Production Engineering (2018) 12:297–307
https://doi.org/10.1007/s11740-018-0817-0

COMPUTER AIDED ENGINEERING

Analyzing the industrial scalability of backwards compatible intralogistics systems

Thomas Aicher[1] · **Markus Spindler**[2] · **Johannes Fottner**[2] · **Birgit Vogel-Heuser**[1]

Received: 12 September 2017 / Accepted: 2 March 2018 / Published online: 16 April 2018

Abstract
The (re-)development of industrial production systems has to deal with high flexibility due to customers' demands as well as constraints such as the dimensions of the manufactory. Therefore, Intralogistics systems, which transport goods or products between machine tools in the production system, must also be flexibly assembled to fit into the given space. This presupposes that parts of the Intralogistics system, such as a belt conveyor, may be modified or exchanged by different ones and with less effort. Additionally, due to the different lifespans of mechanical, electrical and software components, often electrical devices need to be replaced by other ones during operation as a result of lacking availability thanks to shorter life cycles. Nowadays in industry, a manual adaptation of the control software is necessary after the exchange of an electrical/mechanical device with a newer one. In order to ease the software adaptation, in this paper an automatic analysis of the differences between incompatible software and an adaptation approach focusing on the functional behavior of the software is introduced. After identifying seven interaction points of the control software that need to be adapted, the approach is evaluated with an industrial case study and feedback from industrial experts to prove industrial scalability.

Keywords Model-driven software engineering · Automated production systems · Intralogistics system · Software difference analysis

1 Introduction

Due to their high handling capacity, short throughput times, and good process quality, automated material handling systems—often called Intralogistics systems (IntraS's)—are widely used in industry to transport and organize the material flow between sources and destinations within automated production systems [1]. To fully utilize the potential of IntraS's and to meet the demands of a changing production environment [2], an IntraS, e.g. a pallet- or tote-conveying system, has to be custom-built as a result of the dimensions of the system requiring high flexibility. Moreover, the different lifespans of mechanical, electrical and software components and their lacking availability because of shorter life cycles often require electrical devices to be replaced by other ones during operation [3].

Diverse engineering disciplines, e.g. mechanical, electrical/electronic and software engineering, are involved in the development of IntraS's [3]. Not only the development of a new IntraS but also the modification of an existing one, e.g. to extend, reduce or modify parts of the IntraS during its life cycle, requires (re-)engineering provided by all of these disciplines, especially software engineering. The challenges of today's control software are a strong interconnection, on the one hand, and high variability resulting from the customer-specific requirements, on the other [4]. These challenges increase the amount of effort as well as the propensity for errors for the required software (re-)engineering after the exchange of a mechanical

✉ Thomas Aicher
aicher@ais.mw.tum.de

Markus Spindler
spindler@fml.mw.tum.de

Johannes Fottner
fottner@fml.mw.tum.de

Birgit Vogel-Heuser
vogel-heuser@tum.de

[1] Automation and Information Systems (AIS), Technical University of Munich (TUM), Garching near Munich, Germany

[2] Materials Handling, Material Flow, Logistics, Technical University of Munich (TUM), Garching near Munich, Germany

and/or electrical device. Two different variants of a motor, the first one using three Boolean signals (bit coded) as an output signal and the second one using an incremental encoder that delivers an Integer signal as output serve as two examples. In this paper, software modules consisting of similar but not identical functionalities are considered variants, whereas a module, e.g. Module A, that uses newer, state-of-the-art technologies than an older module, e.g. Module B, and covers, extends, or in some cases slightly reduces its functionality can be considered a higher version of the older module, e.g. Module B.

An analysis of such application control software in 16 industrial companies revealed that due to the modification of an electrical/mechanical device, the control software often has to be modified in different lines or parts of the control software as a result of a lack of flexible modularity [5]. In order to improve reusability and minimize the interconnection of the control software, a modular software architecture for IntraS was developed in prior work [6]. To improve the compatibility of the different software modules, (pre)defined interfaces were developed. However, in the case of a software module modification, usually not only the interfaces but also the functional behavior must be (re-)developed, e.g. with different or further states or transitions, in order to support the newly added functionality. Consequently, the number of incompatible software modules and interfaces in the module library increases, which complicates the assembly of the application software. Therefore, the evolved interfaces and functional behavior of the exchanged module and its neighboring modules have to be analyzed by the software application engineer. After that, the software engineers can adapt, i.e. wrap, the exchanged module in order to become compatible with the neighbors. This procedure is applied manually by the software engineer and thus increases the engineering effort and propensity for errors.

In order to begin tackling these drawbacks and to start automatizing this procedure, a Model-Driven Engineering (MDE) approach has been developed. Applying simple lab size demonstrators at the institute, the feasibility of the approach focusing on modules' interfaces could be revealed in prior work [6]. After the interfaces were analyzed and adapted automatically on a model level, IEC 61131-3-compliant control software is generated for Programmable Logic Controllers (PLC). However, since usually the functional behavior is also affected when exchanging a module, industrial availability is not supported yet. The main contribution of this paper is, firstly, to include functional behavior and, secondly, to support the industrial applicability of the MDE approach. Therefore, a prototypical implementation of the approach has been developed. For two industrial applications, a feasibility study using a real IntraS testbed as well as a questionnaire with different experts from industry have been conducted.

The remainder of this paper is structured as follows: In Sect. 2, related work in the area of the model-driven development of the automation software of IntraS's and an analysis and adaptation of software differences will be introduced. The concept of automatic functional behavior adaptation and a real world industrial application example are introduced in Sect. 3. A feasibility study using a real IntraS and the results of expert workshops are shown in Sect. 4.

2 Related work

First, existing model-driven software approaches for the control of IntraS's are introduced and, second, approaches to the model-driven analysis of differences between two software modules and how to adapt them are discussed as the key approach of this concept.

2.1 Model-driven engineering and open control software

In prior works [6], a meta model was introduced that enables the encapsulated description of software modules. PLCs are currently and, at least for the next 10 years, will be used for automation hardware control [7]. To define the software hierarchy, the system architecture for Intralogistics (SAIL) is applied, which is an architectural model used to describe control software for IntraS's as well as their basic functions and principle interfaces [8]. The highest hierarchy level of SAIL describes a whole facility area and is called a conveying area. In industry, usually every conveying area is controlled by one PLC, which contains a local material flow controller to transport the transport units (TU), e.g. pallets or totes, through the respective area [9]. The remaining hierarchy level of SAIL for the area includes a conveying segment, a group and at least one element, thus representing a software module. Based on these models, the control software of one module in different variants or versions can be transferred into a predefined description language and thus compared with each other for the classification of differences, e.g. introduced by Vogel-Heuser et al. [10] (see Fig. 1). As Rutle et al. [11] suggest, a modelling language for wrappers has been developed based on the UML class diagram to represent these simple model differences, not covering the behavioral aspects, scalability, and feasibility studies with real-world applications [12].

Zäh and Pörnbacher introduce an MDE approach to the development of PLC software for machine tools [13]. Therein, a modelling language for the control software and machine tools are introduced based on Unified Modeling Language (UML) and connected with each other using (pre) defined interfaces. After that, executable code for a machine tool's PLC is generated. However, the authors assume that

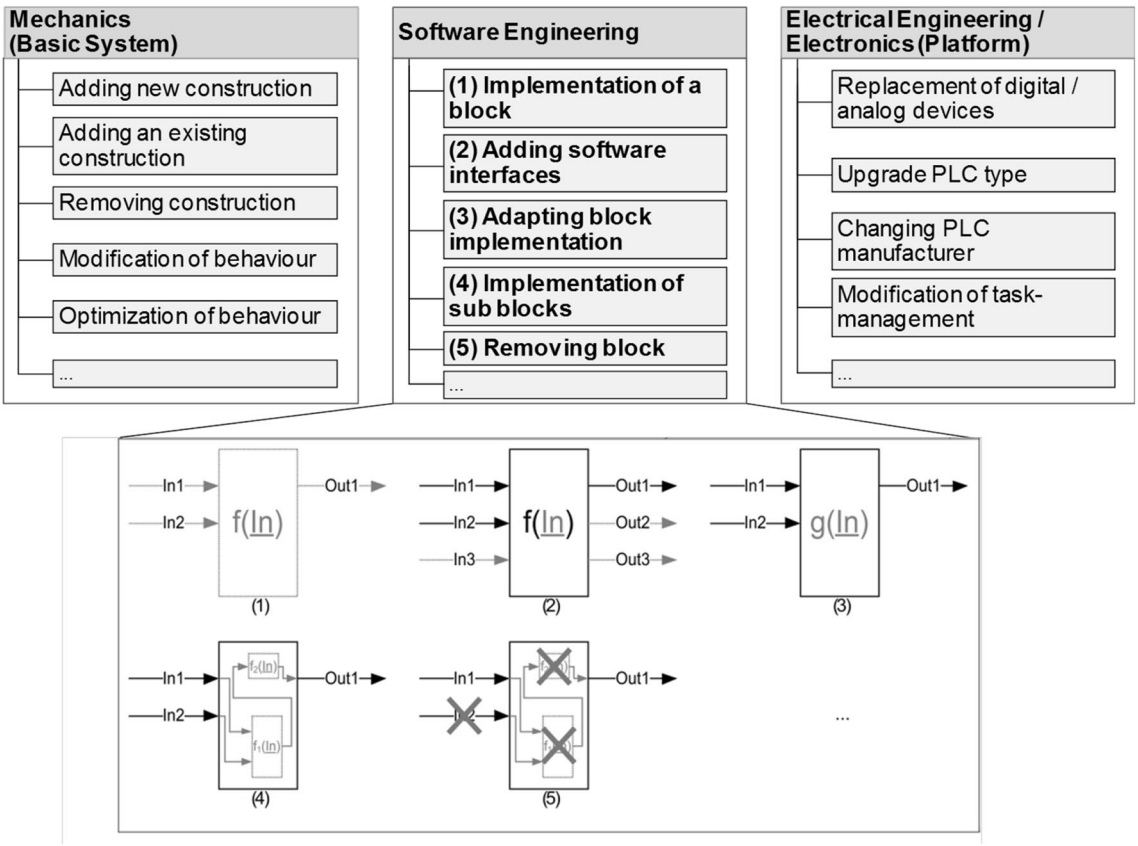

Fig. 1 Classification of variants by Vogel-Heuser et al. [10]

all interfaces are compatible with each other and thus do not address software adaptations.

Krüger et al. [14] show a survey analyzing the research and application of encapsulated control software for entities, i.e. Cyber-Physical Systems (CPS), that can collaborate with other ones and are in intensive connection with the surrounding physical world and its ongoing processes. In this way, a module, e.g. a roller conveyor, would consist of standardized interfaces, for instance, using an administration shell, which eases the exchange or modification of a module. However, apart from a few exceptions, centralized and hierarchical control systems using a PLC are still state-of-the-art in IntraS's [9].

Schlechtendahl et al. [15] present a concept of Industry 4.0 interfaces and how production systems can be identified and included into an Industry 4.0 environment, even though an Industry 4.0 interface was not regarded during the manufacturing of the system. Therefore, a communication gateway and information server was developed focusing on the communication between encapsulated machine tools, which consist of their own control system. An approach to applying this Industry 4.0 interface to integrate, for instance, electrical/mechanical sub-devices into a machine tool that automation software has to run on the same control hardware as the machine tool is not addressed, however, required to apply this approach for today's centralized controlled IntraS.

Brecher et al. present a survey examining the application of open control systems in industry and introduce the Open System Architecture for Controls with Automation Systems (OSACA) [16, 17], which is focused on the human-machine-interface. OSACA was developed as an application program serving as a middleware and supports the definition of encapsulated functions, or so-called architecture objects. For the communication between these objects, different protocols such as OLE for Process Control (OPC), are applied. However, due to the high throughput times of the goods in IntraS's, the control software has to be implemented on a PLC and thus the approach has to be translated into the programming language IEC 61131-3 that includes a high interconnection of the modules, which is contrary to OSACA.

2.2 Model-driven analysis and adaptation of software differences

Model-driven approaches of software adaption are common methodologies for developing automated systems in different domains. In this section, different approaches are analyzed in order to consider existing research results.

Kelter et al. present model-driven approaches and tools for analyzing and adapting differences, i.e. model information which are in one of the analyzed software modules and not in the other ones, based on UML [18]. UML is frequently used to describe models for software systems and to apply approaches to computing differences [19]. To also cover engineering information, an extension of a subset of the UML, so-called SysML, was developed. In this paper, the state chart diagram of SysML is used to define the different functional behaviors of the modules' control software.

Westfechtel et al. present a formal approach to both the two- and three-way merging of models in the well-known Eclipse Modeling Framework (EMF) [20], an Eclipse-based modeling framework with a code generation facility for building tools based on a structured data model. However, the classification and adaptation of detected differences is missing. Based on the proven implementation of the approach in Eclipse, the same tool was chosen for the application.

Roy and Cordy [21] survey the state of the art in clone detection research, which is focused on detecting and removing clones from software systems as well as maintaining clones in their evolution life cycle. Summarized, the authors reveal that several techniques, approaches and tools from the domain of computer science exist that deal with the requirement to analyze and compare control software and identify differences. These works could provide a basis for the industrial application that improves the exchange of modules. However, the transfer of these approaches in industrial domains such as automation engineering including different requirements compared to computer science is not addressed.

Schäfer et al. present a delta-oriented programming language to provide more flexibility for the implementation of Software Product Lines in mechatronics [22]. Based on a delta-oriented programming language, a more flexible and modular implementation of product variability starting from different core products is possible. Delta modules specify the changes to be applied to the core module, which is comparable with wrapper functions. However, to ease the software engineering, the corresponded delta module (wrapper function) should be identified and applied automatically depending on the neighboring modules presupposing an analysis and adaption of the core modules. Additionally, a modelling language to define the structure and behavior of differences according to [11] is not addressed by the authors.

3 Backwards compatibility of software: concept

To illustrate the challenges of exchanging a software module with a newer version or different variant, an application example and an analysis of the potentially concerned functional behavior of software modules are presented. After that, the general procedure of the approach to backwards compatibility and the application are shown.

3.1 Introduction of an application example

To prove the industrial application of the approach, an analysis of IntraS's with experts from a leading supply company was conducted, and a typical use case has been derived in an iterative process (Fig. 2).

In the following, a software module, i.e. electric motor driving the rolls of a roller conveyor, is considered in two different variants. In variant one (VA1), the speed of the motor is bit coded, which means that three Boolean variables

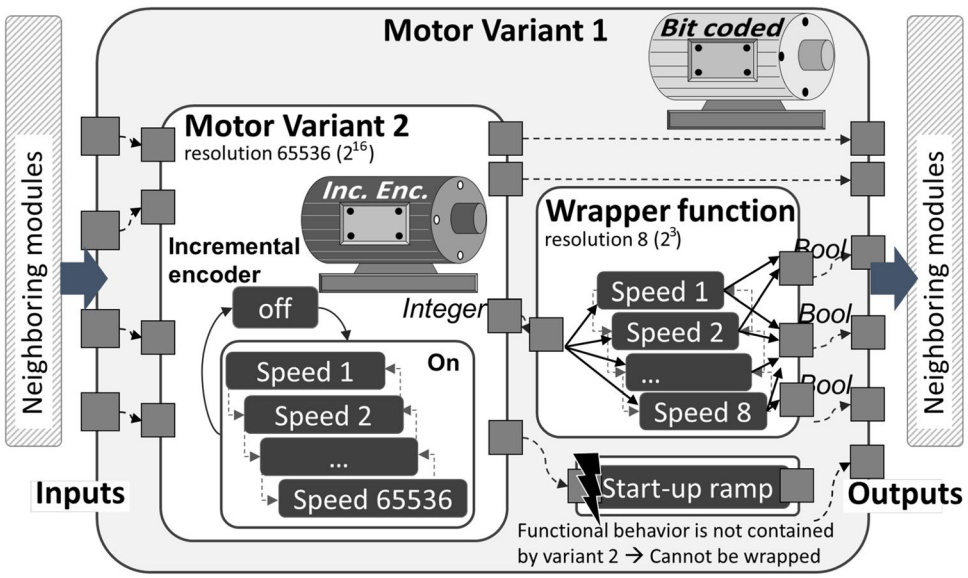

Fig. 2 Exchanging a bit-coded motor with a motor containing an incremental encoder

(representing 2^3 possible values) are used to transfer a decimal number into binary, similarly with a binary code. Thus, the automation software has to set the speed of the motor by using these three Boolean variables. In variant two (VA2), an incremental encoder is used that can represent the speed of the motor by one integer variable (representing 2^{16} possible values). Therefore, a behavioral adaptation, i.e. wrapper function, is defined to transfer the different speed states of both variants to allow for the compatibility of the motor in VA2. However, due to the nature of the subject, wrappers are not able to provide software modules with new functional behaviors that were not considered during the engineering. Hence, the software module, e.g. module VA2, which wants to imitate the functional behavior of another module, e.g. module VA1, needs at least the same or an extension of VA1's functional behavior. In case, VA1 consists of more functional behaviors than VA2, e.g. a start-up ramp (see Fig. 2), no software compatibility of VA2 based on wrapper functions is feasible.

3.2 Scalability of the approach

In order to examine the industrial applicability of the wrapper, the so-called interaction points [23], which can interact with different parts of the control software based on the calling of the module or reading and writing of the shared memory, have to be examined. The goal is to ensure the application of the approach not only on the interface to neighboring modules, but also to further interaction points and the behavior description of a module. Because object-oriented programming languages are not yet widespread in industry, the communication with neighboring modules is usually implemented with a common data set stored in an open Data Block, e.g. supported by the company Siemens [24], which can be written and read by both modules.

To transfer TU to the next conveying area, the handshake between the different PLCs is implemented by the global material flow controller. Furthermore, the interaction points of the software modules at the higher control level, i.e. conveying group, the hardware configuration and the Organization Blocks need to be considered (see Fig. 3). The Global Variable List (variable table), which defines global variables and can be monitored and forced by the application engineer, is an additional interaction point. Summarized, seven interaction points of a software module could be revealed in general. However, the interaction with the hardware configuration and Organization Blocks will not be considered due to the direct interaction of the software modules. Thus, the automatic analysis and adaptation focuses on the interaction with the higher control level, neighboring modules, Global Variable List, Data Blocks and the local material flow controller.

3.3 General procedure for analyzing and adapting software differences

The approach for analyzing and adapting software differences is divided into different phases. At first, the control software is transferred into a model description based on a text-to-model transformation (T2M), to represent the control code in a more formalized way. Hence, the meta model of software modules [6] was extended by the SysML state machine diagram to also represent functional behavior. After that, a model mapping and differencing process is applied to detect all elements that are similar or different in both models, respectively (see Fig. 4). Using a difference metamodel, e.g. the EMF difference meta model, the models are in general minimalistic, transformative, compositional, and typically symmetric, i.e. given a difference representation the inverse of it can be computed [11, 25]. Hence, in this

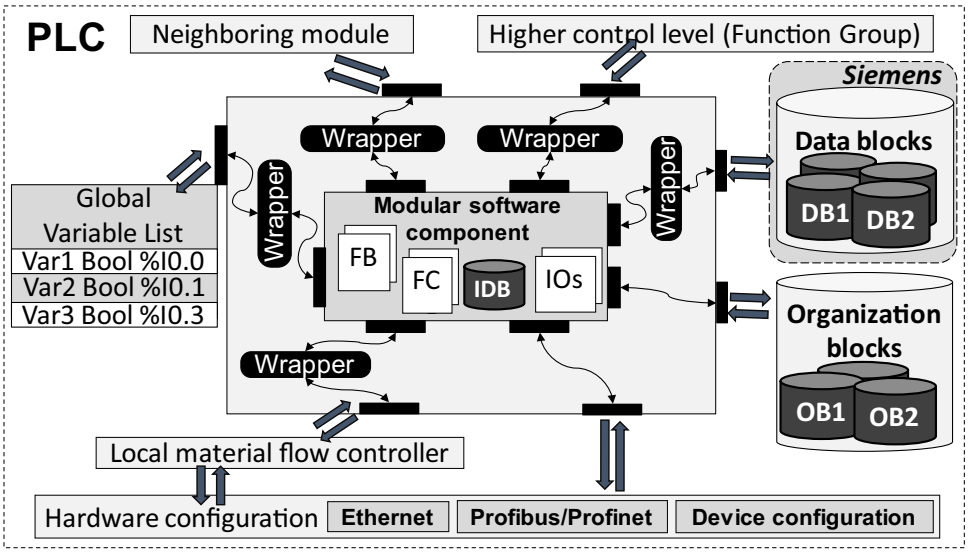

Fig. 3 Interaction points of a software module in general

Fig. 4 General procedure of the difference analysis modeled with event-driven process chain

paper the approach of symmetric differences is applied, to calculate the intersection of two models.

The symmetric differencing process analyzes the mapping result and determines, based on a Global Compatibility List (GCL), which can be stored in an Excel sheet or data base, whether the two models are equal or differ in some parts, e.g. the speed states of both motors are unequal.

To adapt the functional behavior of motor VA2, in order to possess the same functional behavior of motor VA1, a corresponding pre-developed wrapper function has to be

applied based on the identified differences between both models. The resulting wrapper function is applied automatically and exported to the application engineer. In the event any additional modifications are required, the application engineer can modify the wrapper functions via a user interface. Finally, IEC 61131-3-compliant control code is generated based on a model-to-text (M2T) transformation. The development of the required software modules, wrapper functions and GCL is the task of the software engineers. In order to define the functional behavior of a wrapper function, the wrapper's meta model [12] was also extended by a state machine diagram.

3.4 Application of the approach of backwards compatible software modules

In order to apply the wrapper functions, a wrapper editor was developed that describes the graphical user interface. In this interface, users can define the differences between two models, i.e. queries, and corresponding wrapper functions as well as apply these to adapt the interface or behavior of a software module. The queries can be developed by defining an Object Constraint Language (OCL) expression that describes the respective model difference and the choice of both a query library to store the query and wrapper library to store the wrapper function. To reference an OCL query to a respective wrapper function, links between the query library and wrapper library can be defined by the user (see Fig. 5). For the development and application of wrapper functions, a modeling diagram is demanded that clarifies the relationship between one variable and another, e.g. using mathematical relationships. Thus, a diagram following the syntax and semantics of the standardized and proven SysML Parametric Diagram was developed that enables the representation of the mathematical relationships of different variables or constraints (see Fig. 5). The relationship between two variables can be described by a Constraint Property, which is defined within a Constraint Block.

As soon as the queries and wrapper functions are developed, their application is executed based on the SysML Parametric Diagram. Therein, the wrapped source module, i.e. the module that should be adapted, and the target module, i.e. the module that consists of the demanded functional behavior, are figured. Since the interface and functional behavior of the source module should be hidden and not visible to the outside environment, e.g. neighboring modules or the conveying group, it is allocated in the target module. After the analysis of differences as well as the subsequent classification of the encountered differences and interaction points using the (pre)defined queries, the wrapper functions can be applied automatically. In the case of the detection of a new model difference that has not yet been classified, manual modifications or extensions are feasible in the editor. Variables that are not essential to the imitation of the interfaces or functional behavior of the target module can be terminated by linking them with an End Block. A prototype providing a modeling environment has been developed based on Eclipse and EMF.

4 Evaluation

In the following section, the structure and the functionality of an industrial application example "Self-X Material Flow Demonstrator" are introduced. In Sect. 4.2, a feasibility analysis regarding backwards-compatible software modules focused on the functional behavior is shown. In order to prove the industrial application of the approaches, the feedback from different industry experts and a paper survey are presented in Sect. 4.3.

4.1 Introduction of the Self-X material flow demonstrator and application example

In order to evaluate the industrial application of the approach, the Self-X Material Flow Demonstrator of the institute Automation and Information Systems (AIS) was used. The tracks of the demonstrator are divided into separately controllable conveying groups and can be actuated electrically, thus allowing for the individual control of each roller conveyor (see Fig. 6). 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 SoftPLC or Siemens S7-1500 is feasible. The demonstrator contains three conveying elements (roller conveyor (curve), live roller conveyor, and belt diverter) and three conveying groups (accumulation conveying, T-junction (merging) and T-junction (diverting)).

As an application example of exchanging a software module with a different variant, the conveying element B4 as part of the conveying group "diverting" is considered (see Fig. 6). In the first variant (VA1), the totes are diverted by a fixed ratio 1:1, i.e. totes are transported ahead or downward alternately. In the second variant (VA2), the application engineer can parameterize the ratio n:m, e.g. 3:2, with the result that three totes are transported ahead and two downward. In VA1, B4 consists of the Boolean variables *conveyor_free* and *conveyor_occupied* as output signals. Using the state machine diagram, B4's functional behavior can be described in three states, where the values of the variables are set. In VA2, B4 contains the Integer variables *conveyor_ahead_setpoint, conveyor_backward_set point*, *conveyor_ahead_actual_value* and *conveyor_ backward_ actual_value* as output signals in order to parameterize the tote ratio freely. The values of these variables are also set in

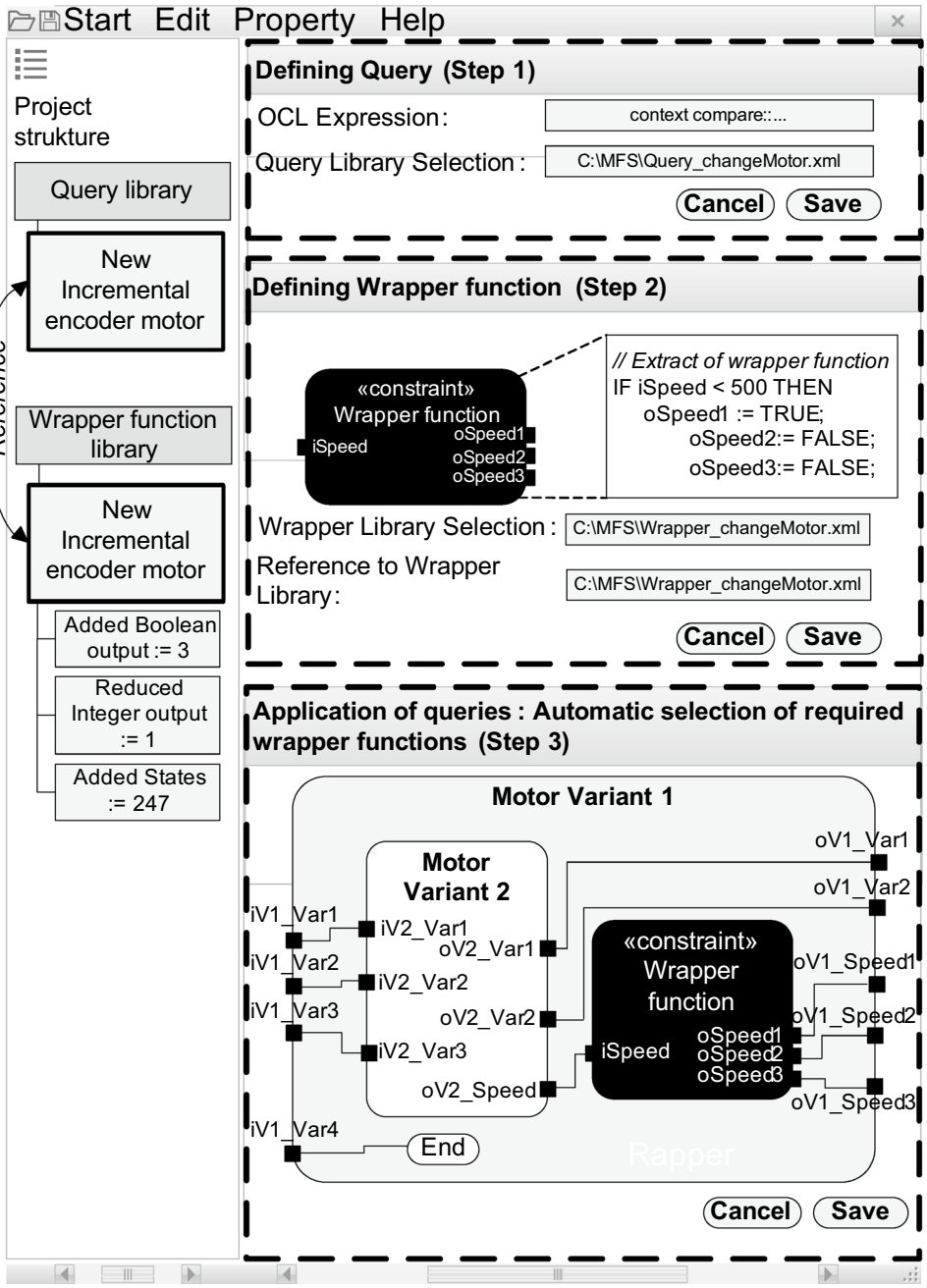

Fig. 5 Conceptual design of the application

three states, but the transition conditions between the states (see C5–C8 in Fig. 6) and the executed actions within the states are different. Since the neighboring software modules B5 and C1 expect Boolean values, the output signals of VA2 have to be adapted. To hand over a TU correctly, the modules in both variants further contain the variables *hand_over* and *transferred* to initialize that a new TU should be handed over to the next module or was already transferred to the next module, respectively. The routing of the TU is determined by the conveying group "diverting." In order to involve the different interaction points of the software modules, the communication between the modules is realized directly between the modules as well as by using the Global Variable List and a global Data Block.

4.2 Industrial case study: exchange of the T-junction ratio

To identify the exchange of the T-junction ratio, an OCL expression that describes the exchange of the motor, e.g. based on adding four integer variables, reducing two Boolean variables and changing four transitions, is defined in the following:

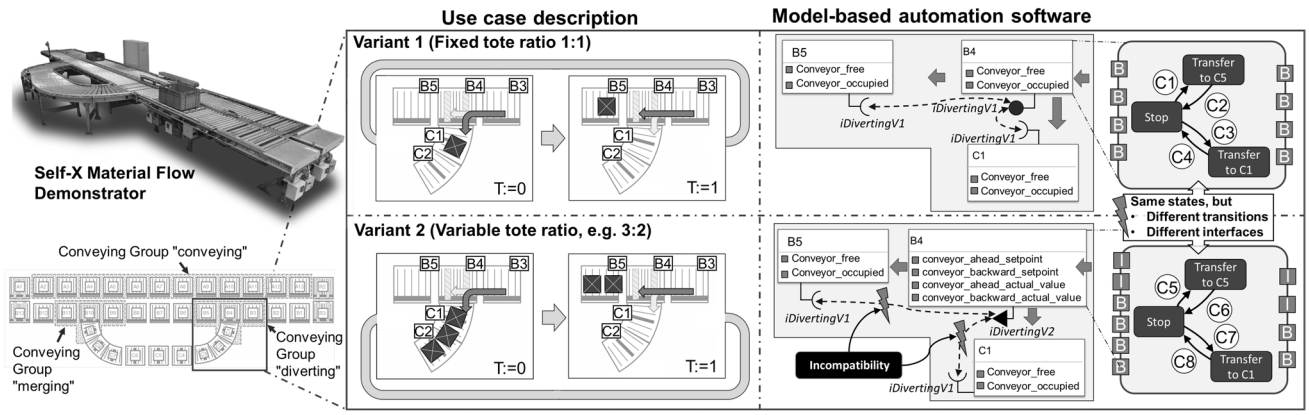

Fig. 6 Introduction of the application example "Self-X material flow demonstrator"

context compare::ModulComparatorself.compare(self.sourceModul,self.targetModul).getVariableDiffs()->select(p|p.isAdd() and p.hasType(step7Datatypes:: INT))->size() = 4 and self.compare(self.sourceModul, self.target Modul).getVariableDiffs() ->select(p|p.is Delete() and p.hasType(step7Datatypes:: BOOL))-> size() = 2 and self.compare(self.sourceModul, self. targetModul).getTransitionDiffs() > exists(p|p.isChange()) = 4.

In the case in which an OCL expression is necessary in more or less detail, the expression can be modified. In the next step, wrapper functions that include the required equations to transfer the different ratios of the T-junction are added. For instance, the following functional behavior is defined for a wrapper function that adapts the interface to the neighboring module: In the case in which a TU has been successfully handed over to one neighboring module, e.g. B5, the variable *conveyor_ahead_actual_value* is set to a value equal to the variable *conveyor_ahead_setpoint*, so that the next TU will be transported to the second neighboring module, e.g. C1. After that, during the next TU is handed over to the second neighboring module, e.g. C1, the variable *conveyor_ahead_actual_value* of B5 is set to a value not equal to the variable *conveyor_ahead_setpoint* and so forth. A similar functional behavior is defined for the second neighboring module, i.e. C1. The association of the query with the respective wrapper function can be achieved using the wrapper editor. After the differences could be adapted on the model level, the control software compliant with IEC 61131-3 is generated automatically based on a M2T transformation (see Fig. 7).

4.3 Expert evaluation

In order to prove the industrial application of the approaches, two workshops with different experts from the industry who are involved in the systems' development, i.e. software

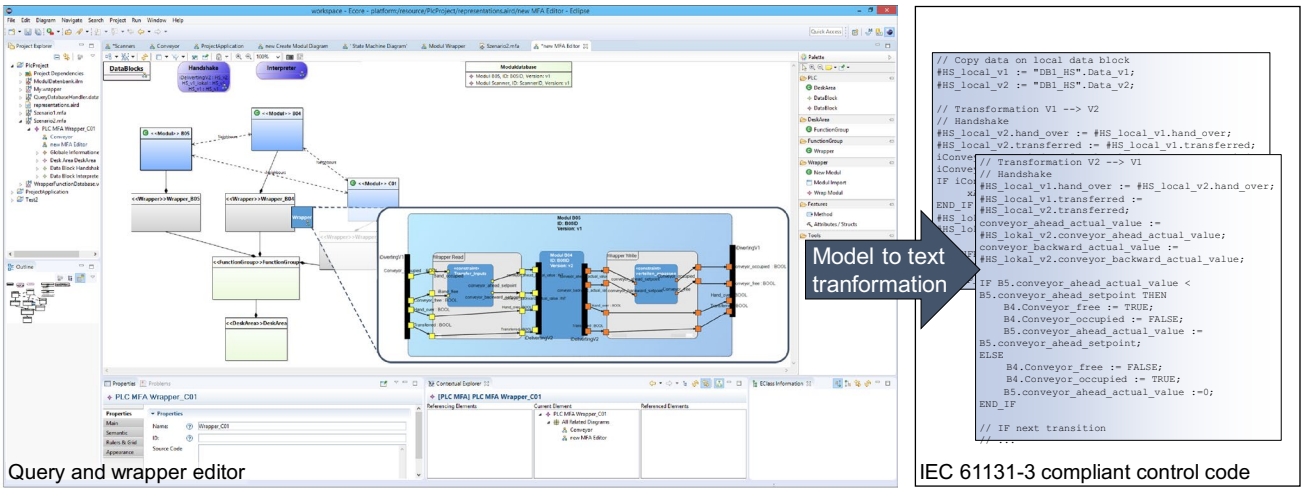

Fig. 7 Application of the prototypical implementation to analyze and adapt differences automatically

developers and application engineers, were conducted. In the first workshop, three experts who work for a worldwide leading provider of intralogistics products and systems (Group 1) participated. The second workshop was conducted with one expert who works for a provider and builder of different presses (Group 2). During the workshops, the approach of backwards-compatible software modules, including two use cases, i.e. the exchange of a barcode scanner with a QR-Code scanner [12] and the adaptation of the diverting ratio of a T-junction (see Sect. 4.1), was presented in order to provide a profound understanding of the approach.

Both user groups confirmed that the level of functional behavior and number of interface variables of a newer module are higher than in the older one. The experts assumed that around 80% of newer modules contain at least the same interface and functional behavior as the older module. In this case, the interface and functional behavior will be expanded, but not reduced. That is a precondition to automatically apply backwards compatibility on modules with different versions or variants. Additionally, the experts have confirmed that the effort to develop and introduce an initial GCL is low because currently all modules can interact with each other. In the case in which a module is developed in the future that is incompatible with other modules, a wrapper has to be developed that fills the GCL automatically. This fact increases the industrial applicability of the approach. During the discussion with the experts, it was suggested that after the wrapper was developed and applied for the one specific use case, a further user interface is necessary, in order to choose additional detected differences.

Next to the discussion with the experts, a paper survey was conducted after a workshop with group 1 (see Fig. 8). The presented values are calculated using the arithmetic average of the experts' surveys. Since three experts are not a sufficient amount of people to make a qualitative statement, further stochastic values such as the variance and standard deviation are not shown. The introduced approach is compared with the experts' current approach in the company. The systematic analysis and classification of differences as well as the error-proneness for adapting the interfaces are satisfied by using the mentioned approach. Since the wrapper has to be developed manually before the interface can be adapted, the initial effort for the development of the wrapper was evaluated slightly better compared to the experts' current approach. However, the approach introduced in this paper allows for the systematic analysis and adaptation of differences and increases the degree of automation and reusability. Moreover, the error-proneness can be reduced significantly. Thus, it can be concluded that the introduction of wrappers in the development of control software for IntraS's would improve the software engineering. That is amplified by the fact that all experts plan to introduce the model-driven wrapper approach in their company in the future.

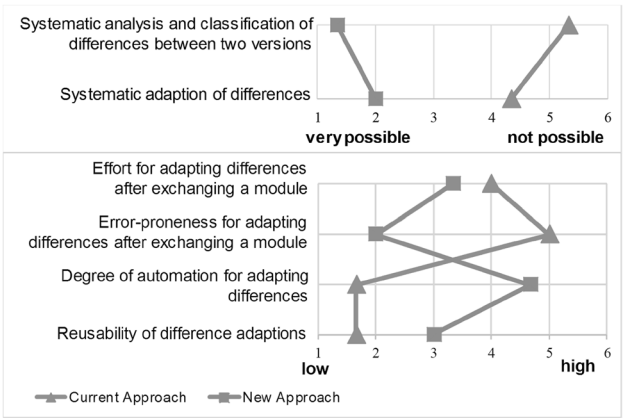

Fig. 8 Evaluation of backwards compatible modules

5 Conclusion and outlook

Since intralogistics systems are developed in a highly specialized way and are custom-built to meet the demanded requirements, a reusable and flexibly adaptable structure of the electrical/mechanical devices as well as the control software is required. Hence, considering the software engineering involved in the development of intralogistics systems, the automation software should be based on a reusable and flexibly adaptable, i.e. modular, structure. Moreover, to improve the flexibility, especially the exchange of a module with a similar module, i.e. a different variant or version, should not call for any engineering effort. Instead, the backwards compatibility of modules, including not only the interface but also the behavioral adaptation that supports the exchange of modules offering similar logistics functionalities in different evolution steps, i.e. versions, is required. In this paper, an approach of model-driven analysis and the adaption of software modules for intralogistics systems focused on the behavior functions was presented. Hence, the definition of consistent interfaces and functional behaviors between different modules and IEC 61131-3-compliant code generation to deploy the logical functions on different field devices, e.g. PLC or industrial computers, could be applied. The evaluations, which include a real industrial intralogistics system and workshops with different experts from the industry, confirmed that the industrial application of the approach is feasible and reveals its benefits compared to today's applied methods in industry.

Future work will examine further more complex use cases by industrial systems. To detect and measure the engineering effort involved in automatically analyzing and adapting intralogistics systems compared to today's common approaches, a paper survey with more participants should be

proposed. The application will be extended by a further user interface in order to choose additional detected differences.

Acknowledgements We thank the Bavarian Research Foundation (BFS) for funding this work as part of the collaborative research project 'aComA–Automatic code generation of modular-based material flow systems'.

References

1. Rouwenhorst B, Reuter B, Stockrahm V, van Houtum GJ, Mantel RJ, Zijm WHM (2000) Warehouse design and control: Framework and literature review. Eur J Oper Res 122(3):515–533
2. Morel G, Pereira CE (2007) Manufacturing plant control: Challenges and issues. Control Eng Pract 15(11):1319–1320
3. Vogel-Heuser B, Fay A, Schaefer I, Tichy M (2015) Evolution of software in automated production systems: challenges and research directions. J Syst Softw 110:54–84. https://doi.org/10.1016/j.jss.2015.08.026
4. Vogel-Heuser B et al (2014) Challenges for software engineering in automation. J Softw Eng Appl 7(5):440–451
5. Vogel-Heuser B, Fischer J, Feldmann S, Ulewicz S, Rösch S (2017) Modularity and architecture of PLC-based software for automated production systems: an analysis in industrial companies. J Syst Softw 131:35–62. https://doi.org/10.1016/j.jss.2017.05.051
6. Aicher T, Regulin D, Schütz D, Lieberoth-Leden C, Spindler M, Günthner WA, Vogel-Heuser B (2016) Increasing flexibility of modular automated material flow systems: a meta model architecture. IFAC-PapersOnLine 49(12):1543–1548. https://doi.org/10.1016/j.ifacol.2016.07.799
7. ARC Advisory Group (2015) PLC and PLC based market research study. In: PLC & PLC-based PAC worldwide outlook: five year market analysis and technology forecast through 2020
8. VDI/VDMA (2011) System architecture for intralogistics (SAIL) fundamentals. VDI Verlag GmbH, Berlin
9. Clemens N (2010) "Materialflusssteuerung heute und ihre Defizite. In: Günthner W, ten Hompel M (eds) Internet der Dinge in der Intralogistik. Springer, Berlin, pp 15–21
10. Vogel-Heuser B, Folmer J, Legat C (2013) Anforderungen an die Softwareevolution in der Automatisierung des Maschinen- und Anlagenbaus. Automatisierungstechnik 63(3):163–174
11. Rutle A, Rossini A, Lamo Y, Wolter U (2009) A category-theoretical approach to the formalisation of version control in MDE. In: Lecture notes in computer science (including subseries lecture notes in artificial intelligence and lecture notes in bioinformatics), vol 5503, pp 64–78
12. Aicher T, Schütz D, Spindler M, Liu S, Günthner WA, Vogel-Heuser B (2017) Automatic analysis and adaption of the interface of automated material flow systems to improve backwards compatibility. IFAC-PapersOnLine 50(1):1217–1224. https://doi.org/10.1016/j.ifacol.2017.08.345
13. Zäh MF, Pörnbacher C (2008) Model-driven development of PLC software for machine tools. Prod Eng Res Dev 2(1):39–46
14. Krüger J et al (2017) Innovative control of assembly systems and lines. CIRP Ann Manufac Technol 66(2):707–730
15. Schlechtendahl J, Keinert M, Kretschmer F, Lechler A, Verl A (2014) Making existing production systems Industry 4.0-ready: Holistic approach to the integration of existing production systems in Industry 4.0 environments. Prod Eng 9(1):143–148
16. Brecher C, Verl A, Lechler A, Servos M (2010) Open control systems: State of the art. Prod Eng 4(2):247–254
17. Brecher C, Kolster D, Herfs W, Plessow M, Jensen S (2012) Plug and play device integration for industrial automation with instant machine visualization using RFID technology. Prod Eng 6(2):179–186
18. Kehrer T, Kelter U, Pietsch P, Schmidt M (2012) Adaptability of model comparison tools. In: Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering—ASE 2012, Essen, Germany. ACM, New York, NY, USA, pp 306–309. https://doi.org/10.1145/2351676.2351731
19. Kelter U, Wehren J, Niere J (2005) A generic difference algorithm for UML models. Softw Eng 64(105–116):4–9
20. Schäfer I, Bettini L, Bono V, Damiani F, Tanzarella N (2010) Delta-oriented programming of software product lines. In: Bosch J, Lee J (eds) Software product lines: going beyond. SPLC 2010. Lecture notes in computer science, vol 6287. Springer, Berlin, Heidelberg
21. Roy CK, Cordy JR (2007) A survey on software clone detection research. Queen's School Comput TR 115:115
22. Schäfer I, Bettini L, Bono V, Damiani F, Tanzarella N (2010) Delta-oriented programming of software product lines. In: International Conference on Software Product Lines, Springer, pp 77–91
23. Vogel-Heuser B, Schütz D, Frank T, Legat C (2014) Model-driven engineering of manufacturing automation software projects—a SysML-based approach. Mechatronics 24(7):883–897
24. Siemens (2014) Programming Guideline for S7-1200/S7-1500. http://www1.siemens.cz/ad/current/content/data_files/automatizacni_systemy/mikrosystemy/simatic_s71200/programmingguideline-for-s71200-s71500_2014-09_en.pdf
25. Sabrina F, Westfechtel B (1998) Differencing and merging of software diagrams—state of the art and challenges. In: 2nd International conference on software and data technologies, pp 90–99