Supporting evolution of automated Material Flow Systems as part of CPPS by using coupled meta models

Birgit Vogel-Heuser[1], Marco Konersmann[2], Thomas Aicher[1], Juliane Fischer[1], Felix Ocker[1], Michael Goedicke[2]

[1]Technical University of Munich
Institute of Automation and Information Systems
85748 Garching near Munich, Germany
{vogel-heuser, juliane.fischer, felix.ocker}@tum.de;
aicher@ais.mw.tum.de

[2]University of Duisburg-Essen
paluno – The Ruhr Institute for Software Technolgy
45127 Essen, Germany
{michael.goedicke, marco.konersmann}@paluno.uni-due.de

***Abstract*—The software of automated Material Flow Systems (aMFS) is usually tightly coupled to the system in which it is embedded. Reuse of control software of aMFS is commonly achieved through copying, pasting, and modifying existing control code from a similar system. When a module of an aMFS is updated in production, e.g. to resolve issues, the program code and hardware components are changed. The change is usually documented informally or resides only in the mind of the automation engineer. When new hardware is added to a module, the control software often has to be changed in many different parts, or a new control function has to be developed, which wraps the existing code. This way of handling the development and evolution of aMFSs is error-prone and reduces the software quality. A modular, model-based software architecture for aMFS can reduce the error-proneness and improve the software quality. In this paper we propose a model-based development method for aMFS. It is used to create a modular software architecture that describes composable modules. The models, their meta models, and the program code therein are coupled with each other via mappings and consistency rules, so that the consistency can be validated. Bidirectional transformations between the program code and models can be created. As part of the documentation, changes in the program code and model must be synchronized.**

Keywords— methodologies and tools; automation systems; flexible manufacturing systems; software evolution; CPPS; intralogistics

I. INTRODUCTION

Automated Material Flow Systems (aMFS) are a crucial and standard part of Cyber Physical Production Systems (CPPS). They connect different machines or plant parts providing the transport of material from one to the other. Considering the current (re-)engineering of aMFS, the initial (re-)engineering stage, i.e. drawing up the material flow plan of the system, generally serves as requirements document for other engineering disciplines, e.g. software engineering. The applied electrical/mechanical components of an aMFS as well as the assembly of these components with one another must be identified manually from the material flow plan and transferred into the control software.

After that, reuse of control software is commonly achieved through copying, pasting and modifying existing control code from a similar aMFS. Due to the static structure and high variability of today's control software of aMFS, the modification of existing control code is error-prone and requires a great deal of effort to develop. More specifically, the static structure prevents commonly required adaptions of the control software for a mechanical and electrical component, e.g. adding a further light barrier, or exchanging a mechanical and electrical component for one possessing a higher version number, e.g. a barcode scanner with a Quick Response (QR) scanner. Often, the control software has to be modified in different parts of the monolithic structure or a new and individual control function has to be programmed to wrap the existing control code, i.e. to adapt the interfaces and/or behavior. Both approaches are error-prone and reduce the software quality.

Reusability can be improved by minimizing the static structure of an aMFS and development effort and error-proneness are reduced by setting up a modular software architecture [1, 2]. By using a modular software architecture for each electrical/mechanical component of an aMFS, a (pre)defined software component–i.e. an automated material flow module (aMFM)–exists, which can execute specific tasks and offer the possibility to interact with neighboring modules. This way, the modular structure of electrical/mechanical components in aMFS can be transferred into software engineering in order to improve reusability and software quality and reduce effort and error-proneness.

Considering different methodologies in software development of automated Production Systems (aPS), model-driven engineering (MDE) is frequently recommended in research and industry due to advantages regarding engineering and commissioning time [3, 4, 5]. Based on an MDE approach, domain-specific meta models, e.g. for aMFS, enabling the encapsulated description of logistics modules and their related information as well as the composition of multiple modules to a system could be developed, which is a widespread approach in research for aPS [5]. To apply the model-based description of the systems, an automated control code generation is used.

For supporting the efficient reuse and evolution, we define four requirements that have to be fulfilled by an MDE approach for aMFS: (R1) An aMFS must be built by instantiating encapsulated modules. Types of modules must be described in a holistic way, so that they can be instantiated and reused. (R2) It must be possible to build systems out of arbitrarily composable

modules. I.e. a module can either be implemented by specific hardware and software (basic modules), or by instantiating and interconnecting other modules (composed modules). (R3) Consistency must be preserved between models of interconnected modules. These models have overlapping semantics. E.g. the output of one module is the input of another module. It must be possible to formally validate or (re)create this consistency. (R4) Consistency must be ensured between models of the aMFMs and the implemented program code. While models describe the way the system is working, the program code implements this specification. When the program code or the model is changed, changes must be propagated to the other artifact, to support (ad-hoc) evolution.

The main contribution of this paper is to highlight the benefits of a more modular and meta model-based approach compared to classical MDE approaches, to ease software evolution of aMFS as a crucial part of any CPPS.

The remaining paper is structured as follows: after the state of the art in intralogistics and virtual models (Section II) a real industrial use case is introduced, which requires evolution of existing automation software (Section III). The evolution process is performed with a current, model-based approach as described in Section IV. This approach fails to fulfill all requirements stated above. In Section V the generic approach is presented, which defines composable, model-based modules. The approach preserves the consistency between the models, and between models and the program code with bidirectional rules. Section VI describes how the approach is used for development and evolution of aMFSs, and how it relates to the requirements stated above, before the paper concludes and gives an outlook on future work in Section VII.

II. State of the art in Intralogistics and Virtual Models

In the following, the state of the art in system development for intralogistics in general is presented and, subsequently, current model-based approaches for aMFSs are introduced. The section concludes with an overview of current approaches for virtual modeling, which is used in the presented approach.

A. Trends in Logistics

Today's aMFSs are static and long-living systems, which are currently mainly controlled by a programmable logic controller (PLC) programmed in accordance to the IEC 61131-3 standard. Thus, the evolution of an aMFS requires a re-engineering of the PLC control program, which is a time-consuming task as a change in the control program usually entails complex software tests [6]. Ten Hompel states that the supply chain will obviously undergo an organizational change and the way forward to the introduction of Internet of Things or Cyber-Physical Systems in the control of aMFS has been marked out due to Industrie 4.0 [7, 8]. Therefore, seven characteristic and interrelated features of Industrie 4.0 can be highlighted, i.e. digitalization, autonomization, transparency, mobility, modularization, network-collaboration and socialization. Furthermore, related technologies and concepts are validated to determine their contribution to the future development of the industrial (r)evolution. For instance, Pfohl et al. [9] propose a theoretical framework to evaluate key technologies and concepts with respect to their impact on the supply chain. The impact of Industrie 4.0 on the control strategies of supply chains, e.g. central, hierarchical/distributed or decentralized, is not addressed.

B. Model-Based Engineering of Automated Material Flow Systems

Ten Hompel and his group present different approaches of distributed control strategies [10-14]. They developed and implemented a decentralized control system based on embedded technology and Internet standards using autonomous software modules to control the aMFS. To improve the flexibility of the system, the control of the logistics functions is separated from the control of the hardware [10]. The transformation of this approach for centralized control strategies is not considered and still an open research question. For the implementation of a distributed system, Libert [13] presents a multi agent system. Thus, the author suggests that the flexibility of the control system can be improved. For the development of the multi agent system, the Java Agent Development Framework (JADE) is used, which is not compliant with IEC 61131-3. Another approach from Libert et al. [15] presents an ontology-based communication model for distributed material flow control systems. The approach is also based on software agents that represent several devices relevant to the Internet of Things. The authors focus on the development of a communication environment for decentralized software agents. Libert et al. describe several Domain Ontology Description Diagrams (DODD) based on UML, e.g. Function ontology, Transport ontology or Workflow ontology. Additionally, the authors introduce a communication ontology model for distributed aMFS based on software agents. This contribution focuses on description of the architecture of the approach. Kipouridis et al. [16] present an approach for a cloud-based platform which enables collaborative development and visualization of decentralized aMFS. The platform is being developed in the context of the research project Collaborative design of Decentralized aMFS (KoDeMat). For the control, a distributed multi agent system is used. However, a model-based approach of developing the multi agent system is not addressed.

Boschian et al. [17] present an integrated system that can be applied to manage intermodal transportation networks at operational and tactical levels. The approach can be divided into two core modules: the intermodal transportation networks reference module and the simulation module. The paper proposes a modeling approach describing the structure and behavior of intermodal transportation networks. The presented modeling procedure is a top-down technique based on UML. The knowledge base of the integrated system is defined by a reference model that foresees the system behavior and provides the data for management strategies.

Black and Vyatkin [18] describe an agent-based design of a baggage handling system applying the IEC 61499 standard. Based on function block descriptions, the decentralized system is able to run immediately without programming. However, the approach does not provide a hierarchical generalized meta model architecture which can be used to describe different modules in the field of aMFS.

To assure availability of automation systems Priego et al. [19] present a model-based design approach that supports the

definition of reconfiguration requirements of IEC 61131-3 software programs. A meta model is defined and used to automatically extend the PLC control code with the elements needed to enable reconfiguration at run time. However, the evolution of the systems is not targeted in the approach.

Aicher et al. [20] introduce a model-based approach for automatically analyzing and adapting the interfaces of aMFMs to enable interaction between neighboring modules while ensuring downward compatibility by the help of wrappers. Within the meta model, aMFMs are grouped into four levels following the System Architecture for Intralogistics [21]: The top level is the conveying area (an entire manufacturing hall), which is separated into conveying segments (such as the demonstrator plant depicted in Fig. 1). These in turn are made up of conveying groups (see Fig. 1, “merging” and “diverting” groups), which consist of conveying elements such as the aMFM roller conveyor “T22” in Fig. 1. Thus, defining aMFM compositions is limited to these levels (R2).

To check consistency between different models (R3) Feldmann et al. [22] proposed a rule-based approach. In this approach, model transformation links are deployed to define consistency rules on the meta level. When an inconsistency is identified, resolution actions are carried out manually. However, consistency checks between models and program code are not considered (R4).

C. Virtual Single Underlying Models (VSUM)

Orthographic Software Modeling (OSM) [23] is an approach for creating, organizing, and managing different views in software development. It uses a Single Underlying Model (SUM), from which different projections (also called "views") are derived. Changes in a projection are propagated to the SUM, which affects all other projections automatically. A SUM has a single, static meta model.

Vitruvius [24] is a modeling framework for handling multiple views upon a set of underlying models. Vitruvius is based on the idea of OSM, but instead of using a SUM with a static meta model, it uses multiple models and meta models. Meta model elements are related to each other with mappings and consistency rules. The models of these interconnected meta models form a virtual single underlying model (VSUM), with a virtual single underlying meta model (VSUMM). Vitruvius provides a uniform access to the underlying (meta-)models and manages the consistency in the VSUM via the consistency rules. OSM and Vitruvius handle model (and partially model-code) consistency in the software engineering domain, but are not used in CPPS.

III. Evolving Material Flow Use Case

In this section, an application example from the domain of aMFS is introduced and, subsequently, an aMFM from this example is considered in detail.

A. Introduction of the Application Example

This section details the application example of an industrial aMFS. This demonstrator provides the implementation and evaluation of different control concepts in the field of intralogistics containing 84 input and 83 output signals. The conveyor hardware consists of two driven roller tracks that are each approximately 10.8 m long and an arc for in- and outfeed of transport units (TUs) which can be filled, emptied or manipulated in various ways. The tracks are divided into separately controllable Conveying Groups and can be actuated pneumatically or electrically, thus allowing individual control of each roller conveyor. There are two stoppers which stop TUs at a specific position. To determine the direction of a TU at the T-junctions, a bar- or QR-code scanner can be added to the system via an industrial fieldbus. The connection to several control systems, such as CODESYS PLC, Siemens S7-300, S7-1500 or Nanobox PC SIMATIC IPC227 is feasible.

At the end of both rows as well as in the two T-junctions for the cross transport, there are belt diverters. By means of a belt diverter the TU is pushed over at a right angle to the conveying direction, ending up on a parallel conveyor line at the right or

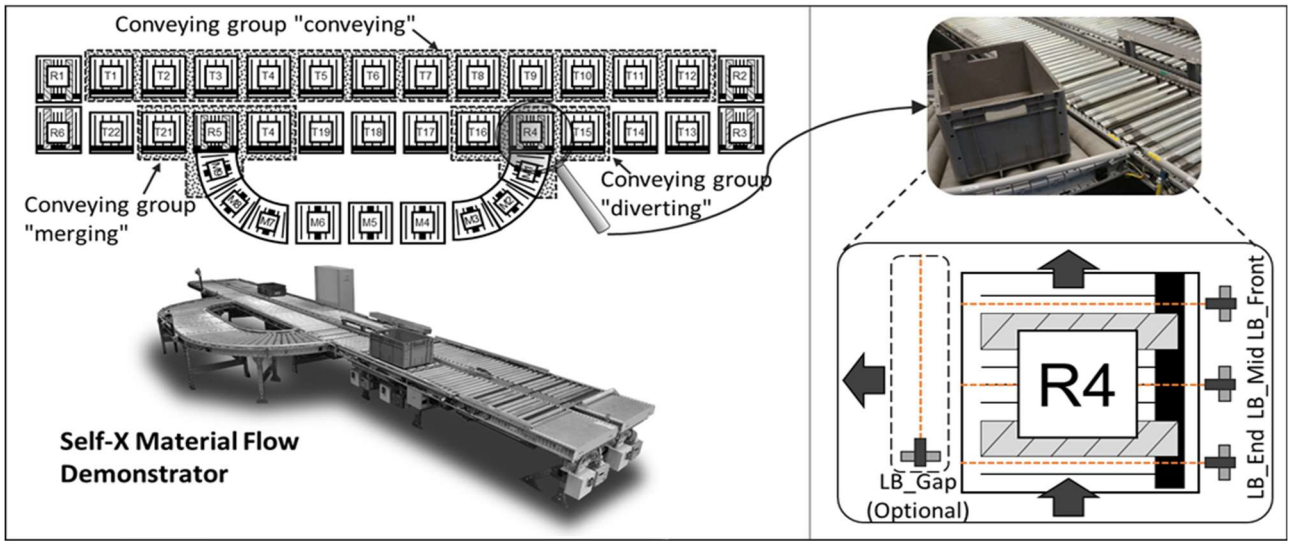

Fig. 1. Overview of conveying segment Self-X Material Flow Demonstrator. with conveying groups “merging” and “diverting” and detailed view on conveying element “R4” with optional light barrier “LB_Gap”

left side. Diversion is carried out by lifting the TU and pushing it over to the adjacent line.

B. Two Different Versions of the Belt Diverter

In the following, the model-based description of a belt diverter using the meta model AutoMFM [25] is demonstrated (cf. Fig. 2). Based on the structure of AutoMFM, the Conveying Element belt diverter contains the five (sub)classes general description, status description, function description, module interface description and control description as well as (sub)classes for actuators to control the drives of the belt diverter and sensors to read the light barriers. For the control of the actuators and the reading of the sensors, Boolean variables have been declared. For example, in order to identify whether the TU is handed over to the neighboring module successfully, the light barrier (LB) "LB_Gap" can be added to the module belt diverter (cf. detailed view of aMFM "R4" in Fig. 1, right).

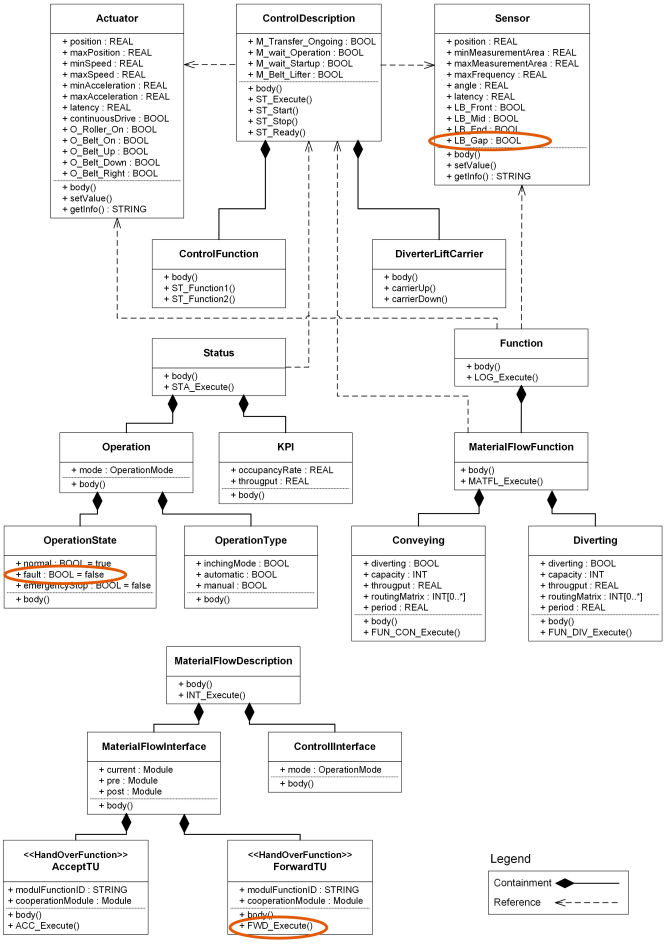

Fig. 2. Required modifications in the control software highlighted in the domain specific AutoMFM meta model

Moreover, the time to hand over the TU can be optimized, i.e. lowering the belts immediately after handing over the TU rather than using a parameterizable timer. Hence, the throughput time of the module can be increased. In addition, with this LB faults can be detected, e.g. TU is stuck. Therefore, the following modifications in the control of the module are necessary: For storing the sensor value of the added LB, the input variable "I_LB_Gap" is required. This variable has to be defined in the function block (FB) of the module and considered in the upper layer level "Group_VBC" to call the FB correctly. Moreover, the function "Fault_Detection()" as well as the Boolean variable "Fault_Detected" have to be defined, to enable the described fault detection functionality. In order to optimize the handover of a TU the function "FWD_Execute ()" in the class "ForwardTU" must be adapted (cf. Fig. 3).

```
    // BEGIN_METHOD FWD_Execute
    IF #Interface_Execute THEN
      CASE #I_HandoverType OF

        "Move_Forward":
    // ... Control code to forward TU

        "Move_Right":
          IF #I_LSU_present THEN

    // Handover with LB
            IF #HS_current.busy AND NOT #HS_succes-
sor.busy AND #I_TU_at_LB_U THEN
              #HS_current.Handover_Begin := TRUE;
            END_IF;
            IF #HS_current.Handover_Begin AND
#I_TU_at_LB_H THEN
              #M_Handover_in_progress := TRUE;
            END_IF;
            IF #M_Handover_in_progress AND NOT
#I_TU_at_LB_H THEN
              #M_Handover_in_progress := FALSE;
              #HS_current.Handover_Begin := FALSE;
              #HS_current.Handover_End := TRUE;
            END_IF;
            IF #HS_successor.Receipt_End THEN
              #HS_current.Handover_End := FALSE;
              #HS_current.busy := FALSE;
            END_IF;

          ELSE
    // Handover without LB
            IF #HS_current.busy AND NOT #HS_succes-
sor.busy AND #Timer1.Q THEN
              #M_wait_Operation := FALSE;
              #HS_current.Handover_Begin := TRUE;
            END_IF;
            IF #HS_current.Handover_Begin AND
#I_TU_at_LB_H THEN
              #M_Handover_in_progress := TRUE;
            END_IF;
            IF #M_Handover_in_progress AND NOT
#I_TU_at_LB_H THEN
              #M_Handover_in_progress := FALSE;
              #HS_current.Handover_Begin := FALSE;
              #HS_current.Handover_End := TRUE;
            END_IF;
            IF #HS_successor.Receipt_End THEN
              #HS_current.Handover_End := FALSE;
              #HS_current.busy := FALSE;
            END_IF;
          END_IF;
      END_CASE;
```

Fig. 3. Different control functions depending on whether belt diverter includes additional light barrier

After the aMFM was modeled in Eclipse, a model to text transformation to generate IEC 61131-3 compliant code was applied. In order to apply the belt diverter model in the layout-oriented editor, a PLCopenXML file dealing with the structure and behavior of an Application Composer module was generated.

In addition to the belt diverter, two further Conveying Elements, a roller conveyor and an accumulation roller conveyor, as well as seven Conveying Groups were developed to describe different logistics functions such as accumulation roller functions for two or three conveyors and merging or diverting T-

junctions. To identify TUs at the diverting T-junction, a model for the handling system, i.e. scanner, was developed.

IV. Requirements for desirable MDE approaches For Automated Material Flow Systems

A desirable approach for MDE provides a solution space, which offers a better degree of flexibility for implementing changes and updates to the system. Especially structural changes i.e. adding new components or changing the (mechanical) configuration/layout of aMFMs require a more modular approach. In addition, it is important and beneficial if the modules are described in an abstract way, so that overarching concerns in terms of a range of aspects/domains can be handled. This means that not only the mechanical layout of a changed system can be checked for consistency but also the interconnection of the components in terms of electrics and software (mechatronic / software interfaces, workflow, production planning etc.) can be checked for compliance and compatibility.

A requirement for a solution is to provide a range of description layers, which address various aspects as well as different abstractions in order to address the consistency problem at the appropriate level and aspect. Therefore, it is possible to create composable modules at an arbitrary number of layers. In order to achieve a high degree of flexibility, a desirable approach also addresses the meta level. It describes a rich feature set at the type level for various aspects of the system. This provides a more flexible and scalable approach to system design and especially system evolution at the same time. Given such a multi-level / multi-domain approach the specifications can be used to check consistency and compatibility across domains and levels. In the past such approaches have also been used only to generate more detailed levels from an abstract level. Such approaches can be characterized as code generation / compilation of implementations from specifications. However, such approaches often provided only one way of working and local changes to the more detailed levels (the implementation) could not be related to the more abstract specifications. Thus, a more global check whether the locally introduced changes are still compliant with other components and aspects of the system was not possible. This lack of global checks leads to the disregard of bidirectional consistency relationships between all meta levels and across all domains of aMFMs.

As another requirement, an approach should offer the basis to detect changes and possibly areas to check consistency. A change should be propagated to other levels and domains in order to handle checks for compliance or to make appropriate changes as well. A desirable approach provides the description levels from the implementation to abstract specifications at the meta level of components covering also a range of related domains for the area of CPPSs. The elements of the various description levels also have a modular nature in order to support the checking of interfaces and good (re)configuration opportunities.

In detail, the model-based engineering of CPPS comprises development artifacts of different domains. AMFMs in the proposed approach comprise hardware and corresponding software to realize specific atomic logistical steps, such as the movement, scan, or manipulation of TUs. Their functionality is implemented with PLCs. PLCs have no explicit knowledge of the entire context in which they are embedded, and solely react on sensor values they observe. The comprised software allows to access the modules and trigger a certain behavior. The modules usually have limited knowledge about their context by interpreting sensor values, but no explicit model of their neighborhood. The *coordinating software* describes the overall intralogistics process. It uses the aMFMs to execute this process. AMFMs and coordinating software are highly interrelated.

Three layers exist such an MDE approach (cf. Fig. 4): The lowest layer includes real-life objects in terms of the system, i.e. program code or hardware. Models are used to describe the real-life objects with appropriate abstractions. Models, their elements and their relationships are typed by meta models and their meta model elements. MDE of intralogistics systems implies that model-based artifacts for all development domains exist. For each domain one or more meta models exist, with partly overlapping semantics. E.g. an aMFM provides a function, which is referenced in a process, or the output of one module is the input of another module. The program codes for the PLCs of aMFMs describe input-output relations for sensors and actuators (cf. Fig. 3). The model representation of this code describes the sensors and actuators, its possible states, and values that the module measures and sets (see Fig. 2). A corresponding meta model includes types of these elements. An aMFM is built up using possibly multiple such PLC programs with associated hardware as reusable units. Reuse can be achieved through building or assembling the hardware of the module and transferring the generated code to the PLCs. The coordinating software, that organizes the material flow process, can access the functions and current states of the aMFMs on the code level.

When an aMFS is built in an environment, that does not match earlier assumptions, unforeseen faults or errors are possible. E.g. the environmental temperature and air pollution are higher than expected, so that the system has to be stopped regularly, to be cooled and cleaned. An automation engineer then evolves the system to increase productivity, and informally documents the changes made to the system. Such change documents can easily be misunderstood or get lost. In both cases, an inconsistency exists between the expectations towards the system's implementation and the actual implementation. A shortcoming of current approaches is that these ad-hoc changes are not properly included in the development and evolution process. It is important that these changes are included in an ordered evolution process, so that the changes are subject to consistency management, and create reusable modules.

V. Concept of Coupled Models and Meta Models to support evolution

We propose a more integrated style of MDE for aMFS. This style uses aMFMs that contain the definition of an electrical/mechanical assembly, program code for the PLCs, and models. Elements of the models of multiple aMFMs can be mapped to each other. These mappings are subject to consistency rules. The software that coordinates the process is developed based on process models, that reference model elements of aMFMs.

Fig. 4 sketches the different development domains as columns. The rows show the meta modeling abstraction layers of MDE [26, Chapter 7]. Following this structure, the real-life objects (electrical/mechanical hardware and software) are in the

meta model layer M0. A model representation of the system is in M1. The meta models are described in layer M2. They define the concepts of the respective domains. The z-axis represents evolution, where different versions of the artifacts may exist in each cell. The arrows represent consistency relationships. Three types of consistency relations exist. *Vertical consistency* is the consistency between meta modeling levels. It is independent of the development domains. The vertical consistency between meta models and models is well understood and thoroughly described [27, Chapter 5]. Approaches exist for validating and enforcing vertical consistency between M1 and M0 in the software engineering domain. In previous work, we considered the vertical consistency between software architecture models and program code in the software engineering domain [28, 29]. *Horizontal consistency* is the consistency between artifacts of the same meta modeling level, independent from the development domain. Mappings can be created between meta model elements (M2) of different domains, based on semantic interrelationships. These mappings can be used as the types for corresponding mappings on the model level (M1). The real-world objects (M0) are very heterogeneous, and hard to grasp in a formal context. Nevertheless, the consistency between these elements benefits from derived consistency relations. If formal consistency relations exist vertically between real-world objects and their model representation, then the consistency can be checked on the model level.

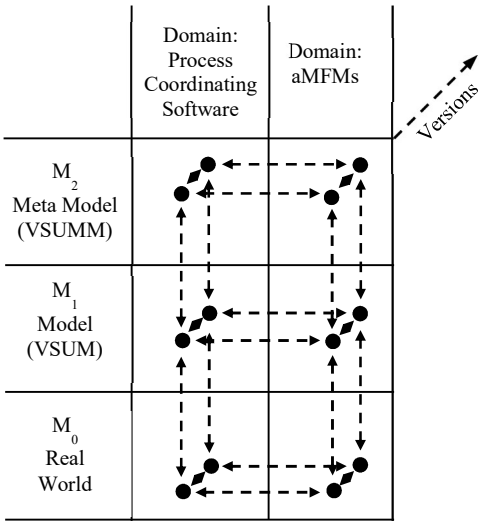

Fig. 4. Coupling of artifacts in MDE for aMFSs in the proposed approach

The consistency between different versions of artifacts (z-axis) is the *lateral consistency* in Fig. 4. It has to be examined in dependency with the meta modelling level, and partly to the development domain. The evolution of real-world objects (M0) invalidates the consistency to the respective models, and to the interrelated real-world objects of the same development domain. The model representation (M1) of the real world objects has to be updated for certain changes in M0. E.g. when an aMFM is updated with a new functionality, the new functionality has to be part of its model. Additionally, it might be necessary to propagate changes to the other development domain. E.g. when a new functionality is modelled, the functionality has to be made "known" to the coordinating software. Changes in one model might require changes in other models to reestablish the consistency. When no abstractions exist in M1 for representing the necessary changes, it can become necessary to change the meta model (M2). Meta model changes can have a huge impact on the overall system, because the meta models are a shared resource for many other artifacts. Typically, extensions are possible with a small impact, but changes and deletions of elements are posing bigger challenges. At the meta modelling levels M2 and M1, evolution consistency relationships are independent from the development domain. We define three maturity levels for handling the aforementioned consistency relationships: 1) Formal checks to ensure consistency concerning the introduced consistency relations exist (checking is done either manually or automatically). 2) Differences between the views are highlighted by an automated validation. 3) Inconsistencies are automatically resolved.

The following aspects should be modeled for aMFMs using domain-specific models from a software perspective: (a) *Functions* describe how they interact with TUs. Examples are "scan" for a barcode scanner or "transport" for a roller conveyor. (b) Abstract *states* define a named configuration of the module. Examples are "transporting" or "stopped", and "filled" or "empty" for a roller conveyor. (c) A *context* abstractly describes other aMFMs with which a module directly interacts, and which functions they provide or require. E.g. a roller conveyor has a predecessor from which TUs are taken and a successor to which a TU is given. For a given set of modules, elements of context models are semantically overlapping: When a roller conveyor A has a successor B, then A is the predecessor of B. Giving a TU from A to B is semantically equivalent to taking it from A for B. Context elements may have pre- and post-conditions as properties. E.g. a conveyor input of a specific conveyor variant assumes that a TU has a temperature $0°C < x < 100°C$; its output guarantees a temperature of $0°C < x < 100°C$ for example in a cooling line for particle boards.

AMFMs are composable. A composed aMFM consists of multiple interconnected instances of underlying aMFMs. They build more abstract functions and states. Elements of multiple context model instances can be coupled with each other by mapping elements. E.g. the output of one roller conveyor has to be mapped to the input of another. This composability is a key to handle large systems with a large set of models.

Mappings also exist between composite aMFMs and their composed modules. E.g. Fig. 5 shows context models of the roller conveyor module a) and the belt diverter module b). A group of roller conveyors and a belt diverter build a diverting group. Part c) of Fig. 5 shows the internals of the group. Three roller conveyors and a belt diverter are instantiated, and their context is mapped, so that the output of a module is the input of another. The diverting group is a reusable composed aMFM. Its functionality is realized by the implementation and models of its composed modules. The group itself has abstract models which describe its functions, states, and context. Part d) shows a sketch of the context model of the diverting group, which hides the internal implementation. It can be used in different contexts without the necessity to change the implementation or model of the composed modules. The diverting group of the example would have the function to divert or not divert a transportation unit onto a second line. Its states contain "transport-

ing", or "stopped". Its context has one predecessor and two successors, one for each line. The mapping of context models can be used to ensure the consistency between aMFMs. Consistency rules can be defined over meta model elements of the context model of aMFMs. E.g. a conveyor input can be mapped to a conveyor output, given the pre- and post-conditions match.

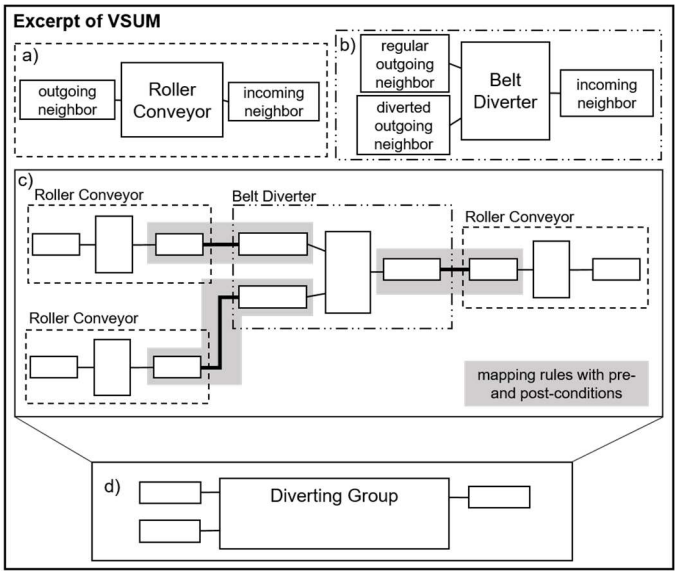

Fig. 5. Hierarchical view upon composed context models (excerpt of VSUM)

Composed aMFMs in the presented approach are aMFMs themselves, so that they can be subject to composition. An arbitrary number of description layers can be built this way. E.g. in the use case of Section III, the roller conveyors and belt diverters are basic aMFMs. The diverting group comprises three interconnected roller conveyors and a belt diverter. This group can be part of a more abstract aMFM ranging from the diverting group to the merging group in Fig. 1. The resulting composed aMFM can then be reused in similar systems, to implement a diverted line for manually checking TUs, and remerging them after fixing any errors. Analogously, the complete system of the use case can be an aMFM, that may be instantiated multiple times for reuse, probably as a module of a larger system.

The model of a complete system can be built by creating and interconnecting instances of the top-most aMFMs. The mapped models can be merged virtually. A software that controls the material flow process can be developed based on the merged model as a virtual single underlying model (VSUM) [23]. The VSUM is not a separate model, but interconnects the models of the aMFS. It describes the functions, states, and context of the overall system. Fig. 5 shows an excerpt (a view) upon the VSUM of the given use case. The meta model of the VSUM is the set of the meta models of its models, with mappings as described above. It is called virtual single underlying meta model (VSUMM) [23].

The coordinating software is implemented using a process-oriented modeling language such as BPMN [30]. It defines and monitors the material flow. The activities and decisions of the process models are mapped to functions and states of the VSUM of instantiated aMFMs. The process model therefore contributes to the VSUM(M). This mapping of process and functions/states can be used to ensure the consistency between process and system.

VI. Development and Evolution with Coupled (Meta) Models

Using the described MDE method, the (top-down) development of an aMFS consists of the following stages: First the material flow must be planned and described in a requirements document. Then the system can be assembled on a model-level by reusing aMFMs from a catalogue or defining new aMFMs where necessary. The elements of the context models must be mapped to each other for all aMFMs. This is not necessary for the children of reused composite aMFMs, because the instantiation and mapping is already included in the composition. The consistency rules must be preserved by the final mapping. A process model must be created that defines the material flow, and its activities must be mapped to functions and states of the system model. Program code for the PLCs and the process implementation can be generated and adapted to the specific needs of the process.

The presented module-oriented approach of MDE supports on-site evolution. An on-site evolution (here usually regarded as bottom-up process) consists of the following stages: An expert (e.g. a development engineer from the domain of aMFS) changes the system's hardware or software on-site. The engineer adapts the models of the affected aMFMs, to make them consistent with the system changes. The engineer defines new versions of the aMFMs with changed hardware, code, or models. In the VSUM, the consistency between the models must now be re-evaluated. Consider the example system, where a belt diverter module is replaced on-site with a module, that has an additional light barrier, as described above. The hardware, program code, and model of the diverting group is updated, and a new version of the module is defined. The updated diverting group can now be in an additional state, that indicates a stuck TU. This state has to be considered by the coordinating software. These differences are detected by consistency rules, e.g. implemented with model checking algorithms, model-transformations, and program code parsers/generators. To support reuse, the new version of the aMFM should be registered in a catalogue. In addition, even "simple" code maintenance e.g. substituting old program code by new less faulty one can be addressed here as well. Given a modular structure, limited model checking of only the relevant code slices can be used to check the compatibility of the changed code with the rest of the module's interior and the result of a correction / change can be traced as well to expose possible desired / undesired interactions with the rest of the system.

This style of aMFMs makes the reuse of basic and compositional modules easier to achieve. It increases the reusability of aMFMs and the evolvability and maintainability of aMFSs. The requirements stated in Section I are fulfilled as follows: (R1) The approach explicitly describes modules, that contain composed mechanical/electrical hardware, models, and program code. (R2) Modules in the approach are composable on arbitrary levels. These modules can be reused in other contexts. (R3) The consistency between different models is handled with the notion of the VSUM. I.e. semantically associated model el-

ements can be mapped to each other with consistency preserving rules. (R4) The code is considered a part of the VSUM. Consistency rules exist to create or adapt program code from (changed) models, and to create models from program code.

VII. Conclusion and Future Work

In this paper we presented a novel approach of model-driven engineering for aMFSs. The presented approach aims at increasing the reusability and decreasing the error-proneness of current development approaches by defining aMFMs. AMFMs comprise hardware, program code, and models that describe their functions, states, and abstract context. Composite aMFMs and complete systems can be defined by instantiating aMFMs and mapping their context model elements to each other in accordance to consistency rules. When such a system evolves, the changes are propagated to the models of new variants of aMFMs, and the consistency is re-evaluated. Changed aMFMs are made available for reuse, to benefit from the knowledge generated in production.

In future work, we plan to automate the propagation of changes in a system towards the models. The mapping between context elements of aMFMs can be automatically created by communication between the aMFMs. The model of the complete system can then be updated automatically and can be checked against its consistency rules. The new functions and states can be made available to the process implementation.

References

[1] S. Feldmann, J. Fuchs, and B. Vogel-Heuser, "Modularity, variant and version management in plant automation - Future challenges and state of the art," in *Proceedings of International Design Conference, DESIGN*, 2012, vol. DS 70, pp. 1689-1698.

[2] B. Vogel-Heuser, J. Fischer, S. Feldmann, S. Ulewicz, and S. Rösch, "Modularity and Architecture of PLC-based Software for Automated Production Systems: An analysis in industrial companies," *Journal of Systems and Software*, vol. 131, pp. 35-62, 2017.

[3] B. Vogel-Heuser *et al.*, "Challenges for Software Engineering in Automation," *Journal of Software Engineering and Applications*, vol. 7, no. 5, pp. 440-451, 2014.

[4] S. A. Bohner and S. Mohan, "Model-based engineering of software: Three productivity perspectives," *Proceedings - 33rd Annual IEEE Software Engineering Workshop, SEW-33 2009*, pp. 35-44, 2010.

[5] V. Vyatkin, "Software Engineering in Industrial Automation," *IEEE Transactions on Industrial Informatics*, vol. 9, no. 3, pp. 1234-1249, 2013.

[6] X. Hoang, A. Fay, P. Marks and M. Weyrich, "Systematization approach for the adaptation of manufacturing machines," in *IEEE International Conference on Emerging Technologies and Factory Automation*, 2016, S. 1-4.

[7] M. ten Hompel and M. Henke, "Logistik 4.0," in *Industrie 4.0 in Produktion, Automatisierung und Logistik: Anwendung · Technologien · Migration*, T. Bauernhansl, M. ten Hompel, and B. Vogel-Heuser, Eds. Wiesbaden: Springer Fachmedien Wiesbaden, 2014, pp. 615-624.

[8] B. Vogel-Heuser, T. Bauernhansl, and M. Ten Hompel, *Handbuch Industrie 4.0 Bd.3 Logistik*. Springer Berlin Heidelberg, 2017.

[9] H.-C. Pfohl, B. Yahsi, and T. Kuznaz, "The impact of Industry 4.0 on the Supply Chain," in *Proceedings of the Hamburg International Conference of Logistic (HICL)-20*, no. August, pp. 32-58, 2015.

[10] M. ten Hompel, S. Libert, and U. Sondhof, "Distributed Control Nodes for Material Flow System Controls on the Example of Unit Load Conveyor and Sorter Facilities," *Logistics Journal*, no. NOVEMBER, pp. 1-10, 2013.

[11] C. Timm *et al.*, "Decentralized Control of a Material Flow System enabled by an Embedded Computer Vision System," in *IEEE International Conference on Communications Workshops (ICC)*, 2011, pp. 1-5.

[12] A. Kamagaew, J. Stenzel, A. Nettstrater, and M. Ten Hompel, "Concept of cellular transport systems in facility logistics," in *ICARA 2011 [- Proceedings of the 5th International Conference on Automation, Robotics and Applications*, 2011, pp. 40-45.

[13] S. Libert, "Beitrag zur agentenbasierten Gestaltung von Materialflusssteuerungen," Dissertation, Faculty of Mechanical Engineering, Technical University of Dortmund, 2011.

[14] M. ten Hompel, S. Libert, and M. Roidl, "Erarbeitung von Methoden und Regeln zur Gestaltung agentengestützter, dezentraler Steuerungen für den Einsatz in komplexen Materialflusssystemen. Forschungsbericht," no. 15313, 2009.

[15] S. Libert and M. Ten Hompel, "Ontology-based communication for the decentralized material flow control of a conveyor facility," *Logistics Research*, vol. 3, no. 1, pp. 29-36, 2010.

[16] O. Kipouridis, M. Roidl, W. A. Günthner, and M. Ten Hompel, "Cloud-Based Platform for Collaborative Design of Decentralized Controlled Material Flow Systems in Facility Logistics," in *Proceedings of the 4th International Conference LDIC*, 2015, pp. 313-322.

[17] V. Boschian, M. Dotoli, M. P. Fanti, G. Iacobellis, and W. Ukovich, "A metamodeling approach to the management of intermodal transportation networks," *IEEE Transactions on Automation Science and Engineering*, vol. 8, no. 3, pp. 457-469, 2011.

[18] G. Black and V. Vyatkin, "Intelligent component-based automation of baggage handling systems with IEC 61499," *IEEE Transactions on Automation Science and Engineering*, vol. 7, no. 2, pp. 337-351, 2010.

[19] R. Priego, A. Armentia, E. Estévez and M. Marcos, "On Applying MDE for Generating Reconfigurable Automation Systems," in *IEEE International Conference on Industrial Informatics*, 2015, pp. 1233–1238.

[20] T. Aicher, D. Schütz, M. Spindler, W. Günthner and B. Vogel-Heuser, "Automatic analysis and adaption of the interface of automated material flow systems to improve backwards compatibility," in *20th IFAC World Congress*, Jul. 2017, pp. 1217-1224.

[21] VDI/VDMA, "System Architecture for Intralogistics (SAIL) – Fundamentals," in *VDI Recommendation 5100*, 2011.

[22] S. Feldmann, M. Wimmer, K. Kernschmidt and B. Vogel-Heuser, "A comprehensive approach for managing inter-model inconsistencies in automated production systems engineering," in *IEEE International Conference on Automation Science and Engineering*, Aug. 2016, pp. 1120-1127.

[23] C. Atkinson, D. Stoll, C. Tunjic: "Orthographic Service Modeling", in *IEEE 15th International Enterprise Distributed Object Computing Conference Workshops*, Institute of Electrical & Electronics Engineers, 2011.

[24] M. E. Kramer, E. Burger, and M. Langhammer: "View-centric engineering with synchronized heterogeneous models". Proceedings of the 1st Workshop on View-Based, Aspect-Oriented and Orthographic Software Modelling, VAO '13, ACM, 2013

[25] T. Aicher, D. Regulin, D. Schütz, M. Lieberoth-Leden, Christian, Spindler, W. A. Günthner, and B. Vogel-Heuser, "Increasing flexibility of modular automated material flow systems: A meta model architecture," in *8th IFAC Conference on Manufacturing Modelling, Management and Control (MIM)*, 2016.

[26] Object Management Group, OMG Meta Object Facility (MOF): "Core Specification, Version 2.5.1", 2016, http://www.omg.org/spec/MOF/2.5.1

[27] A. Kleppe, "Software Language Engineering: Creating Domain-Specific Languages Using Metamodels," Addison-Wesley Professional, 2008, ISBN 978-0-321-55345-4.

[28] M. Konersmann, "A Process for Explicitly Integrated Software Architecture," *Softwaretechnik-Trends*, ISSN: 0720-8928, vol. 36, no. 2, 2016.

[29] M. Konersmann and M. Goedicke, "A Conceptual Framework and Experimental Workbench for Architectures," in *Software Service and Application Engineering*, vol. 7365, M. Heisel, Ed. Springer Berlin Heidelberg, 2012, pp. 36–52.

[30] OMG, "Business Process Model and Notation (BPMN), Version 2.0". Object Management Group, 2011, http://www.omg.org/spec/BPMN/2.0