Modularized Control Algorithm for Automated Material Handling Systems

Markus Spindler*, Thomas Aicher**, Daniel Schütz**, Birgit Vogel-Heuser**, Willibald A. Günthner*
*Institute for Materials Handling, Material Flow, Logistics, Technical University of Munich, Germany
(e-mail: {spindler, kontakt}@fml.mw.tum.de)
**Institute of Automation and Information Systems, Technical University of Munich, Germany
(e-mail: {aicher, schuetz, vogel-heuser}@ais.mw.tum.de)

***Abstract*—Automated material handling systems are widely used in industry due to high throughputs and good process quality. In order to fully utilize their potential, these systems have to be custom-built, requiring individually created control software. The engineering process of the control software includes extensive manual programming, because a modular structure of the control algorithm with predefined modules is often not available. We describe one such modular control algorithm, based upon a two-layer architecture that separates the hardware control from the material flow control. Our modular control algorithm allows predefined modules for both the conveyor hardware and material flow elements to be created and then assembled in the needed configuration. As a result, the control software for a custom-built material handling system can be created by arranging predefined modules and setting their parameters, thus without manual programming. The proper functionality of our control algorithm is demonstrated by a simulation example using emulated hardware.**

Keywords— Intralogistics, material handling systems, model driven engineering (MDE), automation, code generation

I. Introduction

Due to their high handling capacity, short throughput times, and good process quality, automated material handling systems are widely used in transport, distribution, and industry [1]. As a result of increasing e-commerce and thus easy consumer accessibility, distributers are shifting their business model from business-to-business (B2B) to business-to-consumer (B2C) [2]. Since end consumers order single units to meet immediate needs, this shift is characterized by an increasing amount of shipments and more demanding requirements in terms of delivery time and process quality. These requirements can be fulfilled best with automated material handling systems that are custom-made for a specific purpose. Therefore the demand for these systems, and thus the need to effectively plan such systems, is growing.

One important task during the planning of automated material handling systems is the engineering of the control software for the material handling equipment. The development of this custom-built control software is time-consuming and can be improved by providing supporting engineering tools [3]. We develop one such supporting engineering tool for the design of the control software for stationary automated material handling systems, where stationary means that the material handling equipment is affixed to the ground. Examples of such components are roller conveyors or chain transfers, see Fig. 1. The idea behind the creation of the control software is to solely utilize predefined software modules in order to avoid manual programming. The engineer creates the model by taking software modules out of a library, arranging them according to the layout and linking them together. To enable such a design process, the control algorithm needs to guarantee proper function in any potentially desirable layout. In this paper, we state one such control algorithm and demonstrate that it functions properly in an example with emulated material handling hardware.

The paper is structured as followed: first, we give an overview of the relevant literature, after which we state the modular architecture concept, followed by the presentation of the control algorithm. The proper function of this algorithm is then demonstrated by an application example using emulated hardware, and the conclusion summarizes our findings.

Fig. 1. Example of a stationary material handling system from SSI Schäfer.

This work was supported by the Bavarian Research Foundation as part of the project aComA, project no. 1098/13

II. Related Literature

Automated material handling systems are usually controlled by cyclic controls, e.g. programmable logic controllers (PLC), and can either have centralized or decentralized control. Decentralized control approaches make it possible to achieve higher flexibility and are considered in a number of different research projects [4], [5]. Today, centralized control is commonly used in the industry, as it is an well understood method that can achieve high throughputs because of fast communication inside the centralized processor. Our work focuses on centralized control structures, because they represent the state of the art in the industry.

The productivity of software engineering can be improved through model-driven engineering (MDE) [6]. The idea of MDE is to use predefined modules to create a model of the custom-built system. This model is then used to generate further information for the project, e.g. the control code or wiring diagrams. In order to use MDE, a domain-specific system description is necessary, which is implemented in a software tool that supports the engineer in creating the model. One example of such an approved tool is CODESYS [7]. Domain-specific system descriptions exist in various automation sectors, including but not limited to: building automation [8], process engineering [9], and manufacturing systems [10], [11], [12]. So far, for automated material handling systems no such domain-specific description exists, therefore MDE cannot be used at the moment.

Nevertheless, several standards and research projects have undertaken to describe automated material handling systems, each of them focusing on a different aspect. These descriptions are a valuable starting point for developing an integral domain-specific description of automated material handling systems that can be used to automatically generate the control code. In the following, we will give an overview of these works. Centralized control systems are usually structured in the form of an automation pyramid, in which different control tasks are assigned to different control levels of the pyramid. The interfaces between the control levels are considered in the standard *SAIL – System Architecture for Intralogistics* [13]. Since SAIL includes stationary and freely movable conveying systems, the interfaces are defined abstractly. Modularization conceptions of automated material flow systems have been developed in the context of a number of research projects. Wilke [14] proposes a functional modularization, which means that each module encapsulates a function and comprises the hardware as well as the software that is needed to perform that function. Aicher et al. [15] present a meta-model for automated material flow systems that describes the material flow system as individual modules which are connected through standardized interfaces. This model can be used as the basis for generating code for centralized and decentralized control structures. For material handling systems, various control structures have also been developed in the context of research projects. A two-layer architecture for controlling decentralized automated material handling systems is presented by ten Hompel et al. [16], which makes it possible to control the material flow with uniform logic modules and execute tasks with dissimilar conveyors. This architecture establishes a separation between the conveyor layer responsible for controlling the sensors and actuators and the logic layer responsible for decision making. The problem of developing the control software for conveyor systems is also considered by Black et al. [17] and Lepuschitz et al. [18]. All of these works develop control structures that consider a one-to-one correlation between the software modules and conveyor hardware, which represents a constraint on how flexibly the modules can be arranged. In this paper, we present a modularized control structure in which several modules can access a single conveyor, thus allowing the modules to be arranged more flexibly.

III. Modular Architecture Concept

The control algorithm presented in this paper builds upon an existing two-layer architecture for the design of control software for automated material handling systems [19]. The bottom layer of this architecture contains conveyor modules that are responsible for controlling the specific material handling hardware. The upper layer contains logistics modules that are responsible for the material flow control; a layout is shown in Fig. 2 top. Conveyor places (CP) provide the connection between these two layers. The CPs are places within the material handling system where transport units (TU) are detected via a sensor and can be conveyed by an actuator in one or several directions. Each CP is allocated to exactly one conveying module and either one or two logistics modules. The CPs that are allocated to two logistics modules are handover CPs; at these places the responsibility for the material flow control of the TU is handed over from one logistics module to another. The separation of hardware control and material flow control with linking via CPs enables a modular control structure in which the control software of a custom-built automated material handling systems can be created by predefined modules.

In the following subchapters, we first present the different module types used in the control algorithm (subchapter A), followed by data types (subchapter B) and interfaces (subchapter C). Afterwards, in chapter IV, the methods of the control algorithm are explained.

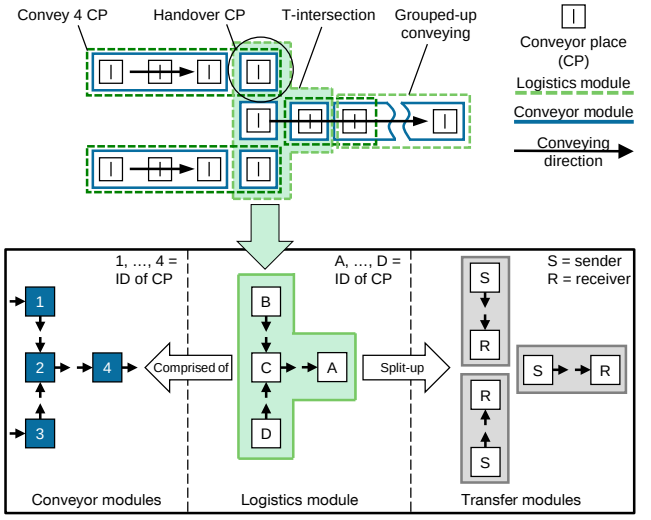

Fig. 2. Layout representation of the two-layer-architecture (top) and relationship between conveyor modules, logistics modules, and transfer modules (bottom).

A. *Module types and interconnection between the modules*

Three module types are used for the control algorithm: conveyor modules, logistics modules, and transfer modules, see Fig. 2 bottom. Conveyor modules represent the conveyor hardware; each conveyor module therefore represents one conveyor. Logistics modules manage the material flow within spatial subsections of the material handling systems, e.g. straight conveying areas or intersections, and are thus comprised of one or several conveyor modules. Each logistics module is split up into several transfer modules, where each transfer module controls exactly one transfer. A transfer describes the process of passing a TU from a sender to a receiver on a single-lane conveyor, where the single lane can be used as either a one-way or a two-way channel. The hierarchical interconnection between the three module types is depicted in Fig. 3 (left) using the class diagram of the unified modeling language (UML). To support the connection and information exchange between the logistics modules and conveyor modules developed by different manufacturers, standardized interfaces are utilized. Hence, (pre-)processed sensor and actuator information can be interchanged. This figure also shows the five cycle phases (A, …, E), which are chronologically performed during each control cycle, starting with cycle phase A. The tasks that the different module types perform during the different cycle phases are explained in the following subchapters.

Conveyor modules

The conveyor modules can be seen as workers, which have two tasks: generating the conveyor's state during cycle phase A and executing the commands during cycle phase E.

During cycle phase A each conveyor module generates its state, which is comprised of its occupancy state and the state of its conveying mechanisms. Generating the states is accomplished by reading and interpreting the sensor information. For the interpretation of the sensor information some prior information is necessary, since some sensor and actor states are ambiguous, e.g. a detected TU in combination with a switched-on drive can either mean a leaving or incoming TU. This prior information is obtained from the conveyor state at the end of the previous control cycle.

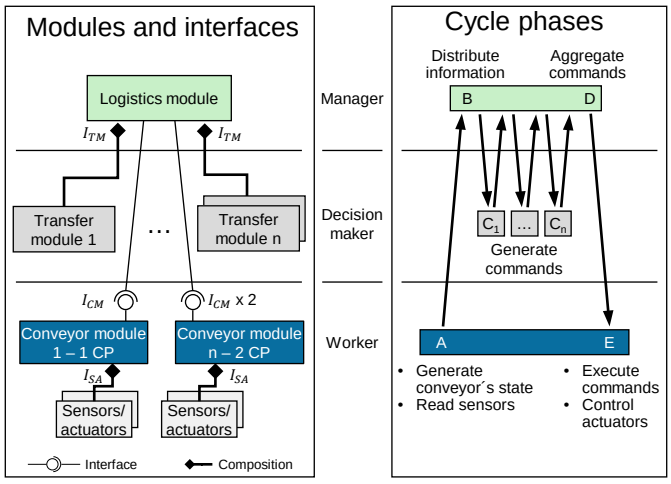

Fig. 3. Modules, interfaces and cycle phases.

During the second call of the conveyor modules, which occurs in cycle phase E, the conveyor modules execute the occupancy commands and the conveying commands, which are both transmitted from the logistics modules. The occupancy commands are used to lock or open certain conveying places, which may be used to dedicate conveying places to a particular transfer and ensure that these conveying places can safely perform the transfer. Conveying commands contain information for the actuators. These commands transmit information about preparing a conveyor for a specific transport, e.g. bringing a chain transfer into the correct handover position, or performing a specific transport. To execute these commands, knowledge of the conveyor's process and the configuration of the actuators is needed.

Logistics modules

The logistics modules perform managing tasks. They are responsible for material flow control within spatial subsections of the material handling systems and for passing commands to the correct conveyor modules. Material flow control is comprised of two tasks: sending a TU to the correct sink and observing right of way, which guarantees the desired material flow and prevents the system from deadlocking. In order that material flow control may be properly accomplished, the minimum range of these spatial subsections is subject to the restriction that each handover CP of a logistics module must have exactly one conveying direction, either an inlet or an outlet. The upper limit of the spatial expansion is limited by defining generic modules in such a way that they have good reusability.

During cycle phase B the logistics modules manage the material flow control. This requires information about the destination of the TU, which can either be static or provided by a material flow computer. The result of material flow control is clearance information to the transfer modules, which describes whether a transfer is allowed to proceed. This clearance information and the conveyor's state information is passed to the corresponding transfer modules.

During cycle phase D the logistics modules aggregate the commands from the transfer modules, which is necessary because each transfer module works independently. The aggregated information is then passed to the corresponding conveyor modules.

Transfer modules

The idea of the transfer modules is to discretize the material handling systems and therefore to describe all conveying tasks within the system as transfers. All transfers consist of two CPs, a sender and a receiver, and a conveying mechanism that enables the TU to be conveyed from the sender to the receiver. For one-way transfers the sender and receiver roles are fixed, whereas for two-way transfers the sender and receiver roles alternate. This implies that a two-way transfer can be seen as two one-way transfers in alternate directions.

One transfer consists of five steps, three of which are handled by the transfer module and the other two of which are handled by the conveyor modules; see Fig. 4. First, the transfer module detects that the sender CP and receiver CP have the correct occupancy, meaning that the sender has the

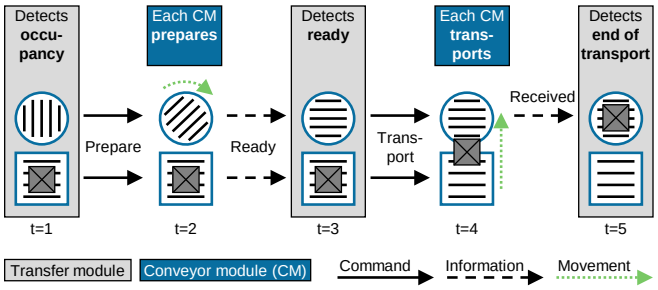

Fig. 4. Process of a transfer.

occupancy state *occupied* and the receiver the occupancy state *free*. As soon as this is detected, the transfer modules send the *prepare* command to both conveyor modules. In the second step the conveyor modules receive this command and prepare the conveyors for the transport; in the depicted example this means turning the turn table to the correct position. Once the preparation is complete, the information *ready* is sent to the transfer modules. As soon as the transfer module has received the information *ready* from both conveyors, step three starts: the transfer module sends the *transport* command to both conveyor modules. This *transport* command starts the fourth step, in which each conveyor module executes the transport, hence transporting the TU from the sender to the receiver. After the receiving CP has received the TU, the corresponding conveyor module sends this information to the transfer module, which triggers the fifth and final step: completion of the transport.

Breaking down all conveying tasks into transfers makes it possible to assemble all logistics modules out of the two basic transfer module types: one-way transfers and two-way transfers. Hence, the transfer modules need only be worked out only once, and can then be combined in the correct manner to create the logistics modules.

B. Data types utilized in the architecture

The basic elements of the algorithm are the CPs. Each CP consists of two different elements: an *occupancy*, which represents the sensor information, and one or several *conveying arrows*, which represents the conveying mechanisms that are executed by the actuators. Additionally, a *clearance* variable is used to handle the right of way in logistics modules by allowing certain transfer modules to convey or preventing them from conveying.

Each CP has exactly one *occupancy* element, which is subdivided into three variables: a *logistics information*, an *occupancy state* and an *occupancy command*. The *logistics information* represents the interpreted sensor information, which is necessary since a detected TU in combination with a switched-on drive can either be an incoming or outgoing TU. Since incoming and outgoing are the only possible descriptions for the TU, the range of values for the *logistics information* is: *no detection*, *incoming*, and *outgoing*. The *occupancy state* represents the state of a CP within the conveying process and can take the states *free*, *occupied*, and *locked*. *Occupied* and *free* mean that the CP respectively does and does not contain a TU, but that the CP has not yet been assigned to a transfer, whereas the *locked* state indicates that the CP is currently assigned to a transfer. To assign or withdraw a CP to or from a transfer, the *occupancy commands* are used, which consists of the list of commands: *close*, *open*, and *noCommand*. The *close* command assigns a CP to a transfer, thereby generating the locked state, whereas the *open* command removes the locked state.

A *conveying arrow* describes the capability of conveying in the direction of the *conveying arrow*, which can either be onto a CP or away from a CP. The number of *conveying arrows* belonging to a given CP depends on the conveying capabilities of the conveyor, e.g. a one-way roller conveyor can convey towards the CP or away from the CP and therefore has two *conveying arrows*. The information about each *conveying arrow* is split into two variables: a *conveying arrow state* and a *conveying arrow command*. The *conveying arrow state* describes whether the CP is currently allowed to convey in the direction of the associated *conveying arrow*, and can therefore be either *ready* or *notReady*. The function that determines the value depends on three restrictions. Firstly, occupancy restrictions, e.g. conveying onto a CP is only allowed if the CP is free; secondly, mechanical restrictions, e.g. the chains of a chain transfer have to be in the correct position for a certain conveying direction; and thirdly, other restrictions, which could for example be a maximum limit for the number of TUs on a certain conveyor. The *conveying arrow commands* trigger the actuators of the conveyors. These commands are generated by the transfer modules and are executed by the conveyor modules. Their value can be: *neutral*, *prepare*, *transport* or *noCommand*. *Neutral* means that no conveying task is pending and the conveyor goes into waiting position. *Prepare* is the first part of the transfer execution, which correctly positions the mechanical components of the conveyor. As soon as the positioning of both conveyors is complete – this can be seen as synchronizing the conveyors – the *transport* command is generated, which starts conveying the TU.

C. Interfaces between modules

Data transmission between the modules is achieved via three interfaces: I_{SA}, the sensor/actuator interface between the conveyor modules and the sensors and actuators; I_{CM}, the interface between the conveyor modules and logistics modules; and I_{TM}, the interface between the transfer modules and logistics modules; see Fig. 3.

Through the interface I_{SA}, the conveyor modules read the sensor information and write the actuator commands. The interface I_{CM} between the conveyor and logistics modules operates on the CP level, so each conveyor module must provide one I_{CM} for each of its CPs. Through this interface, the logistics modules read the *occupancy state* and *conveying arrow states* from the conveyor modules during cycle phase B. During cycle phase E the read direction is reversed, so that the conveyor modules read commands from the logistics modules: specifically the *occupancy command* and the *conveying arrow command*. The interface I_{TM} between the transfer modules and logistics modules operates on the transfer level. Since each transfer module contains exactly one transfer, each transfer module possesses exactly one I_{TM}. During cycle phase C, the transfer modules read during cycle phase C the *occupancy state* and the *conveying arrow state* of both CPs of the transfer module via I_{TM}, and in the

subsequent cycle phase D, the logistics modules read the generated commands from the transfer modules, namely the *occupancy command* and *conveying arrow command*.

In the following chapter we explain how these elements – the modules, data types, the interfaces – can be combined to produce the modularized control algorithm.

IV. Presentation of the Control Algorithm

In this chapter we state the five cycle phases of the control algorithm in chronological order. The chapter is divided into five subchapters, each of which corresponds to one cycle phase.

A. *Conveyor modules – Generating state*

During cycle phase A the conveyor's state is generated. This is performed in three steps; see Fig. 5. These three steps are explained in the following paragraphs.

The method of step one separates detected TUs into incoming and outgoing units. This separation requires information from the previous control cycle, because the information of a snapshot is ambiguous, e.g. it is unknown whether a detected TU in combination with a switched-on drive corresponds to an incoming or an outgoing TU. To obtain this prior information, we used the *previous conveying arrow states* and the *previous conveying arrow commands*. This information in combination with the light barrier information represents the input of this method and the output is given by the logistics information of the CP.

In the second step, the *initial occupancy state* is generated. The *initial occupancy state* describes whether the CP is allocated to a transfer module at the beginning of the control cycle, meaning before any material flow decisions are made by the logistics modules and transfer modules for the current cycle. To generate the *initial occupancy state*, the method combines the *logistics information* with the information indicating whether the TU is already allocated to a transfer. This information is contained in the *previous closing occupancy state*. The *previous closing occupancy state* is the occupancy state from the previous control cycle at the end of the cycle, which is determined during the last cycle phase, phase E.

The third method determines whether the TU is allowed to enter or exit a conveyor in a certain direction during the current control cycle. This is indicated by setting the *conveying arrow state* to *ready* or *notReady* if the conveyor is ready or not ready for conveying, respectively. The method generates the state *ready* if three conditions are TRUE: an occupancy condition, a hardware condition and other conditions. The occupancy condition uses the *initial occupancy state* as input and indicates whether the TU is allowed to enter or exit based on the occupancy. This means that, if the CP is *free*, a TU is allowed to enter, but not to leave, and vice versa if the CP is *occupied*. If the CP is *locked*, TUs are allowed both to enter and leave, since the *locked* state indicates that the CP is already preparing for or executing a distinct transfer and therefore the occupancy condition was already TRUE in a previous control cycle. The hardware condition indicates whether the TU is allowed to enter or exit based on mechanical restrictions, e.g. the chains of a chain transfer need to be in the up position to be able to transport a TU via the chains and they need to be in the down position in order to transport in the roller direction. The decisions about the hardware conditions are based on the hardware state, which is described by sensors, e.g. end switches or distance sensors. Additional conditions can include a maximum for the number of TUs on a conveyer, which may be restricted e.g. by weight or control constraints. These are described by the "other conditions" variable.

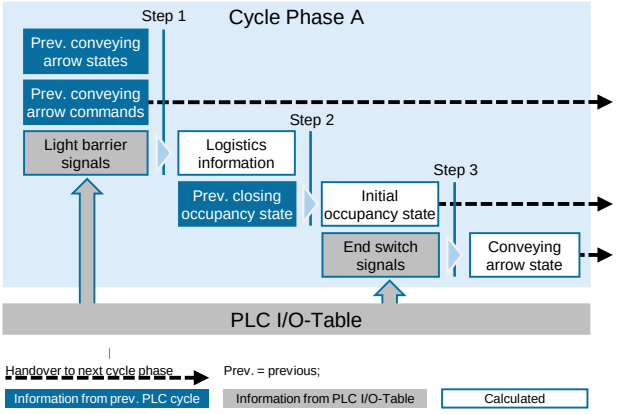

Fig. 5. Correlations and chronological execution in cycle phase A.

B. *Logistics modules – Determine clearance for transfers*

In cycle phase B the logistic modules manage right of way and routing for merger- and divider-type intersections respectively, or both simultaneously for intersections with multiple inlets and outlets. This is done by determining the clearance variable for the corresponding transfers, which is a function of the *initial occupancy states*. The *clearance* information, *initial occupancy state*, and *conveying arrow state* are then transmitted via the interface I_{TM} to the transfer modules.

C. *Transfer modules – Generating commands*

The task of the transfer modules is to generate the *occupancy commands* and the *conveying arrow commands*. The data basis for generating the commands is given by the *initial occupancy states* and *conveying arrow states*, which were generated from the conveying modules, and the *clearance* information, which is generated from the logistics modules. This information is used within a state diagram to generate the commands. Since there are only two types of transfers, one-way transfers and two-way transfers, only two types of state diagrams are necessary: one for one-way transfers and another for two-way transfers.

The simplified state diagram for a one-way transfer is shown in Fig. 6. The depicted state diagram contains the three major states – no job, prepare, and transport v with their entry actions plus the transition conditions between these three states. This simplification neglects the init state, a wait state during which the sender can already perform the preparation without clearance, and a case distinction for the prepare state, which would lead to fewer commands if either the sender or receiver is already *ready* and does not need any preparation. The output of the state diagram, the *occupancy commands* and *conveying arrow commands* are handed over to the logistics modules via the interface I_{TM}.

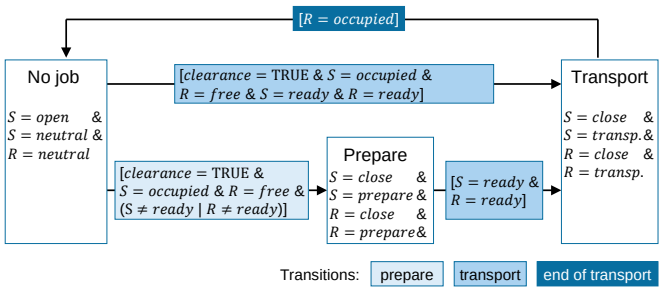

Fig. 6. Simplified state diagram of a one-way transfer. S = CP of sender, R = CP of receiver

D. Logistics modules – Aggregate commands

During the second call of the logistics modules in cycle phase D the logistics modules aggregate the commands that are generated from the transfer modules. This aggregation is necessary for the *occupancy commands*, since the receiver CP of one transfer equals the sender CP of the subsequent transfer, see Fig. 2 bottom. Therefore, if a logistics module has multiple transfers, all transfers overlap. The *occupancy commands* need to be aggregated, but there are never conflicts during any such merger, because the conditions for the command *open* and the conditions for the command *close* are mutually exclusive. The *open* command is transmitted to the sender CP of a transfer module only if the sender CP of this transfer module is *locked* and the corresponding receiver CP is *occupied*. The *close* command is transmitted to the sender CP only in two other cases: either if the sender CP is *occupied* and the receiver CP is *free* or if the sender CP and the receiver CP are both *locked*, see the state diagram of Fig. 6. The receiving CP never receives the command *open* – the occupancy of this CP is changed from *locked* to *occupied* during step 2 of cycle phase A – therefore the receiver CP also never experiences conflict. Since each *conveying arrow* is assigned to exactly one transfer (see Fig. 2 bottom), the aggregation of the *conveying arrow commands* will never lead to the case where two identical *conveying arrows* have different commands. The aggregated *occupancy commands* and *conveying arrow commands* are handed over to the conveyor modules via the interface I_{CM}.

E. Conveyor modules – Execute commands

During the call of the conveyor modules in cycle phase E the conveyor modules execute the *occupancy commands* and the *conveying arrow commands*.

To execute the *occupancy commands*, two sets of input information are necessary: the *initial occupancy states* from phase A and the *occupancy commands*, which are handed over from the logistics modules. The output from executing the *occupancy commands* is the *closing occupancy state*, which has the following correlation: the *open* command results in the *free* state, the *close* command results in the *locked* state, and if the *noCommand* command is transmitted, then the *closing occupancy state* equals the *initial occupancy state*. The determined closing occupancy state is then an input for cycle phase A of the subsequent control cycle; see Fig. 5.

Executing the *conveying arrow commands* results in commands to the conveyor's actuators and is accomplished in two steps. Firstly, for each *conveying arrow*, it is checked whether the transfer module generated a command other than *noCommand* during the current control cycle. If so, this generated command is executed, otherwise the command from the previous control cycle is executed. In the second step each conveyor module is considered and the commands for its actuators are determined. The actuator commands are a function of all *conveying arrow commands* of the conveyor module, meaning that the *conveying arrow commands* of a conveyor module are considered as a whole. This function is specific to each type of conveyor module and must be implemented individually. For example, if a chain transfer receives the *prepare* command from a *conveying arrow* in the direction of the chains, the chains must go up in order to be ready for the upcoming transfer. As soon as the transport command for this *conveying arrow* is transmitted, the drive of the chains is switched on and therefore performs the transfer.

V. Application Example

The application example is used to show that predefined modules including our proposed control algorithm can be created and that these modules can then be arranged according to a custom-built layout and used to control it. This is shown by an example implemented in the simulation software Demo3D, using a cyclic control program and emulated hardware; see Fig. 7. The layout of this example consists of seven roller conveyors, each controlled by a *roller conveyor*-type conveyor module, and one chain transfer, which is controlled by the *chain transfer* conveyor module. Green and blue boxes are created at the upper left and right ends of the layout respectively. The material flow of the layout is as follows: the TUs are first conveyed to the chain transfer and then further conveyed to the bottom end of the layout where they are deleted. The straight sections are controlled by logistics modules of type *convey* and the intersection is controlled by a logistics module of type *T-intersection* configured to give right of way to TUs arriving from the right-hand side.

The control software in this example consists solely of predefined conveyor modules and logistics modules that are interconnected according to the layout. These predefined modules can be arranged as needed, e.g. it is possible for two *T-intersections* to overlap, and the proper function of the control algorithm is always ensured. This implementation proves that the control algorithm functions properly, and the key functionality of conveyor systems could be demonstrated: firstly, right of way at the intersection is

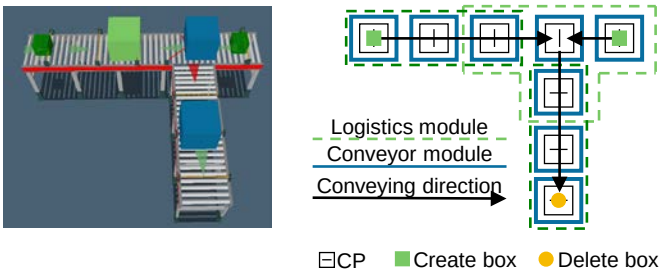

Fig. 7. Layout of the application example including a screenshot of the simulation software Demo3D (left) and a sketch of the modules (right).

key functionality of conveyor systems could be demonstrated: firstly, right of way at the intersection is always observed; secondly, the TUs pass the conveyors without stopping if several conveyors in a row are free. This is important because it prevents the drives of the conveyors from unnecessarily switching on and off, which would slow down transport and waste energy. And thirdly, in the case of a jam, the TUs do not collide and the system resumes conveying after the jam is cleared. In conclusion, this implemented example shows that the control algorithm observes right of way and conveys the TUs to their destination in a controlled manner, and therefore this example proves that our modular control software can control a merger of an automated material handling system.

VI. Conclusion and Outlook

We considered the problem of creating the control software for automated material handling systems. The demand of these systems, and thus the need to efficiently create control software for them, is growing as a result of increasing e-commerce. One way to efficiently create control software for automated systems is to create a model out of predefined modules and then automatically generate the control code from the model. In order to use such a modularized approach, the control software must also be modular.

We presented a modularized control algorithm for automated material handling systems, based on a two-layer architecture: conveyor modules in the lower layer that represent the conveyors and logistics modules in the upper layer, which are composed out of transfer modules and are responsible for material flow control. We stated the task performed by each module type and explained their realization as part of the control algorithm. The designed modules can be combined flexibly and therefore custom-built layouts can be realized with these predefined modules. The proper function of our modularized control algorithm was demonstrated by an application example using emulated hardware.

In future projects, we plan to design additional modules, with the goal of compiling a library of the most common modules, simulate an entire material handling system with a size and complexity similar to that found in industry, and transfer the concept into PLC systems. For this transfer, the algorithm will be implemented in a PLC-compatible language. This implementation will then be used to automatically generate the control software for a demonstration with industrial conveying hardware.

References

[1] B. Rouwenhorst, B. Reuter, V. Stockrahm, G. J. van Houtum, R. J. Mantel, and W. Zijm, "Warehouse design and control: Framework and literature review," *European Journal of Operational Research*, vol. 122, no. 3, pp. 515–533, May 2000.

[2] O. Rotem-Mindali and J. W. J. Weltevreden, "Transport effects of e-commerce: What can be learned after years of research?," *Transportation*, vol. 40, no. 5, pp. 867–885, Mar. 2013.

[3] C. R. Magar, N. Jazdi, and P. Gohner, "Requirements on engineering tools for increasing reuse in industrial automation," in *Proc. 16th IEEE Int. Conf. Emerging Technol. & Factory Autom. (ETFA)*, 2011, pp. 1–7.

[4] S. H. Mayer, "Development of a completely decentralized control system for modular continuous conveyor systems," Ph. D. dissertation, Dept. Mech. Eng., KIT, Karlsruhe, 2009.

[5] S. Feldhorst, S. Libert, and M. ten Hompel, "Integration of a Legacy Automation System into a SOA for Devices," in *Proc. of 14th IEEE Int. Conf. on Emerging Technologies and Factory Autom. (ETFA)*, 2009, pp. 1-8.

[6] S. A. Bohner and S. Mohan, "Model-Based Engineering of Software: Three Productivity Perspectives," in *Proc. of the 35th IEEE Software Engineering Workshop*, 2009, pp. 35–44.

[7] 3S - Smart Software Solutions, CODESYS Application Composer. Available: www.codesys.com

[8] S. Runde, A. Heidemann, A. Fay, and P. Schmidt, "Engineering of Building Automation Systems - State-of-the-Art, Deficits, Approaches," in *Proc. 15th IEEE Int. Conf. Emerging Technol. & Factory Autom. (ETFA)*, 2010, pp. 1-8.

[9] D. Hästbacka, T. Vepsäläinen, and S. Kuikka, "Model-driven development of industrial process control applications," *J. of Syst. and Software*, vol. 84, no. 7, pp. 1100–1113, July 2011.

[10] M. Bonfè, C. Fantuzzi, and C. Secchi, "Design patterns for model-based automation software design and implementation," *Control Engineering Practice*, vol. 21, no. 11, pp. 1608–1619, Nov. 2013.

[11] L. Bassi, C. Secchi, M. Bonfe, and C. Fantuzzi, "A SysML-Based Methodology for Manufacturing Machinery Modeling and Design," *IEEE/ASME Trans. Mechatronics*, vol. 16, no. 6, pp. 1049–1062, Oct. 2011.

[12] E. Estevez, M. Marcos, I. Sarachaga, and D. Orive, "A Methodology for Multidisciplinary Modeling of Industrial Control Systems using UML," in *Proc. 5th IEEE Int. Conf. Ind. Inf.*, 2007, pp. 171–176.

[13] System Architecture for Intralogistics (SAIL), VDI/VDAM 5100, July 2011.

[14] M. Wilke, "Wandelbare automatisierte Materialflusssysteme für dynamische Produktionsstrukturen," Ph. D dissertation, Dept. Mech. Eng., TU München, 2006.

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

[16] 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*, pp. 1–9, 2006.

[17] G. Black and V. Vyatkin, "Intelligent Component-Based Automation of Baggage Handling Systems With IEC 61499," *IEEE Trans. Automat. Sci. Eng.*, vol. 7, no. 2, pp. 337–351, Apr. 2010.

[18] W. Lepuschitz, V. Jirkovsky, P. Kadera, and P. Vrba, "A Multi-Layer Approach for Failure Detection in a Manufacturing System Based on Automation Agents," in *Proc. 9th Int. Conf. on Inform. Technol.: New Generations (ITNG)*, 2012, pp. 1–6.

[19] M. Spindler, T. Aicher, B. Vogel-Heuser, and W. A. Günthner, "Efficient control software design for automated material handling systems based on a two layer architecture," in *Proc. 5th IEEE Int. Conf. on Advanced Logistics and Transport (ICALT)*, June 2016.