInteraction**		
Type	SystemUnitClass		
Description	Model definition for logistics-specific service interaction		
Hierarchy	MTPLogisticsTextSUCLib		
Parent	MTPTextSUCLib/TextDefinition		
RoleClasses			
Version	ModuleTypePackage:Logistics (V0.0.1)		
Properties			
Name	**Type**	**Description**	
-	-	-	
Attributes			
Name	**Type**	**Description**	***AttributeType Reference***
-	-	-	-
Comment			
-			

10.7.3 LogisticsQuestion

The SUC *LogisticsQuestion* (see Table 10.3) is an abstract class derived from the SUC *Text* from VD/VDE/NAMUR 2658-4 [3] representing a logistics-specific question that a LEA can pose to a LOL. There are three specific questions derived from the *LogisticsQuestion* so far – *ParameterRequest, ParameterUpdatedInfo*, and *TransportNodeRequest*. Each of these questions can occur either not at all or exactly once in a LEA.

Table 10.3: Model definition of LogisticsQuestion

Name	**LogisticsQuestion**
Type	SystemUnitClass

Description	Model definition for a generic Question for logistics-specific service interactions		
Hierarchy	MTPLogisticsSUCLib/LogisticsInteraction		
Parent	MTPTextSUCLib/Text		
RoleClasses			
Version	ModuleTypePackage:Logistics (V0.0.1)		
Properties			
Name	**Type**	**Description**	
Name	xs:string	unique number of the question (>= 0)	
Attributes			
Name	**Type**	**Description**	***AttributeType Reference***
-	-	-	-
Comment			
-			

10.7.4 ParameterRequest

The SUC *ParameterRequest* (see Table 10.4) is derived from *LogisticsQuestion* and is used to request parameter sets from a LOL. In contrast to the *Question* specified in VDI/VDE/NAMUR 2658-4, no answers are modelled in the MTP for the *ParameterRequest*. Instead, a value in the number range of DINT is expected as an answer. Numbers greater than or equal to 0 indicate the index at which the LOL has written the requested parameter set to the parameter data storage of the equipment assembly. Thereby the limits of the array (minimum and maximum index) must not be exceeded or undercut. Value "-1" indicates that there is no response yet. Value "-2" indicates that an error occurred during the request. Other responses so far are not necessary and not valid.

Table 10.4: Model definition of ParameterRequest

Name	**ParameterRequest**		
Type	SystemUnitClass		
Description	Model definition for requesting parameter sets from a Logistics Orchestration Layer		
Hierarchy	MTPLogisticsSUCLib/LogisticsInteraction/LogisticsQuestion		
Parent	MTPLogisticsSUCLib/LogisticsInteraction/LogisticsQuestion		
RoleClasses			
Version	ModuleTypePackage:Logistics (V0.0.1)		
Properties			
Name	**Type**	**Description**	
-	-	-	
Attributes			
Name	**Type**	**Description**	***AttributeType Reference***
-	-	-	-
Comment			
-			

10.7.5 ParameterUpdatedInfo

The SUC *ParameterUpdatedInfo* (see Table 10.5) is derived from the *LogisticsQuestion* and is used to inform a LOL that a parameter set in the LEA has changed. In contrast to the *Question* specified in VDI/VDE/NAMUR 2658-4 [3], no *Answers* are modelled in the MTP for the *ParameterUpdatedInfo*. Instead, the value "1" is expected as confirmation that the LOL has acknowledged the parameter change.

Value “-1” indicates that there is no response yet. Value -2 indicates that an error occurred during the request. Other responses so far are not necessary and not valid.

Table 10.5: Model definition of ParameterUpdatedInfo

Name	**ParameterUpdatedInfo**		
Type	SystemUnitClass		
Description	Model definition for informing a LOL of a change in a parameter set		
Hierarchy	MTPLogisticsSUCLib/LogisticsInteraction/LogisticsQuestion		
Parent	MTPLogisticsSUCLib/LogisticsInteraction/LogisticsQuestion		
RoleClasses			
Version	ModuleTypePackage:Logistics (V0.0.1)		
Properties			
Name	**Type**	**Description**	
-	-	-	
Attributes			
Name	**Type**	**Description**	***AttributeType Reference***
-	-	-	-
Comment			
-			

10.7.6 TransportNodeRequest

The SUC *TransportNodeRequest* (see Table 10.6) is derived from the *LogisticsQuestion* and is used to request the next transport node to be approached from a LOL. In contrast to the *Question* specified in VDI/VDE/NAMUR 2658-4 [3], no answers are modelled in the MTP for the *TransportNodeRequest*. Instead, a value in the number range of DINT is expected as answer. Numbers greater than 0 indicate the ID of the next transport node to be approached. Thereby, only values that correspond to the ID of a transport node in the respective logistics system may be returned as answer. Value “-1” indicates that there is no response yet. Value “-2” indicates that an error occurred during the request. Other responses so far are not necessary and not valid.

Table 10.6: Model definition of TransportNodeRequest

Name	**TransportNodeRequest**		
Type	SystemUnitClass		
Description	Model definition for requesting the next transport node to be approached from a Logistics Orchestration Layer		
Hierarchy	MTPLogisticsSUCLib/LogisticsInteraction/LogisticsQuestion		
Parent	MTPLogisticsSUCLib/LogisticsInteraction/LogisticsQuestion		
RoleClasses			
Version	ModuleTypePackage:Logistics (V0.0.1)		
Properties			
Name	**Type**	**Description**	
-	-	-	
Attributes			
Name	**Type**	**Description**	***AttributeType Reference***
-	-	-	-
Comment			
-			

10.7.7 HasLogisticsInteraction

The RC *HasLogisticsInteraction* (see Table 10.7) is derived from the RC *HasTextReference* specified in the VDI/VDE/NAMUR 2658-4 [3]. *HasLogisticsInteraction* is used to assign a logistics interaction to the model definition *Service* (specified in VDI/VDE/NAMUR 2658-4 [3]). For this purpose, a *LogisticsInteraction* interface definition is referenced by means of a text reference. If a logistics interaction of the LEA is intended, exactly one *HasLogisticsInteraction* is to be assigned to the service as RoleRequirement, otherwise none.

Table 10.7: Model definition of HasLogisticsInteraction

Name	**HasLogisticsInteraction**		
Type	RoleClass		
Description	Model definition for assigning a logistics interaction to a service		
Hierarchy	MTPLogisticsRCLib		
Parent	MTPTextRCLib/HasTextReference		
RoleClasses			
Version	ModuleTypePackage:Logistics (V0.0.1)		
Properties			
Name	**Type**	**Description**	
-	-	-	
Attributes			
Name	**Type**	**Description**	***AttributeType Reference***
-	-	-	-
Comment			
-			

10.8 Interface Definitions

10.8.1 StructServParam

The SUC *StructServParam* (see Table 10.8) is used to pass parameters of a user-defined structured data type from a LOL to a LEA.

Table 10.8: Interface definition of StructServParam

Name	**StructServParam**			
Type	SystemUnitClass			
Description	Generic parameter interface for a structured data type following the rules of modelling complex data types			
Hierarchy	MTPDataObjectSUCLib/DataAssembly/ServiceElement/ParameterElement			
Parent	MTPDataObjectSUCLib/DataAssembly/ServiceElement/ParameterElement			
RoleClasses				
Version	ModuleTypePackage:Logistics (V0.0.1)			
Alias	**Access**	**Type**	**Description**	**IRDI**
VExt	LOL → LEA	{VType}	External Value	
VInt	LOL ← LEA	{VType}	Internal Value	
VOp	LOL → LEA	{VType}	Operator Value	
VReq	LOL ← LEA	{VType}	Requested Value	
VOut	LOL ← LEA	{VType}	Output Value	
VType	MTP	Type derived from Structured-DataType	Type Definition of the Values	

The special characteristic of this interface definition is the use of a user-defined data type. Figure 10.14 shows how such a data type can be modelled. The rules for modelling complex data types from VDI/VDE/NAMUR 2658-1 [8] are used.

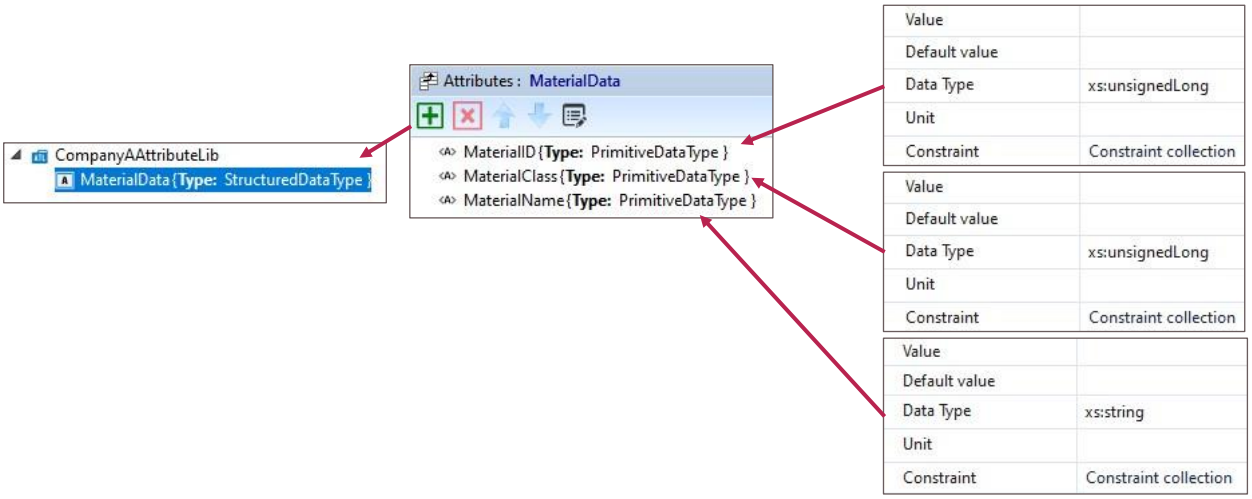

Figure 10.14: Modelling a custom data type

The used complex data type must be derived from the AT *StructuredDataType* defined in VDI/VDE/NAMUR 2658-1 [8]. When using this interface, a user-defined ATL (here: CompanyAAttributeLib) must be created. Within this ATL, the structured data type to be used later in the instance of the *StructServParam* interface definition must be specified. With the assignment of this user-defined AT to the attribute *VType* of the *StructServParam* the used structured data type is defined. This data type is then expected behind the variables *VExt*, *VInt*, *VOp*, *VReq* and *VOut*. The setting of a parameter of the *StructServParam* type is done via the access channels Automatic Internal, Automatic External or Operator in the same way as the setting of all other service parameters defined in VDI/VDE/NAMUR 2658-4 [3].

10.8.2 ArrayServParam

The SUC *ArrayServParam* (see Table 10.9) is used for the LOL to manage an array located in a LEA.

Table 10.9: Interface definition of ArrayServParam

Name	**ArrayServParam**			
Type	SystemUnitClass			
Description	Generic parameter interface for an array data type following the rules of modelling complex data types			
Hierarchy	MTPDataObjectSUCLib/DataAssembly/ServiceElement/ParameterElement			
Parent	MTPDataObjectSUCLib/DataAssembly/ServiceElement/ParameterElement			
RoleClasses				
Version	ModuleTypePackage:Logistics (V0.0.1)			
Alias	**Access**	**Type**	**Description**	**IRDI**
IndexExt	LOL → LEA	DINT	External Index Value	
IndexInt	LOL ← LEA	DINT	Internal Index Value	
IndexOp	LOL ← LEA	DINT	Operator Index Value	
IndexMin	LOL ← LEA	DINT	Low Limit of the Index	
IndexMax	LOL ← LEA	DINT	High Limit of the Index	
IndexCur	LOL ← LEA	DINT	Current Index Value	

VExt	LOL → LEA	{VType}	External Value	
VInt	LOL ← LEA	{VType}	Internal Value	
VOp	LOL ← LEA	{VType}	Operator Value	
VReq	LOL ← LEA	{VType}	Requested Value	
VOut	LOL ← LEA	{VType}	Output Value	
VType	MTP	Type from derived from Base Data Type	Type Definition of the Values	

The challenge with this interface definition is the management of an array of variables with undefined length. This is often not possible in common automation solutions or only under certain conditions. Therefore, a multiplexer mechanism is used, which can access an array of arbitrary length via a structurally static interface.

With the variables *IndexExt*, *IndexInt* and *IndexOp* a pointer-like reference to an array element is defined considering the operation mode. Depending on the active access channel, the variable *IndexCur* is set to one of these three values. The variables of all three access channels are checked to see if they lie in the range between *IndexMin* and *IndexMax*. If an index is set that lies outside this range, the last valid index remains, and the Worst Quality Code (*WQC*) is set to "Out of Specification".

Depending on the value of the *IndexCur* variable, the array element with the corresponding index is selected for editing. The selected array element is processed according to the parameter transfer mechanism specified in VDI/VDE/NAMUR 2658-4 [3]. *VOut* always displays the set value of the array element located at the position of the array defined by *IndexCur*. It should be noted that this value does not necessarily have to match the value currently used in the LEA.

All primitive data types provided in the MTP concept as well as all complex data types according to the conventions from VDI/VDE/NAMUR 2658-1 [8] can be used as data type for the individual array elements. The selection of the data type used is made via the *VType* variable. In the case of a structured data type, the conventions described in Section 10.8.1 for creating a user-defined data type must be followed.

10.8.3 StructView

The SUC *StructView* (see Table 10.10) is used for a LOL to display a LEA variable of a user-defined structured data type.

Table 10.10: Interface definition of StructView

Name	**StructView**			
Type	SystemUnitClass			
Description	Generic interface for displaying a value of structured data type following the rules of modelling complex data types			
Hierarchy	MTPDataObjectSUCLib/DataAssembly/IndicatorElement			
Parent	MTPDataObjectSUCLib/DataAssembly/IndicatorElement			
RoleClasses				
Version	ModuleTypePackage:Logistics (V0.0.1)			
Alias	**Access**	**Type**	**Description**	**IRDI**
V	LOL ← LEA	{VType}	Value	
VType	MTP	Type from derived from Base Data Type	Type Definition of the Value	

The special feature of this interface is the use of a user-defined data type. The modelling and use of such a type has already been described in Section 10.8.1 in the context of the *StructServParam* and shall be done in the same way for the *StructView* interface.

10.8.4 ArrayView

The SUC *ArrayView* (see Table 10.11) is used for the LOL to view the value at a specific position of an array located in a LEA.

Table 10.11: Interface definition of ArrayView

Name	**ArrayView**			
Type	SystemUnitClass			
Description	Generic interface for displaying a value at a specific position of an array located in a LEA by a LOL			
Hierarchy	MTPDataObjectSUCLib/DataAssembly/IndicatorElement			
Parent	MTPDataObjectSUCLib/DataAssembly/IndicatorElement			
RoleClasses				
Version	ModuleTypePackage:Logistics (V0.0.1)			
Alias	**Access**	**Type**	**Description**	**IRDI**
OSLevel	LOL → LEA	BYTE	OSLevel variable	
Index	LOL → LEA	DINT	Index Value	
IndexMin	LOL ← LEA	DINT	Low Limit of the Index	
IndexMax	LOL ← LEA	DINT	High Limit of the Index	
IndexCur	LOL ← LEA	DINT	Current Index Value	
V	LOL ← LEA	{VType}	Output Value	
VType	MTP	Type from derived from Base Data Type	Type Definition of the Values	

Similar to the description in Section 10.8.2 for the *ArrayServParam*, the challenge for this interface is to access an array within a LEA, which can have an arbitrary length. As described in Section 10.8.2, access to this array should also be done in an index-based manner in case of the *ArrayView* interface.

The array position to be displayed is selected via the *Index* variable. The *IndexMin* and *IndexMax* variables indicate the upper and lower limits of the array. The *IndexCur* variable indicates the currently selected index, the value of the array at this point is displayed in *V*. *VType* defines the data type that all array elements have. This can be a primitive data type, or a user-defined data type as introduced in Section 10.8.1.

10.8.5 StructMan

The SUC *StructMan* (see Table 10.12) is used for the LOL to manipulate a LEA variable of a user-defined structured data type.

Table 10.12: Interface definition of StructMan

Name	**StructMan**
Type	SystemUnitClass
Description	Generic interface for manipulating a value of structured data type following the rules of modelling complex data types

Hierarchy	MTPDataObjectSUCLib/DataAssembly/OperationElement			
Parent	MTPDataObjectSUCLib/DataAssembly/OperationElement			
RoleClasses				
Version	ModuleTypePackage:Logistics (V0.0.1)			
Alias	**Access**	**Type**	**Description**	**IRDI**
VOut	LOL ← LEA	{VType}	Value Output	
VMan	LOL → LEA	{VType}	Manual Value	
VRbk	LOL ← LEA	{VType}	Readback Value	
VFbk	LOL ← LEA	{VType}	Feedback	
VType	MTP	Type from derived from Base Data Type	Type Definition of the Value	

VMan is used to enter the desired value of the variable. Following the concept specified in VDI/VDE/NA-MUR 2658-3 [5], *VRbk* is used to verify the communication between a LOL and the StructMan interface within a LEA and displays the raw value communicated to the LEA. *VOut* displays the value given to a further LEA internal block possibly with limitations applied. *VFbk* variable is used to display the current value of the structure affected by the *StructMan* interface. The special feature of this interface is the use of a user-defined data type. The modelling and use of such a type has already been described in Section 10.8.1 in the context of the *StructServParam* and shall be done in the same way for the *StructMan* interface.

10.8.6 StructManInt

The SUC *StructManInt* (see Table 10.13) is used for manipulating a LEA variable of a user-defined structured data type from inside the LEA or by the LOL.

Table 10.13: Interface definition of StructManInt

Name	**StructManInt**			
Type	SystemUnitClass			
Description	Generic interface for manipulating a value of structured data type following the rules of modelling complex data types by the LOL or from inside the LEA			
Hierarchy	MTPDataObjectSUCLib/DataAssembly/OperationElement/StructMan			
Parent	MTPDataObjectSUCLib/DataAssembly/OperationElement/StructMan			
RoleClasses				
Version	ModuleTypePackage:Logistics (V0.0.1)			
Alias	**Access**	**Type**	**Description**	**IRDI**
WQC	LOL ← LEA	BYTE	Worst Quality Code variable	
VMan[3]	*LOL → LEA*	*{VType}*	*(relevant, if SrcManAct is true, see SourceMode) Manual Value*	
VInt	LOL ← LEA	{VType}	(relevant, if SrcIntAct is true, see SourceMode) Internal Value	
SrcChannel	LOL ← LEA	BOOL	selection of the active SourceMode interaction channel 0: The operator switches (*Op) shall be used. 1: The automatic switches (*Aut) shall be used.	

[3] This variable is inherited from the *StructMan* interface. However, its meaning changes slightly in this case since it is only used when the *SourceMode* is set to manual.

SrcManAut	LOL ← LEA	BOOL	Set SourceMode to Manual by automatic interaction. (relevant, if SrcChannel is true) 1: SourceMode is set to Manual. 0: no operation	
SrcIntAut	LOL ← LEA	BOOL	Set SourceMode to Internal by automatic interaction (relevant, if SrcChannel is true). 1: SourceMode is set to Internal. 0: no operation	
SrcIntOp	LOL → LEA	BOOL	Set SourceMode to Internal by operator interaction (relevant, if SrcChannel is false). 0→1: request to set OperationMode to Internal 1→0: acknowledge by PEA	
SrcManOp	LOL → LEA	BOOL	Set SourceMode to Manual by operator interaction (relevant, if SrcChannel is false). 0→1: request to set OperationMode to Manual 1→0: acknowledge by PEA	
SrcIntAct	LOL ← LEA	BOOL	1: current mode is Internal 0: current mode is not Internal	
SrcManAct	LOL ← LEA	BOOL	1: current mode is Manual 0: current mode is not Manual	

The *StructManInt* interface extends the *StructMan* interface, described in Section 10.8.5, by the internal value specification and a source mode in accordance with VDI/VDE/NAMUR 2658-3 [5]. If the internal access channel is selected, a LEA internal value is used instead of the external value setting. Apart from that, the function of this interface is the same as that of the *StructMan* interface.

10.8.7 ArrayMan

The SUC *ArrayMan* (see Table 10.14) is used for the LOL to manipulate a value at a specific position of an array located in a LEA.

Table 10.14: Interface definition of ArrayMan

Name	**ArrayMan**			
Type	SystemUnitClass			
Description	Generic interface for the LOL to manipulate a value at a specific position of an array located in a LEA			
Hierarchy	MTPDataObjectSUCLib/DataAssembly/OperationElement			
Parent	MTPDataObjectSUCLib/DataAssembly/OperationElement			
RoleClasses				
Version	ModuleTypePackage:Logistics (V0.0.1)			
Alias	**Access**	**Type**	**Description**	**IRDI**
VOut	LOL ← LEA	{VType}	Value Output	
Index	LOL → LEA	DINT	Index Value	
IndexMin	LOL ← LEA	DINT	Low Limit of the Index	
IndexMax	LOL ← LEA	DINT	High Limit of the Index	
IndexCur	LOL ← LEA	DINT	Current Index Value	
VMan	LOL → LEA	{VType}	Manual Value	

VRbk	LOL ← LEA	{VType}	Readback Value	
VFbk	LOL ← LEA	{VType}	Feedback	
VType	MTP	Type from derived from Base Data Type	Type Definition of the Values	

Similar to the description in Section 10.8.2 for the *ArrayServParam*, the challenge for this interface is to access an array within a LEA, which can have an arbitrary length. As described in Section 10.8.2, access to this array should also be done in an index-based manner in the case of the *ArrayMan* interface.

The array position to be manipulated is selected via the *Index* variable. The *IndexMin* and *IndexMax* variables indicate the upper and lower limits of the array. The *IndexCur* variable indicates the currently selected index of variable to be manipulated. The *VMan* variable is used to enter the desired value of this variable. Following the concept specified in VDI/VDE/NAMUR 2658-3 [5], *VRbk* is used to verify the communication between a LOL and the *ArrayMan* interface within a LEA and displays the raw value of the variable communicated to the LEA. When a new Index is selected, the *VMan* and *VRbk* variables are set to the value at the selected position in the array. *VOut* displays the value given to a further LEA internal block possibly with limitations applied. *VFbk* variable is used to display the current value of the structure affected by the *ArrayMan* interface. *VType* defines the data type that all array elements have. This can be a primitive data type, or a user-defined data type as introduced in Section 10.8.1.

10.8.8 ArrayManInt

The SUC *ArrayManInt* (see Table 10.15) is used for the LOL or for a LEA internal logic to manipulate a value at a specific position of an array located in a LEA.

Table 10.15: Interface definition of ArrayManInt

Name	**ArrayManInt**			
Type	SystemUnitClass			
Description	Generic interface for the LOL or for a LEA internal logic to manipulate a value at a specific position of an array located in a LEA			
Hierarchy	MTPDataObjectSUCLib/DataAssembly/OperationElement/ArrayMan			
Parent	MTPDataObjectSUCLib/DataAssembly/OperationElement/ArrayMan			
RoleClasses				
Version	ModuleTypePackage:Logistics (V0.0.1)			
Alias	**Access**	**Type**	**Description**	**IRDI**
WQC	LOL ← LEA	BYTE	Worst Quality Code variable	
VMan[4]	*LOL → LEA*	*{VType}*	*(relevant, if SrcManAct is true, see SourceMode) Manual Value*	
VInt	LOL ← LEA	{VType}	(relevant, if SrcIntAct is true, see SourceMode) Internal Value	
SrcChannel	LOL ← LEA	BOOL	selection of the active SourceMode interaction channel 0: The operator switches (*Op) shall be used. 1: The automatic switches (*Aut) shall be used.	

[4] This variable is inherited from the *ArrayMan* interface. However, its meaning changes slightly in this case since it is only used when the *SourceMode* is set to manual.

SrcManAut	LOL ← LEA	BOOL	Set SourceMode to Manual by automatic interaction. (relevant, if SrcChannel is true) 1: SourceMode is set to Manual. 0: no operation	
SrcIntAut	LOL ← LEA	BOOL	Set SourceMode to Internal by automatic interaction (relevant, if SrcChannel is true). 1: SourceMode is set to Internal. 0: no operation	
SrcIntOp	LOL → LEA	BOOL	Set SourceMode to Internal by operator interaction (relevant, if SrcChannel is false). 0→1: request to set OperationMode to Internal 1→0: acknowledge by PEA	
SrcManOp	LOL → LEA	BOOL	Set SourceMode to Manual by operator interaction (relevant, if SrcChannel is false). 0→1: request to set OperationMode to Manual 1→0: acknowledge by PEA	
SrcIntAct	LOL ← LEA	BOOL	1: current mode is Internal 0: current mode is not Internal	
SrcManAct	LOL ← LEA	BOOL	1: current mode is Manual 0: current mode is not Manual	

The *ArrayManInt* interface extends the *ArrayMan* interface, described in Section 10.8.7, by the internal value specification and a source mode in accordance with VDI/VDE/NAMUR 2658-3 [5]. If the internal access channel is selected, a LEA internal value is used instead of the external value setting. Apart from that, the function of this interface is the same as that of the *ArrayMan* interface.

10.8.9 StructReportValue

According to VDI/VDE/NAMUR 2658-4 [3], the same interface definitions are used for report values as for the corresponding *IndicatorElements*. However, the value of a report value can be frozen triggered by a variable on the *ServiceControl* interface. Optionally, a *MissedValueFlag* can be added to the interface definition of a report value. This principle is also adopted for the *StructReportvalue* interface definition. This is based on the *StructView* interface definition described in Section 10.8.3.

10.8.10 ArrayReportValue

According to VDI/VDE/NAMUR 2658-4 [3], the same interface definitions are used for report values as for the corresponding *IndicatorElements*. However, the value of a report value can be frozen triggered by a variable on the *ServiceControl* interface. Optionally, a *MissedValueFlag* can be added to the interface definition of a report value. This principle is also adopted for the *ArrayReportValue* interface definition. This is based on the *ArrayView* interface definition described in Section 10.8.4.

If several or all values of an array shall be read out for documentation purposes, several, or all indices between *IndexMin* and *IndexMax* have to be set at the *ArrayReportValue* interface one after the other by the LOL. Afterwards they can be stored one after the other.

10.8.11 StructProcessValueIn

The SUC *StructProcessValueIn* (see Table 10.16) is used for a LEA to access the value of structured data type of another LEA.

Table 10.16: Interface definition of StructProcessValueIn

Name	**StructProcessValueIn**			
Type	SystemUnitClass			
Description	Generic interface for accessing a value of structured data type from another LEA			
Hierarchy	MTPDataObjectSUCLib/DataAssembly/InputElement			
Parent	MTPDataObjectSUCLib/DataAssembly/InputElement			
RoleClasses				
Version	ModuleTypePackage:Logistics (V0.0.1)			
Alias	**Access**	**Type**	**Description**	**IRDI**
V	LOL → LEA	{VType}	Value	
VType	MTP	Type from derived from Base Data Type	Type Definition of the Value	

In the *V* variable the desired value is transferred. The special feature of this interface is the use of a user-defined data type. The modelling and use of such a type has already been described in Section 10.8.1 in the context of the *StructServParam* and shall be done in the same way for the *StructProcessValueIn* interface.

10.8.12 StructProcessValueOut

According to VDI/VDE/NAMUR 2658-4 [3], the interface definitions of the *IndicatorElement* or its derivatives are used for process value outputs. Accordingly, for *StructProcessValueOuts* the *StructView* interface definition specified in Section 10.8.3 is used.

10.8.13 ArrayProcessValueIn

The SUC *ArrayProcessValueIn* (see Table 10.17) is used for a LEA to access a value at a specific position of an array located in another LEA.

Table 10.17: Interface definition of ArrayProcessValueIn

Name	**ArrayProcessValueIn**			
Type	SystemUnitClass			
Description	Generic interface for accessing a value of array data type from another LEA			
Hierarchy	MTPDataObjectSUCLib/DataAssembly/InputElement			
Parent	MTPDataObjectSUCLib/DataAssembly/InputElement			
RoleClasses				
Version	ModuleTypePackage:Logistics (V0.0.1)			
Alias	**Access**	**Type**	**Description**	**IRDI**
Index	LOL ← LEA	DINT	Operator Index Value	
IndexMin	LOL → LEA	DINT	Low Limit of the Index	
IndexMax	LOL → LEA	DINT	High Limit of the Index	
IndexCur	LOL → LEA	DINT	Current Index Value	

V	LOL → LEA	{VType}	Output Value	
VType	MTP	Type from derived from Base Data Type	Type Definition of the Values	

Similar to the description in Section 10.8.2 for the *ArrayServParam*, the challenge for this interface is to access an array within a LEA, which can have an arbitrary length. As described in Section 10.8.2, access to this array should also be done in an index-based manner in case of the *ArrayProcessValueIn* interface.

The array position to be displayed is selected via the *Index* variable. The *IndexMin* and *IndexMax* variables indicate the upper and lower limits of the array. The *IndexCur* variable indicates the currently selected index, the value of the array at this point is displayed in *V*. *VType* defines the data type that all array elements have. This can be a primitive data type, or a user-defined data type as introduced in Section 10.8.1.

Note 1: So far, no use case exists for this interface definition, it is only listed here for the sake of completeness. If a use case for such an interface is identified, it should be implemented as shown in Table 10.17.

Note 2: This interface definition differs from all other interfaces derived from *InputElement* interface definitions, as it includes information flows from the LEA to the LOL. This is not intended so far.

10.8.14 OutputElement

The SUC *OutputElement* (see Table 10.18) is an abstract interface from which specific process value outputs of different data type can be derived. The interface definition itself fulfils only an organizational purpose and does not involve any own variables.

Table 10.18: Interface definition of OutputElement

Name	**OutputElement**			
Type	SystemUnitClass			
Description	Abstract interface from which process value outputs of different data type can be derived			
Hierarchy	MTPDataObjectSUCLib/DataAssembly			
Parent	MTPDataObjectSUCLib/DataAssembly			
RoleClasses				
Version	ModuleTypePackage:Logistics (V0.0.1)			
Alias	**Access**	**Type**	**Description**	**IRDI**
-	-	-	-	-

Note: Although the *IndicatorElements* of all other MTP data types and of the structured data type have the same interface definitions as the corresponding process value outputs, it may be useful to model separate process value output interfaces in the interest of unambiguous semantics. These should then also be derived from this newly specified *OutputElement*.

10.8.15 ArrayProcessValueOut

The SUC *ArrayProcessValueOut* (see Table 10.19) is used for a LEA to make the values of a LEA-internal array available to other LEAs.

Table 10.19: Interface definition of ArrayProcessValueOut

Name	**ArrayProcessValueOut**			
Type	SystemUnitClass			
Description	Generic interface for making available a value of array data type to another LEA			
Hierarchy	MTPDataObjectSUCLib/DataAssembly/OutputElement			
Parent	MTPDataObjectSUCLib/DataAssembly/OutputElement			
RoleClasses				
Version	ModuleTypePackage:Logistics (V0.0.1)			
Alias	**Access**	**Type**	**Description**	**IRDI**
Index	LOL → LEA	DINT	Index Value	
IndexMin	LOL ← LEA	DINT	Low Limit of the Index	
IndexMax	LOL ← LEA	DINT	High Limit of the Index	
IndexCur	LOL ← LEA	DINT	Current Index Value	
V	LOL ← LEA	{VType}	Output Value	
VType	MTP	Type from derived from Base Data Type	Type Definition of the Values	

The *ArrayProcessValueOut* interface definition works nearly the same as the *ArrayView* interface definition (see Section 10.8.4). The only difference is that the *ArrayProcessValueOut* interface definition does not contain an *OSLevel* variable as it is always controlled by another LEA. When considering the use of the *ArrayProcessValueOut* interface the notes on the *ArrayProcessValueIn* (see Section 10.8.13) also shall be taken into account.

10.8.16 LogisticsInteractionExtension

The RC *LogisticsInteractionExtension* (see Table 10.20) extends the *ServiceControl* interface definition (from VDI/VDE/NAMUR 2658-4 [3]) with the variables that are necessary for logistics interactions. If a logistics interaction is provided in the LEA, exactly one *LogisticsInteractionExtension* must be assigned to the *ServiceControl* as SupportedRoleClass, otherwise none.

Table 10.20: Interface definition of LogisticsInteractionExtension

Name	**LogisticsInteractionExtension**			
Type	RoleClass			
Description	Interface definition extending the *ServiceControl* interface for logistice interaction			
Hierarchy	MTPDataObjectSUCLib/DataAssembly/ServiceElement/ParameterElement			
Parent				
RoleClasses	AutomationMLBaseRoleClassLib/AutomationMLBaseRole			
Version	ModuleTypePackage:Logistics (V0.0.1)			
Alias	**Access**	**Type**	**Description**	**IRDI**
LogisticsQuestionID	LOL ← LEA	DINT	Identifier of a currently pending logistics question	
LogisticsQuestionParam	LOL ← LEA	STRING	Question parameter of a currently pending logistics question	
LogisticsAnswerID	LOL → LEA	DINT	Identifier of a currently given answer to a pending question	
LogisticsAnswerTimeout	LOL → LEA	TIME_OF_DAY	Timeout for a LEA to wait for an	

			answer from a LOL 0: timeout function deactivated > 0: timeout in ms	

A logistics interaction follows a similar principle to the service interaction described in VDI/VDE/NAMUR 2658-4 [3]. However, values in the value range of DINT (instead of DWORD) are provided for the IDs of the questions (*LogisticsQuestionID*) and answers (*LogisticsAnswerID*), whereby the value 0 and also negative values can be valid IDs. The value -1 signals that currently no question or no answer is pending. By means of the *LogisticsQuestionParam* (in the same way as by means of the *InteractAddInfo* from the VDI/VDE/NAMUR 2658-4 [3]) an additional information can be added to a request. The variable *LogisticsAnswerTimeout* allows the input of a time period, which the LEA should wait for the answer of a LOL. After this time has elapsed, the LEA can execute an alternative program sequence without the response of the LOL, if necessary. Setting the timeout to the value 0 is interpreted as deactivating the timeout function.

11 Specifications of the Choreography Aspect

For mapping the choreography aspect in the IH of an MTP, the SUCs shown in Figure 11.1 have been developed.

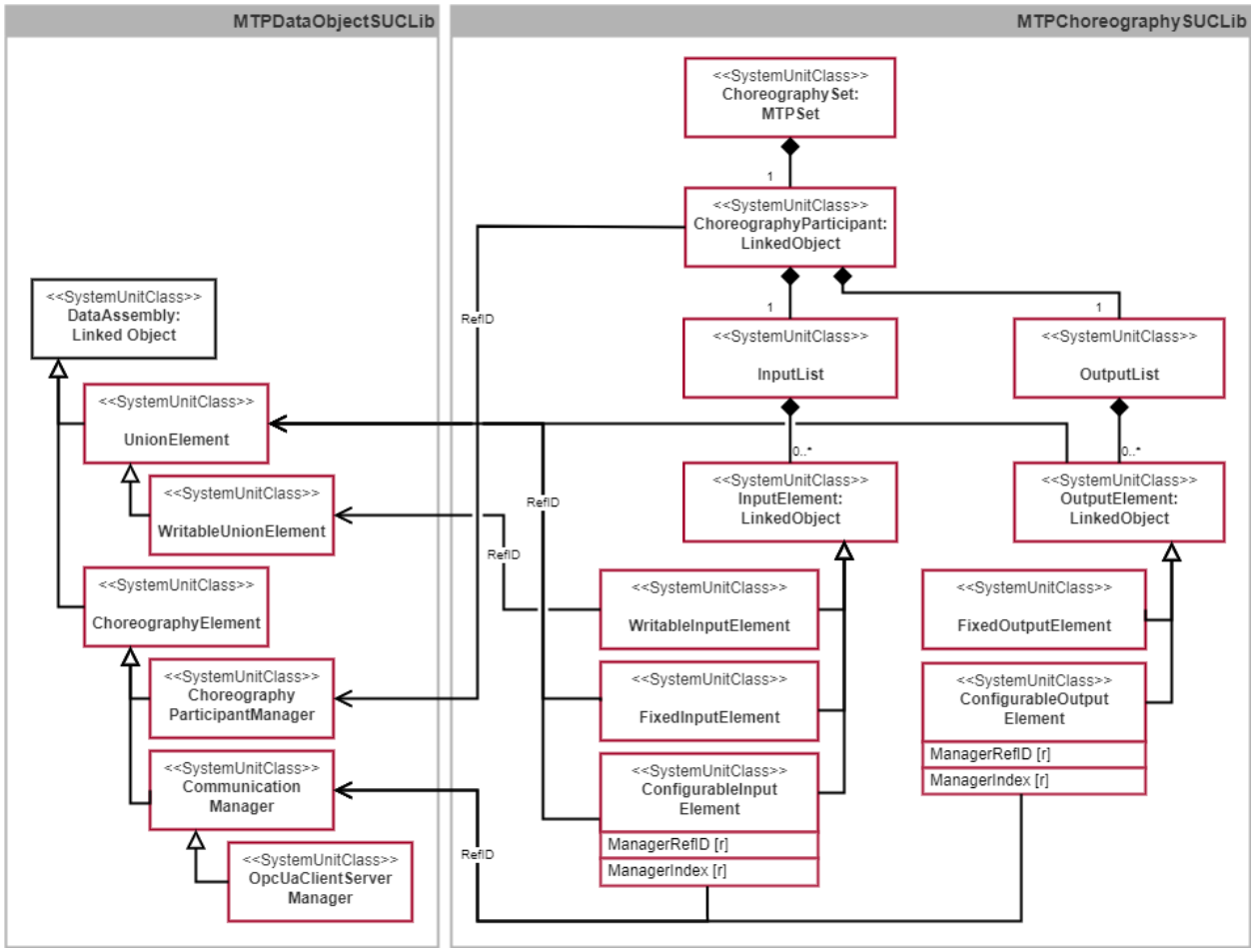

Figure 11.1: SUCs of the choreography aspect

The *ChoreographySet* is derived from the abstract SUC *MTPSet*, which is specified in VDI/VDE/NAMUR 2658-1 [8]. The *ChoreographySet* always contains exactly one *ChoreographyParticipant* derived from the *LinkedObject* specified in VDI/VDE/NAMUR 2658-1 [8]. The *ChoreographyParticipant* organizes all further model definitions necessary for the choreography aspect. These are in the first instance exactly one *InputList* and one *OutputList* containing the incoming and outgoing system variables of the choreography participant. Any number of *InputElements* or *OutputElements* can be included in these lists. These are derived from the *LinkedObject* and each represent an incoming or outgoing system variable. The *InputElements* and *OutputElements* exist in a statically defined (*FixedInputElement* or *FixedOutputElement*) and in a configurable (*ConfigurableInputElement* or *ConfigurableOutputElement*) variant each. In addition, there is a *WritableInputElement* which is passive and can be written to from another LEA.

These model definitions are based on the types of MTPChoreographySUCLib shown in Figure 11.2.

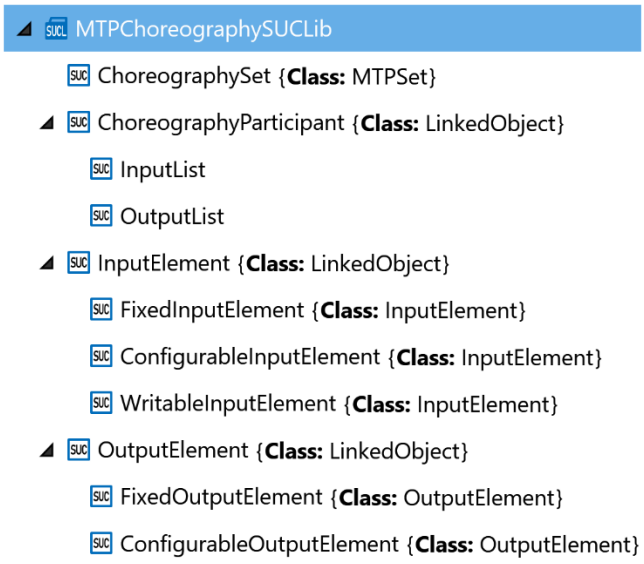

Figure 11.2: Model definitions of the choreography aspect in MTPChoreographySUCLib

In addition, new interface definitions are required for the implementation of choreographies, which are to be specified in the MTPDataObjectSUCLib (see Figure 11.1 and Figure 11.3). The *UnionElement* and the *ChoreographyElement* are directly derived from the *DataAssembly*, which is specified in VDI/VDE/NAMUR 2658-1 [8]. The *ChoreographyParticipantManager* and the *CommunicationManager* are in turn derived from the *ChoreographyElement*. As a specific derivation of the *CommunicationManager*, only the *OpcUaClientServerManager* has been implemented so far; others are conceivable. Especially for *WritableInputLements*, a *WritableUnionElement* is foreseen, which is derived from *UnionElement*. Interface definitions and their associated model definitions are linked by RefID relations.

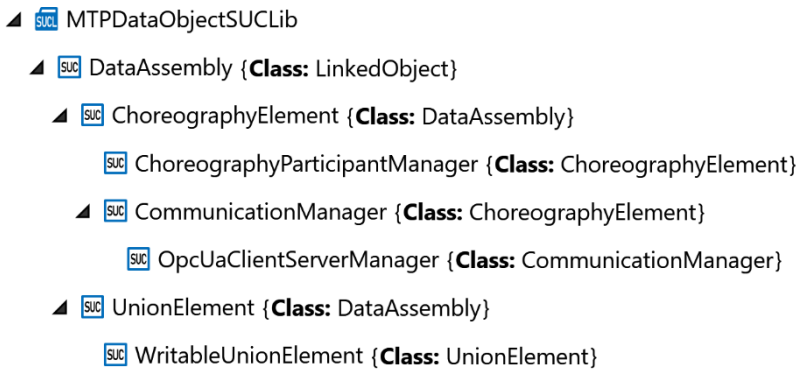

Figure 11.3: Interface definitions of the choreography aspect in MTPDataObjectSUCLib

All model and interface definitions necessary for the choreography aspect are specified in the following Sections 11.1 and 11.2.

11.1 Model Definitions

11.1.1 ChoreographySet

The SUC *ChoreographySet* (see Table 11.1) as a new aspect set of the MTP specification is derived from *MTPSet* and organizes all necessary model definitions to describe an LEA as participant of a choreography.

Table 11.1: Model definition of ChoreographySet

Name	**ChoreographySet**		
Type	SystemUnitClass		
Description	Model definition for choreography aspect set		
Hierarchy	MTPChoreographySUCLib		
Parent	MTPSUCLib/MTPSet		
RoleClasses			
Version	ModuleTypePackage:ChoreographySet (V0.0.1)		
Properties			
Name	**Type**	**Description**	
-	-	-	
Attributes			
Name	**Type**	**Description**	***AttributeType Reference***
-	-	-	-
Comment			
-			

11.1.2 ChoreographyParticipant

The SUC *ChoreographyParticipant* (see Table 11.2) describes a LEA a choreography participant. The *ChoreographyParticipantManager* interface definition is assigned to this model definition via RefID relation, which can be used to pass the choreography-relevant configuration to the participant.

Table 11.2: Model definition of ChoreographyParticipant

Name	**ChoreographyParticipant**		
Type	SystemUnitClass		
Description	Model definition for choreography participant		
Hierarchy	MTPChoreographySUCLib		
Parent	MTPSUCLib/LinkedObject		
RoleClasses			
Version	ModuleTypePackage:ChoreographySet (V0.0.1)		
Properties			
Name	**Type**	**Description**	
-	-	-	
Attributes			
Name	**Type**	**Description**	***AttributeType Reference***
-	-	-	-
Comment			
-			

11.1.3 InputList

The SUC *InputList* (see Table 11.3) organizes all incoming system variables that are relevant for the adaptive logic of a choreography participant. The MTP of a choreography participant always contains exactly one *InputList*.

Table 11.3: Model definition of InputList

Name	**InputList**

Type	SystemUnitClass		
Description	Model definition for the list of input elements of a choreography participant		
Hierarchy	MTPChoreographySUCLib/ChoreographyParticipant		
Parent			
RoleClasses	AutomationMLBaseRoleClassLib/AutomationMLBaseRole		
Version	ModuleTypePackage:ChoreographySet (V0.0.1)		
Properties			
Name	**Type**	**Description**	
-	-	-	
Attributes			
Name	**Type**	**Description**	***AttributeType Reference***
-	-	-	-
Comment			
-			

11.1.4 InputElement

The SUC *InputElement* (see Table 11.4) describes an incoming system variable that is relevant for the adaptive logic of a choreography participant. This can be a statically defined internal process variable of the participant or a configurable process variable the participant obtains from another participant.

Table 11.4: Model definition of InputElement

Name	**InputElement**		
Type	SystemUnitClass		
Description	Model definition for an input element of a choreography participant		
Hierarchy	MTPChoreographySUCLib		
Parent	MTPSUCLib/LinkedObject		
RoleClasses			
Version	ModuleTypePackage:ChoreographySet (V0.0.1)		
Properties			
Name	**Type**	**Description**	
Name	xs:string	Unique Number as Index in the Input List (beginning at 0)	
Attributes			
Name	**Type**	**Description**	***AttributeType Reference***
-	-	-	-
Comment			
-			

11.1.5 FixedInputElement

The SUC *FixedInputElement* (see Table 11.5) is derived from the *InputElement* and describes a statically defined incoming system variable that is provided by the choreography participant itself. A *FixedInputElement* is assigned to the *UnionElement* interface definition via RefID relation.

Table 11.5: Model definition of FixedInputElement

Name	**FixedInputElement**
Type	SystemUnitClass
Description	Model definition for a statically defined input element
Hierarchy	MTPChoreographySUCLib/InputElement

Parent	MTPChoreographySUCLib/InputElement		
RoleClasses			
Version	ModuleTypePackage:ChoreographySet (V0.0.1)		
Properties			
Name	**Type**	**Description**	
-	-	-	
Attributes			
Name	**Type**	**Description**	***AttributeType Reference***
-	-	-	-
Comment			
-			

11.1.6 ConfigurableInputElement

The SUC *ConfigurableInputElement* (see Table 11.6) is derived from the *InputElement* and describes a configurable incoming system variable that the choreography participant obtains from another choreography participant. A *ConfigurableInputElement* is assigned to a *CommunicationManager* interface definition via RefID relation. A *ManagerIndex* is used to refer to a specific communication element (e.g., reader) within the manager. In this way, the communication is configured to exchange the necessary system variable. The interpretation for the *ManagerIndex* depends on the concrete derivation of the *CommunicationManager*. In the case of the *OpcUaClientServer* manager, e.g., this is the index of the used reader. A *FixedInputElement* is assigned to the *UnionElement* interface definition via RefID relation.

Table 11.6: Model definition of ConfigurableInputElement

Name	**ConfigurableInputElement**		
Type	SystemUnitClass		
Description	Model definition for a configurable input element		
Hierarchy	MTPChoreographySUCLib/InputElement		
Parent	MTPChoreographySUCLib/InputElement		
RoleClasses			
Version	ModuleTypePackage:ChoreographySet (V0.0.1)		
Properties			
Name	**Type**	**Description**	
-	-	-	
Attributes			
Name	**Type**	**Description**	***AttributeType Reference***
ManagerRefID	xs:string	Reference identifier to the associated *CommunicationManager* interface	
ManagerIndex	xs:unsignedInt	Index of the incoming configurable communication entity within the communication manager	
Comment			
-			

11.1.7 WritableInputElement

The SUC *WritableInputElement* (see Table 11.7) is derived from the *InputElement* and describes an incoming system variable to which values can be written by another choreography participant. A *WritableInputElement* is assigned to a *WritableUnionElement* interface definition via RefID relation.

Table 11.7: Model definition of WritableInputElement

Name	**WritableInputElement**		
Type	SystemUnitClass		
Description	Model definition for a configurable input element		
Hierarchy	MTPChoreographySUCLib/InputElement		
Parent	MTPChoreographySUCLib/InputElement		
RoleClasses			
Version	ModuleTypePackage:ChoreographySet (V0.0.1)		
Properties			
Name	**Type**	**Description**	
-	-	-	
Attributes			
Name	**Type**	**Description**	***AttributeType Reference***
-	-	-	-
Comment			
-			

11.1.8 OutputList

The SUC *OutputList* (see Table 11.8) organizes all outgoing system variables of the adaptive logic of a choreography participant. The MTP of a choreography participant always contains exactly one *OutputList*.

Table 11.8: Model definition of OutputList

Name	**OutputList**		
Type	SystemUnitClass		
Description	Model definition for the list of output elements of a choreography participant		
Hierarchy	MTPChoreographySUCLib/ChoreographyParticipant		
Parent			
RoleClasses	AutomationMLBaseRoleClassLib/AutomationMLBaseRole		
Version	ModuleTypePackage:ChoreographySet (V0.0.1)		
Properties			
Name	**Type**	**Description**	
-	-	-	
Attributes			
Name	**Type**	**Description**	***AttributeType Reference***
-	-	-	-
Comment			
-			

11.1.9 OutputElement

The SUC *OutputElement* (see Table 11.9) describes an outgoing system variable from the adaptive logic of a choreography participant. This can be a statically defined internal process variable of the participant, or

a configurable process variable the participant sends to another participant. An *OutputElement* is assigned to the *UnionElement* interface definition via RefID relation.

Table 11.9: Model definition of OutputElement

Name	**OutputElement**		
Type	SystemUnitClass		
Description	Model definition for an output element of a choreography participant		
Hierarchy	MTPChoreographySUCLib		
Parent	MTPSUCLib/LinkedObject		
RoleClasses			
Version	ModuleTypePackage:ChoreographySet (V0.0.1)		
Properties			
Name	**Type**	**Description**	
Name	xs:string	Unique Number as Index in the Input List (beginning at 0)	
Attributes			
Name	**Type**	**Description**	***AttributeType Reference***
-	-	-	-
Comment			
-			

11.1.10 FixedOutputElement

The SUC *FixedOutputElement* (see Table 11.10) is derived from the *OutputElement* and describes a statically defined outgoing system variable that is used by the internal program of the choreography participant.

Table 11.10: Model definition of FixedOutputElement

Name	**FixedOutputElement**		
Type	SystemUnitClass		
Description	Model definition for a statically defined output element		
Hierarchy	MTPChoreographySUCLib/OutputElement		
Parent	MTPChoreographySUCLib/OutputElement		
RoleClasses			
Version	ModuleTypePackage:ChoreographySet (V0.0.1)		
Properties			
Name	**Type**	**Description**	
-	-	-	
Attributes			
Name	**Type**	**Description**	***AttributeType Reference***
-	-	-	-
Comment			
-			

11.1.11 ConfigurableOutputElement

The SUC *ConfigurableOutputElement* (see Table 11.11) is derived from the *OutputElement* and describes a configurable outgoing system variable that the choreography participant sends to another choreography participant. A *ConfigurableOutputElement* is associated with a *CommunicationManager* interface definition via RefID relation. A *ManagerIndex* is used to refer to a specific communication element (e.g.,

writer) within the manager. In this way, the communication is configured to exchange the necessary system variable. The interpretation for the *ManagerIndex* depends on the concrete derivation of the *CommunicationManager*. In the case of the *OpcUaClientServerManager*, e.g., this is the index of the used writer.

Table 11.11: Model definition of ConfigurableOutputElement

Name	**ConfigurableOutputElement**		
Type	SystemUnitClass		
Description	Model definition for a configurable output element		
Hierarchy	MTPChoreographySUCLib/OutputElement		
Parent	MTPChoreographySUCLib/OutputElement		
RoleClasses			
Version	ModuleTypePackage:ChoreographySet (V0.0.1)		
Properties			
Name	**Type**	**Description**	
-	-	-	
Attributes			
Name	**Type**	**Description**	***AttributeType Reference***
ManagerRefID	xs:string	Reference identifier to the associated *CommunicationManager* interface	-
ManagerIndex	xs:unsignedInt	Index of the outcoming configurable communication entity within the communication manager	
Comment			
-			

11.2 Interface Definitions

11.2.1 ChoreographyElement

The SUC *ChoreographyElement* (see Table 11.12) is an abstract class derived from the *DataAssembly* interface specified in VDI/VDE/NAMUR 2658-1 [8]. The choreography-relevant interface definitions *ChoreographyParticipantManager* and *CommunicationManager* are derived from the *ChoreographyElement*.

Table 11.12: Interface definition of ChoreographyElement

Name	**ChoreographyElement**			
Type	SystemUnitClass			
Description	Root Interface Class for Choreography-related Interface Definitions			
Hierarchy	MTPDataObjectSUCLib/DataAssembly			
Parent	MTPDataObjectSUCLib/DataAssembly			
RoleClasses				
Version	ModuleTypePackage:ChoreographySet (V0.0.1)			
Alias	**Access**	**Type**	**Description**	**IRDI**
WQC	LOL ← LEA	BYTE	Worst Quality Code	

11.2.2 ChoreographyParticipantManager

The SUC *ChoreographyParticipantManager* (see Table 11.13) is derived from the *ChoreographyElement* and is used to pass the choreography-relevant configuration to a choreography participant. This interface definition is assigned to a *ChoreographyParticipant* in the *ChoreographySet* via RefID relation.

Table 11.13: Interface definition of ChoreographyParticipantManager

Name	**ChoreographyParticipantManager**			
Type	SystemUnitClass			
Description	Configuration Interface for a choreography participant			
Hierarchy	MTPDataObjectSUCLib/DataAssembly/ChoreographyElement			
Parent	MTPDataObjectSUCLib/DataAssembly/ChoreographyElement			
RoleClasses				
Version	ModuleTypePackage:ChoreographySet (V0.0.1)			
Alias	**Access**	**Type**	**Description**	**IRDI**
ViewSel	LOL → LEA	BOOL	Selection to view prepared configuration (false) or active configuration (true)	
ViewCur	LOL ← LEA	BOOL	Currently selected view: false = prepared, true = active	
RestoreDefaultEn	LOL ← LEA	BOOL	Enable flag to restore default configuration	
RestoreDefault	LOL → LEA	BOOL	Restores the default config of all inputs, logics, and outputs	
ExecuteEn	LOL ← LEA	BOOL	Enable flag to execute the Adaptive Logic	
ExecuteOn	LOL → LEA	BOOL	Trigger to apply the current configuration and start the execution	
ExecuteOff	LOL → LEA	BOOL	Trigger to quit the execution, outputs are set to default value	
ExecuteAct	LOL ← LEA	BOOL	Flag which indicates the active execution	
ExecuteErr	LOL ← LEA	BOOL	Flag which indicates min. one processing error	
Input_IndexSel	LOL → LEA	UINT	Index of the desired input configuration to be shown	
Input_IndexMax	LOL ← LEA	UINT	Maximum index for input configuration	
Input_IndexCur	LOL ← LEA	UINT	Index of the currently selected input configuration	
Input_VQC	LOL ← LEA	BYTE	Value quality code of the currently selected input	
Input_DataType	LOL ← LEA	BYTE	Data Type of the currently selected input	
Input_VReal	LOL ← LEA	REAL	Real value of the currently selected input	
Input_VDInt	LOL ← LEA	DINT	Double Integer value of the currently selected input	
Input_VDWord	LOL ← LEA	DWORD	Double Word value of the currently selected input	
Input_VBool	LOL ← LEA	BOOL	Boolean value of the currently selected input	
Input_VString	LOL ← LEA	STRING	String value of the currently selected input	
Logic_IndexSel	LOL → LEA	UINT	Index of the desired logic configuration to be shown	
Logic_IndexMax	LOL ← LEA	UINT	Maximum index for logic configuration	

Logic_IndexCur	LOL ← LEA	UINT	Index of the currently selected logic configuration	
Logic_FuncTypeSel	LOL → LEA	UINT	Function type selector of the currently selected logic element	
Logic_In0_Source	LOL → LEA	SINT	Source of input 0 of the currently selected logic element	
Logic_In0_Index	LOL → LEA	UINT	Index of input 0 of the currently selected logic element	
Logic_In0_Const_VQC	LOL → LEA	BYTE	Constant value quality code of input 0 of the currently selected logic element	
Logic_In0_Const_DataTypeSel	LOL → LEA	BYTE	Constant data type of input 0 of the currently selected logic element	
Logic_In0_Const_VReal	LOL → LEA	REAL	Constant Real value of input 0 of the currently selected logic element	
Logic_In0_Const_VDInt	LOL → LEA	DINT	Constant Double Integer value of input 0 of the currently selected logic element	
Logic_In0_Const_VDWord	LOL → LEA	DWORD	Constant Double Word value of input 0 of the currently selected logic element	
Logic_In0_Const_VBool	LOL → LEA	BOOL	Constant Boolean value of input 0 of the currently selected logic element	
Logic_In0_Const_VString	LOL → LEA	STRING	Constant String value of input 0 of the currently selected logic element	
Logic_In1_Source	LOL → LEA	SINT	Source of input 1 of the currently selected logic element	
Logic_In1_Index	LOL → LEA	UINT	Index of input 1 of the currently selected logic element	
Logic_In1_Const_VQC	LOL → LEA	BYTE	Constant value quality code of input 1 of the currently selected logic element	
Logic_In1_Const_DataTypeSel	LOL → LEA	BYTE	Constant data type of input 1 of the currently selected logic element	
Logic_In1_Const_VReal	LOL → LEA	REAL	Constant Real value of input 1 of the currently selected logic element	
Logic_In1_Const_VDInt	LOL → LEA	DINT	Constant Double Integer value of input 1 of the currently selected logic element	
Logic_In1_Const_VDWord	LOL → LEA	DWORD	Constant Double Word value of input 1 of the currently selected logic element	
Logic_In1_Const_VBool	LOL → LEA	BOOL	Constant Boolean value of input 1 of the currently selected logic element	
Logic_In1_Const_VString	LOL → LEA	STRING	Constant String value of input 1 of the currently selected logic element	
Logic_In2_Source	LOL → LEA	SINT	Source of input 2 of the currently selected logic element	
Logic_In2_Index	LOL → LEA	UINT	Index of input 2 of the currently selected logic element	
Logic_In2_Const_VQC	LOL → LEA	BYTE	Constant value quality code of input 2 of the currently selected logic element	
Logic_In2_Const_DataTypeSel	LOL → LEA	BYTE	Constant data type of input 2 of the currently selected logic element	
Logic_In2_Const_VReal	LOL → LEA	REAL	Constant Real value of input 2 of the currently selected logic element	
Logic_In2_Const_	LOL → LEA	DINT	Constant Double Integer value of input 2	

VDInt			of the currently selected logic element	
Logic_In2_Const_ VDWord	LOL → LEA	DWORD	Constant Double Word value of input 2 of the currently selected logic element	
Logic_In2_Const_ VBool	LOL → LEA	BOOL	Constant Boolean value of input 2 of the currently selected logic element	
Logic_In2_Const_ VString	LOL → LEA	STRING	Constant String value of input 2 of the currently selected logic element	
Logic_In3_Source	LOL → LEA	SINT	Source of input 3 of the currently selected logic element	
Logic_In3_Index	LOL → LEA	UINT	Index of input 3 of the currently selected logic element	
Logic_In3_Const_ VQC	LOL → LEA	BYTE	Constant value quality code of input 3 of the currently selected logic element	
Logic_In3_Const_ DataTypeSel	LOL → LEA	BYTE	Constant data type of input 3 of the currently selected logic element	
Logic_In3_Const_ VReal	LOL → LEA	REAL	Constant Real value of input 3 of the currently selected logic element	
Logic_In3_Const_ VDInt	LOL → LEA	DINT	Constant Double Integer value of input 3 of the currently selected logic element	
Logic_In3_Const_ VDWord	LOL → LEA	DWORD	Constant Double Word value of input 3 of the currently selected logic element	
Logic_In3_Const_ VBool	LOL → LEA	BOOL	Constant Boolean value of input 3 of the currently selected logic element	
Logic_In3_Const_ VString	LOL → LEA	STRING	Constant String value of input 3 of the currently selected logic element	
Logic_Out_VQC	LOL ← LEA	BYTE	Constant value quality code of output of the currently selected logic element	
Logic_Out_ DataType	LOL ← LEA	BYTE	Constant data type of output of the currently selected logic element	
Logic_Out_VReal	LOL ← LEA	REAL	Constant Real value of output of the currently selected logic element	
Logic_Out_VDInt	LOL ← LEA	DINT	Constant Double Integer value of output of the currently selected logic element	
Logic_Out_ VDWord	LOL ← LEA	DWORD	Constant Double Word value of output of the currently selected logic element	
Logic_Out_VBool	LOL ← LEA	BOOL	Constant Boolean value of output of the currently selected logic element	
Logic_Out_VString	LOL ← LEA	STRING	Constant String value of output of the currently selected logic element	
Logic_Ret	LOL ← LEA	UINT	Return value of the currently selected logic element	
Output_IndexSel	LOL → LEA	UINT	Index of the desired output configuration to be shown	
Output_IndexMax	LOL ← LEA	UINT	Maximum index for output configuration	
Output_IndexCur	LOL ← LEA	UINT	Index of the currently selected output configuration	
Output_Source	LOL → LEA	SINT	Source of the currently selected output	
Output_Index	LOL → LEA	UINT	Index of the currently selected output	
Output_DataType	LOL → LEA	BYTE	Data type of the currently selected output	
Output_Const_ VQC	LOL → LEA	BYTE	Constant value quality code of the currently selected output	

Output_Const_ DataTypeSel	LOL → LEA	BYTE	Constant data type of the currently selected output	
Output_Const_ VReal	LOL → LEA	REAL	Constant Real value of the currently selected output	
Output_Const_ VDInt	LOL → LEA	DINT	Constant Double Integer value of the currently selected output	
Output_Const_ VDWord	LOL → LEA	DWORD	Constant Double Word value of the currently selected output	
Output_Const_ VBool	LOL → LEA	BOOL	Constant Boolean value of the currently selected output	
Output_Const_ VString	LOL → LEA	STRING	Constant String value of the currently selected output	
Output_Value_ VQC	LOL → LEA	BYTE	Value quality code of the currently selected output	
Output_Value_ DataTypeSel	LOL → LEA	BYTE	Data type of the currently selected output	
Output_Value_ VReal	LOL → LEA	REAL	Real value of the currently selected output	
Output_Value_ VDInt	LOL → LEA	DINT	Double Integer value of the currently selected output	
Output_Value_ VDWord	LOL → LEA	DWORD	Double Word value of the currently selected output	
Output_Value_ VBool	LOL → LEA	BOOL	Boolean value of the currently selected output	
Output_Value_ VString	LOL → LEA	STRING	String value of the currently selected output	
Output_Ret	LOL ← LEA	UINT	Return value of the currently selected output	

11.2.3 CommunicationManager

The SUC *CommunicationManager* (see Table 11.14) is an abstract class derived from the *ChoreographyElement*. It is to be understood as a generic interface definition for the communication between different Choreography participants. In order to use the *CommunicationManager* interface definition, a concrete manager for a specific communication technology must be derived from it. So far, only the *OpcUAClientServerManager* has been implemented for communication via OPC UA Client/Server; others will be developed in the future.

Table 11.14: Interface definition of CommunicationManager

Name	**CommunicationManager**			
Type	SystemUnitClass			
Description	Abstract interface definition for the communication between different choreography participants			
Hierarchy	MTPDataObjectSUCLib/DataAssembly/ChoreographyElement			
Parent	MTPDataObjectSUCLib/DataAssembly/ChoreographyElement			
RoleClasses				
Version	ModuleTypePackage:ChoreographySet (V0.0.1)			
Alias	**Access**	**Type**	**Description**	**IRDI**
-	-	-	-	-

11.2.4 OpcUaClientServerManager

The SUC *OpcUaClientServerManager* (see Table 11.15) is derived from the abstract *CommunicationManager*. It is used to configure the OPC UA Client/Server communication between different choreography participants and the necessary system variables to be exchanged. The manager is assigned to the model definition of *ConfigurableInputElements* and *ConfigurableOutputElements* in the *ChoreographySet* via RefID relation. In addition, a *ManagerIndex* is specified by each of the model definitions, which refers to a concrete communication element within the manager. In the case of the *OpcUaClientServerManager*, these communication elements are the OPC UA readers and writers managed by the manager, which are referenced by means of their index. The readers are each assigned to a *ConfigurableInputElement* and the writers are each assigned to a *ConfigurableOutputElement*.

Table 11.15: Interface definition of OpcUaClientServerManager

Name	**OpcUaClientServerManager**			
Type	SystemUnitClass			
Description	Interface for managing the OPC UA connections, readers und writers of a choreography participant			
Hierarchy	MTPDataObjectSUCLib/DataAssembly/ChoreographyElement/CommunicationManager			
Parent	MTPDataObjectSUCLib/DataAssembly/ChoreographyElement/CommunicationManager			
RoleClasses				
Version	ModuleTypePackage:ChoreographySet (V0.0.1)			
Alias	**Access**	**Type**	**Description**	**IRDI**
ConnectionViewSel	LOL → LEA	BOOL	Selection to view prepared configuration (false) or active configuration (true)	
ConnectionViewCur	LOL ← LEA	BOOL	Currently selected view: false = prepared, true = active	
ConnectionIndexSel	LOL → LEA	BYTE	Index of the desired connection configuration to be shown	
ConnectionIndexMax	LOL ← LEA	BYTE	Maximum index for connection configuration	
ConnectionIndexCur	LOL ← LEA	BYTE	Index of the currently selected connection configuration	
ConnectionCountActive	LOL ← LEA	BYTE	Number of active connections	
ConnectionCountInactive	LOL ← LEA	BYTE	Number of inactive but configured connections	
ConnectionCountError	LOL ← LEA	BYTE	Number of failed connections	
Connection_ RestoreDefaultEn	LOL ← LEA	BOOL	Enable flag to restore default configuration of the currently selected connection	
Connection_ RestoreDefault	LOL → LEA	BOOL	Restore Default configuration of the currently selected connection	
Connection_ConnectEn	LOL ← LEA	BOOL	Connect the currently selected connection	
Connection_Connect	LOL → LEA	BOOL	Apply the configuration and establish the currently selected connection	
Connection_	LOL ← LEA	BOOL	Indication whether the currently	

ConnectAct			selected connection is established	
Connection_ ConnectErr	LOL ← LEA	BOOL	Indication whether the currently selected connection has an error	
Connection_ DisconnectEn	LOL ← LEA	BOOL	Enable flag to disconnect the currently selected connection	
Connection_Disconnect	LOL → LEA	BOOL	Disconnect the currently selected connection	
Connection_Reset	LOL → LEA	BOOL	Reset the currently selected connection	
Connection_Active	LOL → LEA	BOOL	Indicates that the selected connection is activated to be used	
Connection_ServerUrl	LOL → LEA	STRING	Server URL for the connection	
Connection_ NamespaceUriCount	LOL → LEA	BYTE	Number of namespace URIs	
Connection_ NamespaceUri_1	LOL → LEA	STRING	Namespace URI 1	
Connection_ NamespaceUri_2	LOL → LEA	STRING	Namespace URI 2	
Connection_ NamespaceUri_3	LOL → LEA	STRING	Namespace URI 3	
Connection_ NamespaceUri_4	LOL → LEA	STRING	Namespace URI 4	
Connection_ NamespaceUri_5	LOL → LEA	STRING	Namespace URI 5	
Connection_SessionInfo_ SessionName	LOL → LEA	STRING	Name of the session assigned by the client (when empty, then generated by the server)	
Connection_SessionInfo_ ApplicationName	LOL → LEA	STRING	Readable name of the OPC UA client application	
Connection_SessionInfo_ SecurityMsgMode	LOL → LEA	UDINT	ENUM UASecurityMsgMode	
Connection_SessionInfo_ SecurityPolicy	LOL → LEA	UDINT	ENUM UASecurityPolicy	
Connection_SessionInfo_ ServerUri	LOL → LEA	STRING	Defines the URI of the server, coded in ASCII	
Connection_SessionInfo_ CheckServerCertificate	LOL → LEA	BOOL	Flag indicating if the server certificate should be checked	
Connection_SessionInfo_ TransportProfile	LOL → LEA	UDINT	ENUM UATransportProfile	
Connection_SessionInfo_ UserIdentityTokenType	LOL → LEA	UDINT	ENUM UAUserIdentityTokenType	
Connection_SessionInfo_ UserTokenParam1	LOL → LEA	STRING	Meaning according to UserIdentityTokenType, e.g., username	
Connection_SessionInfo_ UserTokenParam2	LOL → LEA	STRING	Meaning according to UserIdentityTokenType, e.g., password	
Connection_SessionInfo_ CertificateID	LOL → LEA	UDINT	Certificate identifier	
Connection_SessionInfo_ SessionTimeout	LOL → LEA	TIME	Timeout for the session in case of connection loss	
Conn_SessionInfo_ MonitorConnection	LOL → LEA	TIME	Interval time to check the connection	

Connection_SessionInfo_ LocaleID_1	LOL → LEA	STRING	Optional language and regional identifier acc. to RFC 3066. 0 = no or unknown LocaleID.	
Connection_SessionInfo_ LocaleID_2	LOL → LEA	STRING	Optional language and regional identifier acc. to RFC 3066. 0 = no or unknown LocaleID.	
Connection_SessionInfo_ LocaleID_3	LOL → LEA	STRING	Optional language and regional identifier acc. to RFC 3066. 0 = no or unknown LocaleID.	
Connection_SessionInfo_ LocaleID_4	LOL → LEA	STRING	Optional language and regional identifier acc. to RFC 3066. 0 = no or unknown LocaleID.	
Connection_SessionInfo_ LocaleID_5	LOL → LEA	STRING	Optional language and regional identifier acc. to RFC 3066. 0 = no or unknown LocaleID.	
Connection_Status	LOL ← LEA	DWORD	Status of current connection	
ReaderViewSel	LOL → LEA	BOOL	Selection to view prepared configuration (false) or active configuration (true)	
ReaderViewCur	LOL ← LEA	BOOL	Currently selected view: false = prepared, true = active	
ReaderCountInUse	LOL ← LEA	UINT	Number of readers in use	
ReaderCountError	LOL ← LEA	UINT	Number of readers with failures	
ReaderIndexSel	LOL → LEA	UINT	Index of the desired reader configuration to be shown	
ReaderIndexMax	LOL ← LEA	UINT	Maximum index for reader configuration	
ReaderIndexCur	LOL ← LEA	UINT	Index of the currently selected reader configuration	
Reader_ RestoreDefaultEn	LOL ← LEA	BOOL	Enable Flag to restore the default configuration of the currently selected reader	
Reader_RestoreDefault	LOL → LEA	BOOL	Restore the default configuration of the currently selected reader	
Reader_Reset	LOL → LEA	BOOL	Reset the reader	
Reader_ConnectionIndex	LOL → LEA	INT	Connection index the currently selected reader should use	
Reader_InputIndex	LOL ← LEA	UINT	Indicates the index of the Input List the reader refers to	
Reader_DataTypeSel	LOL → LEA	BYTE	Data type of the currently selected reader	
Reader_Timeout	LOL → LEA	TIME	Timeout for the used OPC UA operations	
Reader_MaxTryCount	LOL → LEA	BYTE	Number of tries for an OPC UA operation until the Reader transitions into the error state	
Reader_CycleSel	LOL → LEA	TIME	Target cycle for the read operation	
Reader_CycleCur	LOL ← LEA	TIME	Actual read cycle	
Reader_Error	LOL ← LEA	BOOL	Actual read cycle	
Reader_Status	LOL ← LEA	DWORD	Status of the Reader (e.g., status codes of OPC UA operations in case of	

			an error)	
Reader_Value_ NamespaceIndex	LOL → LEA	UINT	Namespace index of the value of the currently selected reader	
Reader_Value_Identifier	LOL → LEA	STRING	Identifier of the value of the currently selected reader	
Reader_Value_ IdentifierType	LOL → LEA	UDINT	Identifier type of the value of the currently selected reader	
Reader_QC_ NamespaceIndex	LOL → LEA	UINT	Namespace index of the quality code of the currently selected reader	
Reader_QC_Identifier	LOL → LEA	STRING	Identifier of the quality code of the currently selected reader	
Reader_QC_ IdentifierType	LOL → LEA	UDINT	Identifier type of the quality code of the currently selected reader	
WriterViewSel	LOL → LEA	BOOL	Selection to view prepared configuration (false) or active configuration (true)	
WriterViewCur	LOL ← LEA	BOOL	Currently selected view: false = prepared, true = active	
WriterCountInUse	LOL ← LEA	UINT	Number of writers in use	
WriterCountError	LOL ← LEA	UINT	Number of writers with failures	
WriterIndexSel	LOL → LEA	UINT	Index of the desired writer configuration to be shown	
WriterIndexMax	LOL ← LEA	UINT	Maximum index for writer configuration	
WriterIndexCur	LOL ← LEA	UINT	Index of the currently selected writer configuration	
Writer_RestoreDefaultEn	LOL ← LEA	BOOL	Enable Flag to restore the default configuration of the currently selected writer	
Writer_RestoreDefault	LOL → LEA	BOOL	Restore the default configuration of the currently selected writer	
Writer_Reset	LOL → LEA	BOOL	Reset the writer	
Writer_ConnectionIndex	LOL → LEA	INT	Connection index the currently selected writer should use	
Writer_OutputIndex	LOL ← LEA	UINT	Indicates the index of the Output List the writer refers to	
Writer_DataTypeSel	LOL → LEA	BYTE	Data type of the currently selected writer	
Writer_Timeout	LOL → LEA	TIME	Timeout for the used OPC UA operations	
Writer_MaxTryCount	LOL → LEA	BYTE	Number of tries for an OPC UA operation until the writer transitions into the error state	
Writer_CycleSel	LOL → LEA	TIME	Target cycle for the write operation	
Writer_CycleCur	LOL ← LEA	TIME	Actual write cycle	
Writer_Error	LOL ← LEA	BOOL	True, if the writer is in the error state	
Writer_Status	LOL ← LEA	DWORD	Status of the Writer (e.g., status codes of OPC UA operations in case of an error)	
Writer_Value_ NamespaceIndex	LOL → LEA	UINT	Namespace index of the value of the currently selected writer	

Writer_Value_Identifier	LOL → LEA	STRING	Identifier of the value of the currently selected writer	
Writer_Value_ IdentifierType	LOL → LEA	UDINT	Identifier type of the value of the currently selected writer	

11.2.5 UnionElement

The SUC *UnionElement* (see Table 11.16) is used to display the value of an *InputElement* or *OutputElement*. Accordingly, the *UnionElement* is assigned to these model definitions by means of RefID relation.

Table 11.16: Interface definition of UnionElement

Name	**UnionElement**			
Type	SystemUnitClass			
Description	Interface for displaying a value with datatype defined at runtime			
Hierarchy	MTPDataObjectSUCLib/DataAssembly			
Parent	MTPDataObjectSUCLib/DataAssembly			
RoleClasses				
Version	ModuleTypePackage:ChoreographySet (V0.0.1)			
Alias	**Access**	**Type**	**Description**	**IRDI**
VQC	LOL ← LEA	BYTE	Quality Code of the value	
DataType	LOL ← LEA	BYTE	Identifier of selected data type (0 : None, 1: VReal, 2: VDInt, 3: VDWord, 4: VBool, 5: VString)	
VReal	LOL ← LEA	REAL	Real Value (Type: 1)	
VDInt	LOL ← LEA	DINT	Double Integer Value (Type: 2)	
VDWord	LOL ← LEA	DWORD	Double Word Value (Type: 3)	
VBool	LOL ← LEA	BOOL	Boolean Value (Type: 4)	
VString	LOL ← LEA	STRING	String Value (Type: 5)	

VReal, *VDInt*, *VDWord*, *VBool* and *VString* variables are used to display the desired value. *DataType* variable specifies which data type is activated at the moment and as a consequence which of the previously mentioned variables shall be interpreted. Thus, the *UnionElement* can only display one value of a defined data type at a time. *VQC* provides information about the quality code of the displayed value.

11.2.6 WritableUnionElement

The SUC Writable*UnionElement* (see Table 11.17) is derived from *UnionElement* and used to write a value to a *WritableInputElement*. Accordingly, the *WritableUnionElement*is assigned to this model definition by means of RefID relation.

Table 11.17: Interface definition of WritableUnionElement

Name	**WritableUnionElement**
Type	SystemUnitClass
Description	Interface for writing a value with datatype defined at runtime
Hierarchy	MTPDataObjectSUCLib/DataAssembly/UnionElement
Parent	MTPDataObjectSUCLib/DataAssembly/UnionElement
RoleClasses	
Version	ModuleTypePackage:ChoreographySet (V0.0.1)

Alias	Access	Type	Description	IRDI
VQC	LOL → LEA	BYTE	Quality Code of the value	
DataType	LOL → LEA	BYTE	Identifier of selected data type (0 : None, 1: VReal, 2: VDInt, 3: VDWord, 4: VBool, 5: VString)	
VReal	LOL → LEA	REAL	Real Value (Type: 1)	
VDInt	LOL → LEA	DINT	Double Integer Value (Type: 2)	
VDWord	LOL → LEA	DWORD	Double Word Value (Type: 3)	
VBool	LOL → LEA	BOOL	Boolean Value (Type: 4)	
VString	LOL → LEA	STRING	String Value (Type: 5)	

VReal, *VDInt*, *VDWord*, *VBool* and *VString* variables are used to enter the desired value. *DataType* variable specifies which data type is activated at the moment and as a consequence which of the previously mentioned variables shall be used in the LEA program. Thus, the *WritableUnionElement* only accepts one value of a defined data type at a time. *VQC* can be used to enter information about the quality code of the entered value.

12 Specifications of the Transport Aspect

To model the transportation aspect in the IH of an MTP, the SUCs shown in Figure 12.1 have been developed.

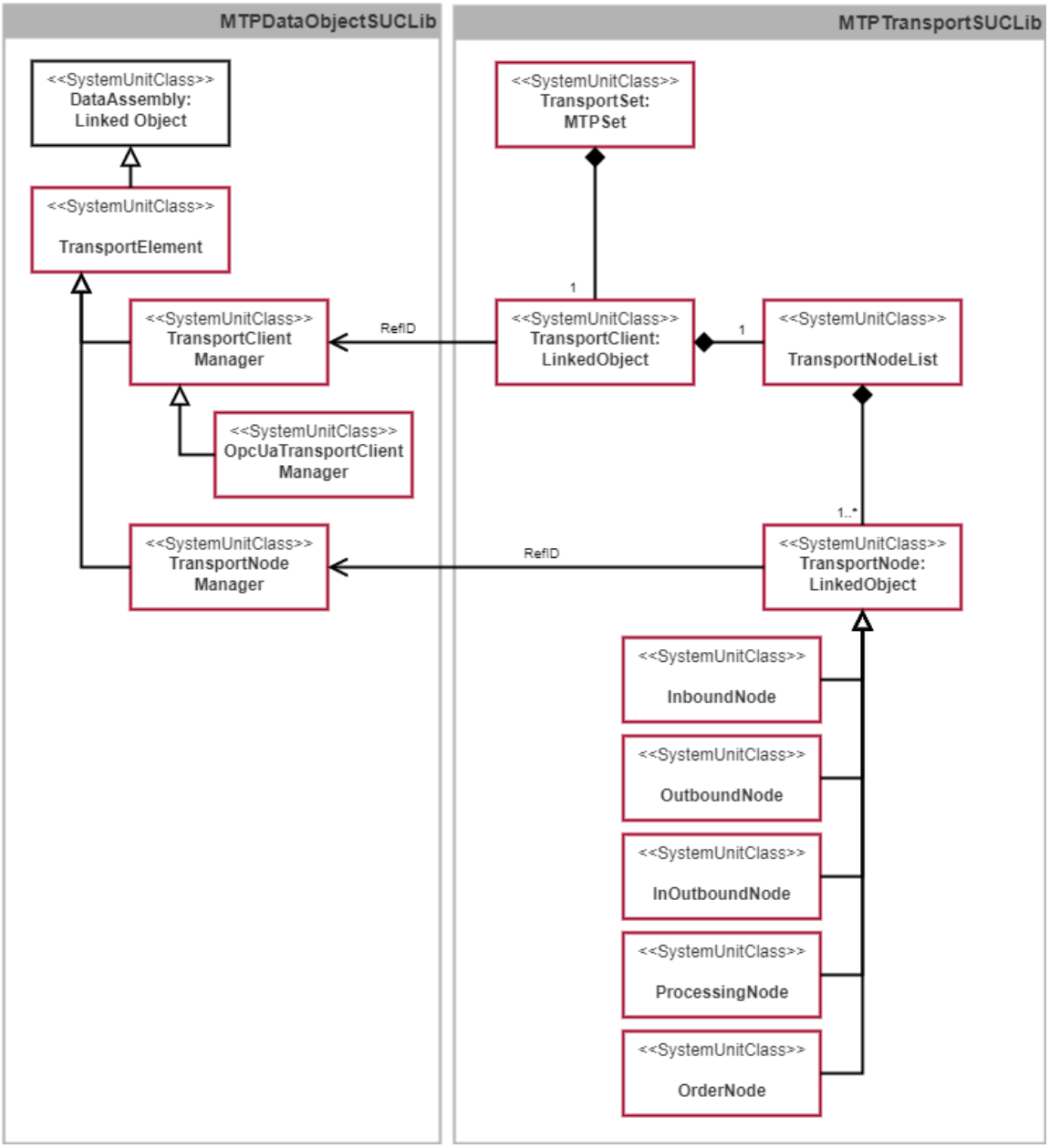

Figure 12.1: SUCs of the transport aspect

The *TransportSet* is derived from the abstract SUC *MTPSet*, which is specified in VDI/VDE/NAMUR 2658-1 [8]. The *TransportSet* always contains exactly one *TransportClient*, derived from the *LinkedObject* specified in VDI/VDE/NAMUR 2658-1 [8]. The *TransportClient* organizes all further model definitions necessary for the transport aspect. This is in first instance exactly one list *TransportNodeList*, which contains all transport nodes available in the LEA. Subordinate to this list any number of *TransportNode* model definitions is listed. These are derived from the *LinkedObject* and represent one available transport node each. There are model definitions for 5 different specific transport nodes derived from the abstract *TransportNode - InboundNode*, *OutboundNode*, *InOutboundNode*, *ProcessingNode* and *OrderNode*.

These model definitions are based on the types of MTPTransportSUCLib shown in Figure 12.2.

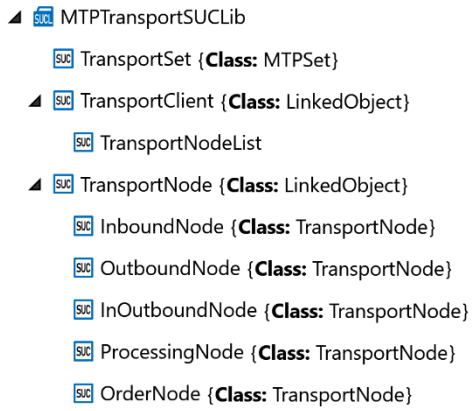

Figure 12.2: Model definitions of the transport aspect in MTPTransportSUCLib

In addition, new interface definitions are required for the implementation of flexible transport processes, which must be specified in the MTPDataObjectSUCLib (see Figure 12.1 and Figure 12.3). The *TransportElement* is derived directly from the *DataAssembly* specified in VDI/VDE/NAMUR 2658-1 [8]. The *TransportClientManager* and the *TransportNodeManager* are in turn derived from the *TransportElement*. The interface definitions and their associated model definitions are linked via RefID relations. The *TransportClientManager* is a generic client that is used to establish a communication connection between a LEA and a transport system. Different communication technologies can be used for this purpose. So far, only one variant based on OPC UA Client/Server has been implemented. The associated interface definition is the *OpcUaCSTransportClientManager*, which is derived from the generic *TransportClientManager*.

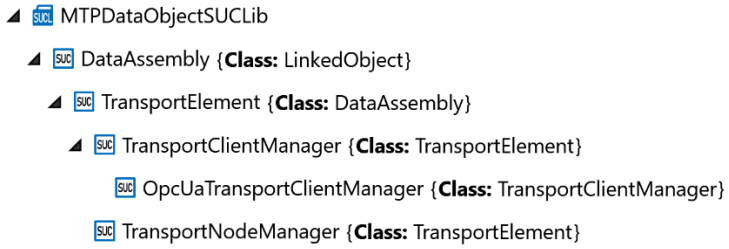

Figure 12.3: Interface definitions of the transport aspect in MTPDataObjectSUCLib

All model and interface definitions necessary for the transport aspect are specified in the following Sections 12.1 and 12.2.

12.1 Model Definitions

12.1.1 TransportSet

The SUC *TransportSet* (see Table 12.1) as a new aspect set of the MTP specification contains all necessary model definitions to enable a LEA to interact with a flexible transport system.

Table 12.1: Model definition of TransportSet

Name	**TransportSet**
Type	SystemUnitClass

Description	Model definition for transport aspect set		
Hierarchy	MTPTransportSUCLib		
Parent	MTPSUCLib/MTPSet		
RoleClasses			
Version	ModuleTypePackage:TransportSet (V0.0.1)		
Properties			
Name	**Type**	**Description**	
-	-	-	
Attributes			
Name	**Type**	**Description**	***AttributeType Reference***
-	-	-	-
Comment			
-			

12.1.2 TransportClient

The SUC *TransportClient* (see Table 12.2) describes the client of a LEA for the communication connection to a flexible transport system. RefID relation is used to assign the *TransportClientManager* interface definition to this model definition, which can be used to configure the necessary communication connection to the transport system.

Table 12.2: Model definition of TransportClient

Name	**TransportClient**		
Type	SystemUnitClass		
Description	Model definition for the client communicating transport-relevant data		
Hierarchy	MTPTransportSUCLib		
Parent	MTPSUCLib/LinkedObject		
RoleClasses			
Version	ModuleTypePackage:TransportSet (V0.0.1)		
Properties			
Name	**Type**	**Description**	
-	-	-	
Attributes			
Name	**Type**	**Description**	***AttributeType Reference***
-	-	-	-
Comment			
-			

12.1.3 TransportNodeList

The SUC *TransportNodeList* (see Table 12.3) organizes all transport nodes available in a LEA within one list. The MTP of a LEA that is capable of interacting with flexible transport systems always contains exactly one *TransportNodeList*.

Table 12.3: Model definition of TransportNodeList

Name	**TransportNodeList**
Type	SystemUnitClass
Description	Model definition for the list of transport nodes of a transport-enabled Logistics Equipment Assembly

Hierarchy	MTPTransportSUCLib/TransportClient		
Parent			
RoleClasses	AutomationMLBaseRoleClassLib		
Version	ModuleTypePackage:TransportSet (V0.0.1)		
Properties			
Name	**Type**	**Description**	
-	-	-	
Attributes			
Name	**Type**	**Description**	***AttributeType Reference***
-	-	-	-
Comment			
-			

12.1.4 TransportNode

The SUC *TransportNode* (see Table 12.4) is an abstract model definition to describe a transport node available in a LEA. As of today, 5 concrete types of transport nodes are derived from this model definition – *InboundNode*, *OutboundNode*, *InOutBoundNode*, *ProcessingNode* and *OrderNode*. A *TransportNode* is assigned to the *TransportNodeManager* interface definition via RefID relation, which enables the assignment of transport orders to the transport node.

Table 12.4: Model definition of TransportNode

Name	**TransportNode**		
Type	SystemUnitClass		
Description	Model definition for a transport node of a transport-enabled Logistics Equipment Assembly		
Hierarchy	MTPTransportSUCLib		
Parent	MTPSUCLib/LinkedObject		
RoleClasses			
Version	ModuleTypePackage:TransportSet (V0.0.1)		
Properties			
Name	**Type**	**Description**	
-	-	-	
Attributes			
Name	**Type**	**Description**	***AttributeType Reference***
-	-	-	-
Comment			
-			

12.1.5 InboundNode

The SUC *InboundNode* (see Table 12.5) is derived from *TransportNode* and describes a transport node for transferring an object from a transport system to the LEA.

Table 12.5: Model definition of InboundNode

Name	**InboundNode**
Type	SystemUnitClass
Description	Model definition for a transport node transferring objects from a transport system to the Logistics Equipment Assembly

Hierarchy	MTPTransportSUCLib/TransportNode		
Parent	MTPTransportSUCLib/TransportNode		
RoleClasses			
Version	ModuleTypePackage:TransportSet (V0.0.1)		
Properties			
Name	**Type**	**Description**	
-	-	-	
Attributes			
Name	**Type**	**Description**	***AttributeType Reference***
-	-	-	-
Comment			
-			

12.1.6 OutboundNode

The SUC *OutboundNode* (see Table 12.6) is derived from *TransportNode* and describes a transport node for transferring an object from the LEA to a transport system.

Table 12.6: Model definition of OutboundNode

Name	**OutboundNode**		
Type	SystemUnitClass		
Description	Model definition for a transport node transferring objects from the Logistics Equipment Assembly to a transport system		
Hierarchy	MTPTransportSUCLib/TransportNode		
Parent	MTPTransportSUCLib/TransportNode		
RoleClasses			
Version	ModuleTypePackage:TransportSet (V0.0.1)		
Properties			
Type	**Type**	**Type**	
-	-	-	
Attributes			
Type	**Type**	**Type**	***AttributeType Reference***
-	-	-	-
Comment			
-			

12.1.7 InOutboundNode

The SUC *InOutboundNode* (see Table 12.7) is derived from *TransportNode* and describes a transport node for transferring objects between the LEA and a transport system in both directions.

Table 12.7: Model definition of InOutboundNode

Name	**InOutboundNode**
Type	SystemUnitClass
Description	Model definition for a transport node transferring objects between the Logistics Equipment Assembly and a transport system in both directions
Hierarchy	MTPTransportSUCLib/TransportNode
Parent	MTPTransportSUCLib/TransportNode
RoleClasses	

Version	ModuleTypePackage:TransportSet (V0.0.1)		
Properties			
Type	**Type**	**Type**	
-	-	-	
Attributes			
Type	**Type**	**Type**	***AttributeType Reference***
-	-	-	-
Comment			
-			

12.1.8 ProcessingNode

The SUC *ProcessingNode* (see Table 12.8) is derived from *TransportNode* and describes a transport node for processing an object by the LEA without taking this object from the transport system.

Table 12.8: Model definition of ProcessingNode

Name	**ProcessingNode**		
Type	SystemUnitClass		
Description	Model definition for a transport node processing an object without transferring the object from the transport system to the Logistics Equipment Assembly		
Hierarchy	MTPTransportSUCLib/TransportNode		
Parent	MTPTransportSUCLib/TransportNode		
RoleClasses			
Version	ModuleTypePackage:TransportSet (V0.0.1)		
Properties			
Type	**Type**	**Description**	
-	-	-	
Attributes			
Type	**Type**	**Description**	***AttributeType Reference***
-	-	-	-
Comment			
-			

12.1.9 OrderNode

The SUC *OrderNode* (see Table 12.9) is derived from *TransportNode* and describes a node for indicating transport demands and initializing corresponding transport processes.

Table 12.9: Model definition of OrderNode

Name	**OrderNode**	
Type	SystemUnitClass	
Description	Model definition for a node to indicate transport demands an initialize corresponding transport processes	
Hierarchy	MTPTransportSUCLib/TransportNode	
Parent	MTPTransportSUCLib/TransportNode	
RoleClasses		
Version	ModuleTypePackage:TransportSet (V0.0.1)	
Properties		
Type	**Type**	**Type**

-	-	-	
Attributes			
Type	**Type**	**Type**	***AttributeType Reference***
-	-	-	-
Comment			
-			

12.2 Interface Definitions

12.2.1 TransportElement

The SUC *TransportElement* (see Table 12.10) is an abstract class that is derived from *DataAssembly*. The transport-relevant interface definitions *TransportClientManager* and *TransportNodeManager* are derived from *TransportElement*. The *TransportElement* interface definition cannot be used independently, but only in the form of one of its derivations.

Table 12.10: Interface definition of TransportElement

Name	**TransportElement**			
Type	SystemUnitClass			
Description	Root interface class for transport-related Interface definitions			
Hierarchy	MTPDataObjectSUCLib/DataAssembly			
Parent	MTPDataObjectSUCLib/DataAssembly			
RoleClasses				
Version	ModuleTypePackage:TransportSet (V0.0.1)			
Alias	**Access**	**Type**	**Description**	**IRDI**
WQC	LOL ← LEA	BYTE	Worst Quality Code	

12.2.2 TransportClientManager

The SUC *TransportClientManager* (see Table 12.11) is derived from the *TransportElement* and is an abstract interface definition for configuring the communication connection between the LEA and a transport system. In order to implement this interface definition, a concrete manager must be derived from it. So far, only the *OpcUaCSTransportClientManager* has been implemented. The *TransportClientManager* and thus also its derivatives are assigned to a *TransportClient* in the *ChoreographySet* via RefID relation.

Table 12.11: Interface definition of TransportClientManager

Name	**TransportClientManager**			
Type	SystemUnitClass			
Description	Abstract interface definition for configuring the communication of the Logistics Equipment Assembly to a transport system			
Hierarchy	MTPDataObjectSUCLib/DataAssembly/TransportElement			
Parent	MTPDataObjectSUCLib/DataAssembly/TransportElement			
RoleClasses				
Version	ModuleTypePackage:TransportSet (V0.0.1)			
Alias	**Access**	**Type**	**Description**	**IRDI**
-	-	-	-	-

12.2.3 OpcUaCSTransportClientManager

The SUC *OpcUaCSTransportClientManager* (see Table 12.12) is derived from the *TransportClientManager* and is used to configure an OPC UA Client/Server communication connection between the LEA and a transport system.

Table 12.12: Interface definition of OpcUaCSTransportClientManager

Name	**OpcUaCSTransportClientManager**			
Type	SystemUnitClass			
Description	Configuration interface for a client communicating transport-relevant data			
Hierarchy	MTPDataObjectSUCLib/DataAssembly/TransportElement/TransportClientManager			
Parent	MTPDataObjectSUCLib/DataAssembly/TransportElement/TransportClientManager			
RoleClasses				
Version	ModuleTypePackage:TransportSet (V0.0.1)			
Alias	**Access**	**Type**	**Description**	**IRDI**
ConfigApplyEn	LOL → LEA	BOOL	Enable flag to apply the prepared configuration	
ConfigApplyExt	LOL ← LEA	BOOL	Apply the prepared configuration	
ConnectEn	LOL → LEA	BOOL	Enable flag to establish connection	
ConnectExt	LOL ← LEA	BOOL	Establish connection	
DisconnectEn	LOL → LEA	BOOL	Enable flag to remove connection	
DisconnectExt	LOL ← LEA	BOOL	Remove connection	
ResetExt	LOL → LEA	BOOL	Reset communication block	
ConnectionAct	LOL ← LEA	BOOL	Flag indicating an established connection	
ConnectionErr	LOL ← LEA	BOOL	Flag indicating a connection error	
ErrorId	LOL ← LEA	DWORD	Identifier of the connection error	
EndpointExt	LOL → LEA	STRING	Defines the server URL to connect with	
NamespaceExt	LOL → LEA	STRING	Defines Namespace to be used	
EndpointReq	LOL ← LEA	STRING	Requested server URL	
NamespaceReq	LOL ← LEA	STRING	Requested namespace	
EndpointCur	LOL ← LEA	STRING	Currently configured server URL	
NamespaceCur	LOL ← LEA	STRING	Currently configured namespace	
LeaStateCur	LOL ← LEA	DWORD	MTP service state of the LEA service	

12.2.4 TransportNodeManager

The SUC *TransportNodeManager* (see Table 12.13) is derived from the *TransportElement* and is used to assign a transport order to a specific transport node. This interface definition is assigned to a *TransportNode* in the *ChoreographySet* via RefID relation.

Table 12.13: Interface definition of TransportNodeManager

Name	**TransportNodeManager**			
Type	SystemUnitClass			
Description	Configuration interface for transport nodes			
Hierarchy	MTPDataObjectSUCLib/DataAssembly/TransportElement			
Parent	MTPDataObjectSUCLib/DataAssembly/TransportElement			
RoleClasses				
Version	ModuleTypePackage:TransportSet (V0.0.1)			
Alias	**Access**	**Type**	**Description**	**IRDI**
ConfigApplyEn	LOL → LEA	BOOL	Enable flag to apply the prepared configuration	

ConfigApplyExt	LOL ← LEA	BOOL	Apply the prepared configuration	
ConnectEn	LOL → LEA	BOOL	Enable flag to establish connection	
ConnectExt	LOL ← LEA	BOOL	Establish connection	
DisconnectEn	LOL → LEA	BOOL	Enable flag to remove connection	
DisconnectExt	LOL ← LEA	BOOL	Remove connection	
ResetExt	LOL → LEA	BOOL	Reset communication block	
ConnectionAct	LOL ← LEA	BOOL	Flag indicating an established connection	
ConnectionErr	LOL ← LEA	BOOL	Flag indicating a connection error	
ErrorId	LOL ← LEA	DWORD	Identifier of the connection error	
ProxyIdExt	LOL → LEA	DINT	Defines related proxy in the transportsystem	
ProxyIdReq	LOL ← LEA	DINT	Requested transport proxy	
ProxyIdCur	LOL ← LEA	DINT	Currently configured transport proxy	

13 References

[1] *NE 171: Application of a modular plant engineering concept in production-related logistics,* NAMUR Working Group WG 4.19 Production-related logistics, Dec. 2020.

[2] S. Cordes, T. Busert, A. Fay, S. Kessler, and A. Schick, “NAMUR-MTP for plug-&-operate in production-oriented logistics: Requirements for modular automation,” *atp magazin*, 01-02, pp. 86–93, 2020.

[3] *VDI/VDE/NAMUR 2658-4: Automation engineering of modular systems in the process industry - Modelling of module services,* VDI/VDE-GMA, Berlin, 2022. [Online]. Available: https://www.vdi.de/2658

[4] *VDI/VDE/NAMUR 2658-2: Automation engineering of modular systems in the process industrie - Modeling of human-machine interfaces,* VDI/VDE-GMA, Berlin, 2019. [Online]. Available: https://www.vdi.de/2658

[5] *VDI/VDE/NAMUR 2658-3: Automation engineering of modular systems in the process industry - Library for data objects,* VDI/VDE-GMA, Berlin, 2020. [Online]. Available: https://www.vdi.de/2658

[6] M. Blumenstein *et al.,* “Design principles for the module and service design in modular logistics facilities,” in *VDI Congress Automation 2021*.

[7] M. Blumenstein *et al.,* “Modular Automation in production logistics - MTP concepts for logistics equipment assemblies,” *atp magazin*, vol. 63, no. 10, pp. 66–75, 2022.

[8] *VDI/VDE/NAMUR 2658-1: Automation engineering of modular systems in the process industry - General concept and interfaces (Draft),* VDI/VDE-GMA, Berlin, 2022. [Online]. Available: https://www.vdi.de/2658

[9] A. Stutz, A. Fay, M. Barth, and M. Maurmaier, “Software Patterns for the Realization of Automation Service Choreographies,” in *IEEE ETFA 2021*.

[10] M. Blumenstein, A. Stutz, A. Fay, M. Barth, and M. Maurmaier, “Coordination of Modular Packaging Lines Using Automation Service Choreographies,” in *IEEE ETFA 2022*.

[11] M. Blumenstein, V. Henkel, A. Fay, A. Stutz, S. Scheuren, and N. Austermann, “Integration of Flexible Transport Systems into Modular Production-Related Logistics Areas,” in *IEEE INDIN 2023*.

[12] A. Stutz, M. Maurmaier, F. Spaethe, and P. Hill, “Introduction, Development and Application of sub-elementary Services for modular Plants,” in *VDI Congress Automation 2020*.

[13] *VDI 5600-1: Manufacturing execution systems (MES),* VDI, Berlin. [Online]. Available: https://www.vdi.de/richtlinien/details/vdi-5600-blatt-1-fertigungsmanagementsysteme-manufacturing-execution-systems-mes

[14] *VDI 5600-7: Manufacturing execution systems (MES) - MES and Industrie 4.0,* VDI, Berlin, Apr. 2021. [Online]. Available: https://www.vdi.de/richtlinien/details/vdi-5600-blatt-7-fertigungsmanagementsysteme-manufacturing-execution-systems-mes-mes-und-industrie-40

[15] M. Blumenstein *et al.,* “Logistics Orchestration Layer - Requirements for the Orchestration of Modular Logistics Systems,” in *VDI Congress Automation 2023*.

[16] M. Blumenstein, A. Stutz, L. Beers, M. Maurmaier, and A. Fay, “Comparative study concerning the orchestration of modular plants in the process industry, the manufacturing industry and production-related logistics,” in *VDI Congress Automation 2022*.

[17] M. Runge, L.-T. Reiche, K.-H. Niemann, and A. Fay, “Universal energy information model for industrial communication,” in *IEEE ETFA 2022*.

[18] L.-T. Reiche and A. Fay, “Concept for extending the Module Type Package with energy management functionalities,” in *IEEE ETFA 2022*.

[19] *VDI/VDE/NAMUR 2658: Automation engineering of modular systems in the process industry,* VDI/VDE-GMA, Berlin. [Online]. Available: https://www.vdi.de/2658

[20] *IEC 62714-1:2018: Engineering data exchange format for use in industrial automation systems engineering - Automation Markup Language - Part 1: Architecture and general requirements,* IEC, Berlin. [Online]. Available: https://www.vde-verlag.de/iec-normen/225580/iec-62714-1-2018.html

[21] *VDI/VDE/NAMUR 2658-5: Automation engineering of modular systems in the process Industry - Runtime and communication aspects (Draft),* VDI/VDE-GMA, Berlin, 2022. [Online]. Available: https://www.vdi.de/2658