Executing Model-based Software Development for Embedded I4.0 Devices Properly

Fabian Burzlaff [1], Christian Bartelt [2] and Steffen Jacobs [3]

Abstract: Technical interoperability in "Industrie 4.0" scenarios is currently being achieved by standards such as OPC UA. Such standards allow operators to use a common communication interface for heterogeneous production devices. However, production flexibility (e.g. self-configuration or dynamic self-adaptation) can only be achieved if system structure and engineering processes change. At the moment, traditional engineering processes for embedded systems generate communication interfaces from software. This stands in stark contrast to component-based software engineering approaches. In this paper, we introduce a tool-based software engineering approach that puts models back at the core of embedded system development. This approach enables flexible production scenarios by bringing together process-oriented software development and operator-oriented interface construction.

Keywords: Software Engineering Processes; Component-based Software Development; Knowledge-driven Architecture Composition; Self-Adaptive Systems; Industrial Internet of Things

1 Motivation and Problem Statement

As postulated by the RAMI 4.0[4], future manufacturing lines consist of I4.0-Components. These I4.0-Components require transactions during their development phase that externalize program-specific control variables for standardized communication efforts. Current engineering approaches focus on generating the required communication layer out of code written in various programming languages. However, this bottom-up approach cannot deal efficiently with information integration issues (e.g. incompatible data types or different data semantics). As a consequence, a system integrator must manually adapt software components of embedded or cyber-physical systems [Le08] instead of their corresponding models. Hence, they are mainly driven by individual interface-adaptations without following a sophisticated software engineering process. Abstractly speaking, the classical model-based software development paradigm is turned inside out. This means that interface models for communication purposes between embedded systems are generated from code instead of the other way around.

[1] University of Mannheim, Institute for Enterprise Systems, L15 1-6, Mannheim, 68159, burzlaff@es.uni-mannheim.de
[2] University of Mannheim, Institute for Enterprise Systems, L15 1-6, Mannheim, 68159, bartelt@es.uni-mannheim.de
[3] University of Mannheim, L15 1-6, Mannheim, 68159
[4] https://www.zvei.org/themen/industrie-40/das-referenzarchitekturmodell-rami-40-und-die-industrie-40-komponente/

Amalgamating software components into a cohesive system architecture is an everlasting engineering problem. To ease the integration burden, a lot of successful standards have been produced in various application fields. For example, the eCl@ss[5] standard enables a company and its suppliers to communicate data in a clear and concise format. Although such standards facilitate an automated data exchange across company boundaries, the required manual work for implementing such standards is high. Furthermore, these standards mostly define technical and syntactical decisions made by a group of stakeholders at a specific point of time although the semantics of data may vary. Overall, this standard reduces manual data transformation and enables production automation as site operatory actively participate in building this standard. Still, this standard requires predefined rules that transform proprietary data into eCl@ss adequate information. Then, a communication between programmable logic controllers (PLC) that support this standard is possible by using a supported communication protocol (e.g. PROFIBUS[6]).

Another example is the AUTOSAR[7] platform standard. In contrast to eCl@ass, this standard showed in the automotive domain that formalizing information semantics is possible. By providing a standardized runtime solution to embedded system developers, the amount of manual integration effort is reduced to a minimum. Sadly, this approach seems to be only feasible in a market that is dominated by a few players. This is due to the circumstance, that all cooperating players have to agree on one common truth at a time (i.e. technical, syntactical and semantic information characteristics). Here, no transformation rules are necessary, as program logic must be refactored completely. This code must comply with the AUTOSAR runtime environment that then handles the communication of different components (e.g. electronic control units (ECU)).

Currently, there are various research efforts that try to transfer the platform idea to the industrial manufacturing domain. For example, the research project BaSys 4.0[8] aims at inventing an operating system for embedded and cyber-physical systems in various production scenarios. Their goal is to realize a production process that allows for a high flexibility in manufacturing scenarios even across company boundaries. Several demonstrators are used to simulate the ability to efficiently change a production line. This is especially important for production lines that are specialized to produce multivariate goods. Furthermore, this operating system is designed to be open for unknown devices. Although it only provides a technical integration platform, standards such as eCl@ass can be incorporated to ensure syntactic and partially semantic interoperability. However, these standards still require aforementioned transformation rules.

All presented examples aim at formalizing the information structure of production devices. However, engineering processes used for hardware and software parts of I4.0-

[5] https://www.eclass.eu/
[6] https://www.profibus.com/technology/profinet/
[7] https://www.autosar.org/
[8] https://www.basys40.de/

Components are not compatible as they rely on different assumptions for using models. Based on the current bottom-up engineering approach, the main pain-points from a business perspective are:

- Technical information models are generated from program code (e.g. increased adaptation costs if models must be changed)
- Requirements realized by software are restricted by the device hardware early in the system engineering process (e.g. external consultants are needed to test early)
- Manual integration effort in the industrial manufacturing context increases as no standard as a single point of truth is present (e.g. inflexibility hinders new business models)

In this paper, we will introduce a top-down modelling approach for integrated I4.0 embedded devices based solely on information models. As a consequence, site operator and system integrator can specify required communication interfaces at the model level. This will help the involved parties to communicate and choose a suitable hardware device based on specific use-case requirements rather than creating an information model from device specific code.

1.1 Running example

Currently, core principles in the industrial manufacturing sector are changing. The so-called “Industrie 4.0” or “Industrial Internet of Things” is postulated to enable a high flexibility in production processes across company boundaries. Nonetheless, the required amount of time to reach a formal agreement for integration and information standards between all stakeholders may not be viable. Among others, further problems of the standardization process are standard conform implementations by the stakeholders and the integration of legacy devices. This is mainly due to the fact that standard creation processes are slow and the wheel of technical innovations turn fast. As a consequence, an abstraction layer that decouples software (e.g. code) and hardware (e.g. device model) is strongly needed in order to be as independent as possible from concrete hardware. This would allow for hardware exchange without any effect on the software.

The de-facto standard in the industrial automation domain for machine-to-machine communication is OPC UA[9]. OPC UA follows the service-oriented architecture principle and exposes an integrated information model as a public interface to restricted machine functionality. Despite the fact that technologies such as OPC UA do provide means to formalize operator-oriented pairs of standards for communication purposes, the desired end for automating system integration is not reachable as no common ground of truth between multiple parties exists. Furthermore, it cannot be guaranteed that these individual agreements are also applicable for other use-cases. In fact, current ways to make hardware device information available to software reinforces this. Currently, OPC

[9] https://opcfoundation.org/about/opc-technologies/opc-ua/

UA information models are generated from files and then instantiated during runtime. These files contain externally available program variables that are defined by a program in a programmable logic controller (PLC).

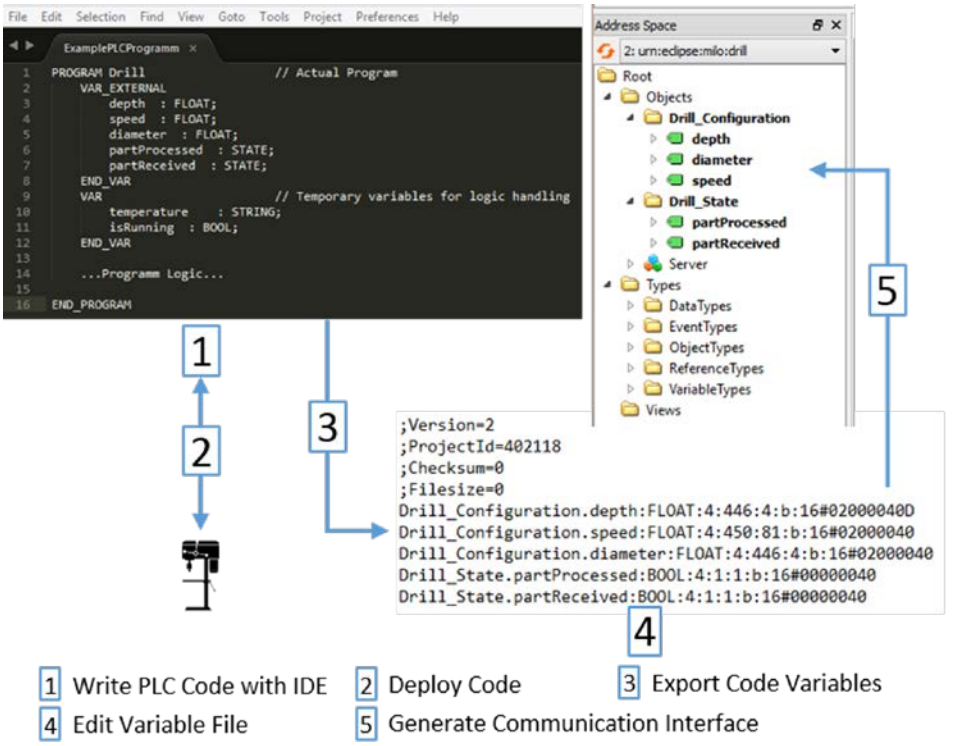

Figure 1: Generating Communication Interface based on code variables

For example, imagine a drilling machine that can drill a hole in a piece of wood. Assume that a PLC is attached to it and that the PLC contains two state variables “partReceived” and “partProcessed”. In addition, there are three configuration variables “speed”, “depth” and “diameter” which specifies the hole diameter this drill supports. This setup is visualized in figure 1.

At the moment, the overall engineering process currently includes the following abstract steps:

1. The system integrator (SI) must deploy application code (see 1+2 in Figure 1)
2. Next, SI must export selected program variables and edit the variable file (3+4)
3. Finally, SI must import the variable file into a communication framework (5)

This information model can be modified in accordance to the standards in use (e.g. eCl@ss). After instantiating the framework (e.g. OPC UA), the information model is synchronized at runtime with the application specific variables. In this way, another system that can speak OPC UA is now able to access information about the drill located in the model. Furthermore, a variable can serve as a trigger that performs application specific PLC actions (e.g. drilling a hole into a piece of wood).

It can be seen that this process produces an information model from code. From the viewpoint of a model-based engineering approach, this is not intended as the level of abstraction is raised instead of lowered when generating the information model. Furthermore, the site operator cannot change the information model without changing the code. Hence, a technician is needed, as the information model requires interventions at the PLC level and on the information model. We assume that this results in a unrulable amount of manual integration effort.

2 Supporting Solution Approaches

In order to put our example into the context of software engineering processes for embedded systems, the abstracted engineering steps (see Figure 2) to be taken are: Constructing a detailed draft (1), realize the system elements (2) and integrate the development artefacts into the system environment (3). In the following chapter, we integrate this these steps into a more sophisticated engineering process.

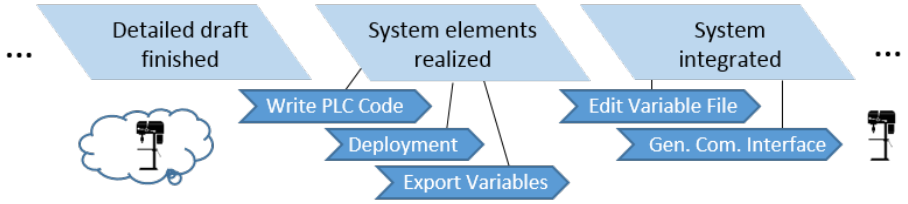

Figure 2: Derived engineering steps from running example

Overall, various process models have been developed that focus on a close cooperation between client and contractor. Applications and Information Systems that do not have a direct influence on their environment are currently being developed by using agile engineering methods (e.g. SCRUM or DevOps). Such rapid prototyping approaches for embedded systems or cyber-physical systems with a direct influence on their environment are in general forbidden by law (e.g. DIN EN ISO 10218 for collaborative robots). Nevertheless, examples such as the V-Model XT [BA05] also provide agile solutions for more controlled process models without sacrificing a close cooperation between client and contractor.

In the context of V-Model XT, the process is as follows: As soon as the project scope is fixed (see left side of Figure 3), the development of hardware and software components

is started. At the end of the development phase, all self-sustaining components are put back together and evaluated by both parties (see right side Figure 3).

Let´s take the classical model-driven software engineering approach for the specification and decomposition of the system. First, a high-level system architecture is derived from the results of the system requirements phase. Then, more details are specified over time until a software engineer can write code. During this process, different supportive tools are used. For example, the SysML[10] modelling language can be used for visualizing system characteristics from different viewpoints and for the identification problems (e.g. module incompatibility issues). However, this approach is constrained by the assumption that all needed components (i.e. software and hardware) are known at design time.

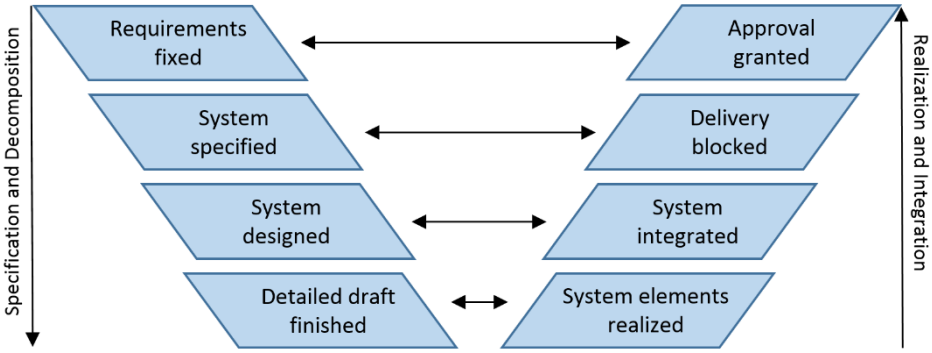

Figure 3: V-Model after Project Scope is fixed (adapted from [BA05])

Although a component refinement throughout the engineering process is possible (e.g. as realized by SPES XT approach [Po16]), we assume fixed interface definitions in the beginning.

Comparing this process to our running example (Figure 1), one can see that models of a system are not serving as a “root”. For instance, if a system engineer needs to change the OPC UA model, he must actually change the PLC-Code, export the variables and reimport them to the OPC UA server again. From our point of view, this is violating the desired principle of top-down, model-based system engineering.
Even worse, the tight coupling between machine-oriented code and its hardware-dependant information model doesn´t allow for quick changes. In the light of embedded and cyber-physical systems, this might sound counter-intuitive, as functional safety is essential for survival not only to machines but also to human operators. However, the point to be made here is not about incorporating safety requirements into the engineering process efficiently, but the requirement that hardware independent information models are needed to facilitate a sustainable integration process. If a site operator has to change code for changing a model, the amount of manual integration work will increase and may finally hinder flexible manufacturing plants. Thus, a site operator must be able to

[10] http://www.sysml4industry.org/

make changes at the software level. Softening the coupling between model and implementation, a site operator should be furthermore enabled to easily test the software-layer independently. This is especially important for cases where production devices are used in dynamic and self-adaptive contexts [Ba05][BRE15].

In the end, the site operator having less technical integration knowledge than the system integrator or device manufacturer should be able to express his requirements only in information models. As a consequence, the production equipment manufacturer must make sure that the models requirements are fulfilled in order to facilitate an easy device integration between software and hardware components.

3 Solution to our challenge

In favor of shifting back the focus from code to software models, we propose three adaptations to the engineering steps described in figure 2: First, information models should server as basis for software development for embedded systems. Second, software instances realizing these information models should be generated based only on software and not hardware requirements. Third, these instances should be used to integrate the system virtually (see figure 4).
In case of an error during the virtual system integration phase, it is also possible to go back to the information model and remodel the required interface. As soon as the information model is successfully integrated into the existing production environment, the needed hardware is equipped with the necessary PLC code.

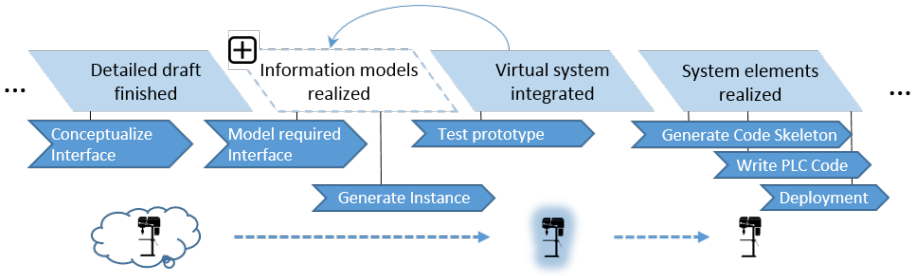

Figure 4: Adapted engineering process for the running example

By consulting our running example once again, the site operator (SO) and the system integrator (SI) can now perform the following steps:

1. First, SO/SI defines the required communication interface
2. Next, SO/SI can directly generate a test instance an integrate the system virtually
3. Finally, SI can continue the classical development process as in fig. 2

However, simplifying the technical realization of provided and requested device communication interfaces is only one aspect of integration issues in this context. Another aspect is the realignment of responsibility. Site operators should be in charge of defining the required interfaces, not the device manufacturer. As a consequence, site operators should also be able to test interface information models out of the box. This allows for selecting hardware devices that can perform the required functionality based on models. Overall, this role shift can serve as an initial basis for (automated) device integration in a flexible production scenario that is realized by heterogeneous and changing machines.

In order to execute our adapted top-down and model-based software development process, an integrated toolchain is beneficial. Based on our example, this toolchain should support code generation from models, automated component instantiation and at least support the following user requirements:

- Graphical CRUD-Operations of (existing/new) interface information models
- Fast set-up of test scenarios for testing the generated instances
- Easy adaptation of required and provided interface in case of production line reconfiguration

From the viewpoint of model-based software engineering, models are now back at their place.
Furthermore, not only the system integrator but also the site operator can quickly set up new scenarios, as no initial coding is needed anymore. Now, the process enables the site operator to specify the required communication interfaces without paying attention to device specific hardware parameters. This definition can then be handed over to the device manufacturer who must support the required functionality. On the other side, the site operator can also test the adaptations immediately by simulating a new communication interface. In the end, the site operator can independently define what is needed and the device producer must comply with these needs.

Nevertheless, this conceptual approach is particularly limited in two ways: First, the site operator must come up with an adequate interface definition almost out of thin air. This may require a comprehensible modelling language and engineers geared towards this task. Second, the required production functionality specified within the model must be connected manually to device specific application code. This application code may be unknown at the time of interface creation and must be manually adapted when a device is put into production in the first place.

4 Technical Realization

We implemented our suggested approach within the Eclipse Platform[11]. Therefor we designed a modelling plugin that can be used for creating new interface specifications or import them from running embedded devices. The models are based on the OPC UA[12] standard and the communication is handled by the Eclipse Milo[13] framework. An OPC UA test server can be started by importing the (adapted) information model from the modelling plugin into the Milo framework. This framework has been extended with an OPC UA standard conform XML-parser that instantiates the model elements.

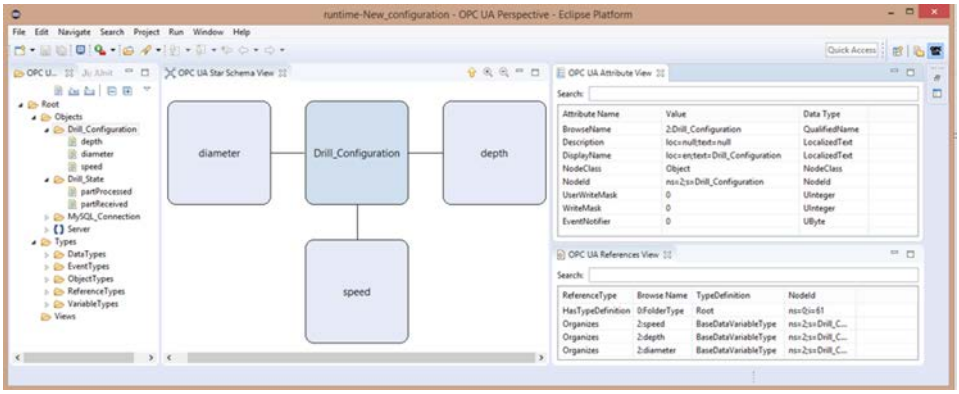

Figure 5: Tool support for adapted engineering process

The prototype displayed in figure 5 supports our adapted engineering approach (cp. figure 4). First, the engineer either creates a new communication interface or can import an existing model either from file or from a running OPC UA Server. Next, the information model can be adapted (e.g. adding a new node called distance or renaming the loaded node speed into torque). Next, the engineer can automatically generate an OPC UA Server from the edited model. An example of a generated and instantiated interface information model is displayed in figure 6. Otherwise, the engineer can continue with the traditional engineering process. Please note that during the virtual system integration phase, the information model cannot only be used for testing the software. It can also serve as a specification document that can be handed to various device manufacturers so that they can provide different solutions that realize the required functionality.

[11] http://help.eclipse.org/kepler/index.jsp?nav=%2F2
[12] https://opcfoundation.org/developer-tools/specifications-unified-architecture/part-5-information-model/
[13] https://github.com/eclipse/milo

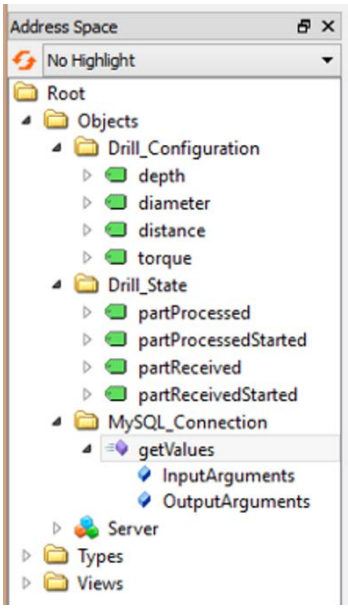

Figure 6: Generated OPC UA interface information model

Although there exist other tools like UaModeler[14] or FreeOpcUa[15], none of these tools are based on an open source development platform, can import OPC UA models and include an automated server instantiation feature. In addition, the conceptual challenge of connecting information models with embedded system code has been realized by attaching generic setters to all externalized variables. Hence, simple device functionality (e.g. writing values to a database for testing purposes) can be executed out of the box. However, more sophisticated device simulations based on methods represented in the information model or other behavioural aspects of embedded systems are currently not supported. Aside the mentioned practical advantages realized by the modelling tool, we spotted three major technical limitations. First, OPC UA is not well suited for specifying the required functionality of production devices accurately. Second, this prototype only allows changes to the interface at design and not at runtime. Furthermore, the code generation from an information model to PLC-skeleton code is currently not available but required when integrating software and hardware components. Third and last, the integration of legacy devices using a different communication standard is currently not supported by our tool.

5 Related work

In the last decades, a lot of work has been done in the fields of software development (e.g. component- and model based-development [Va16]), cyber-physical systems as well as dynamic and self-adaptive systems. Hence, this section only provides selected key resources corresponding to the running example context, but is far from complete.

Crnkovic, Chaudrin and Larsson introduce in their work about “Component-based Development Process and Component Lifecycle” [CCL06] a general engineering viewpoint for systems that are based on software components. Their work is seen as a basis for our prototypical implementation.
Hehenberger et al. used component-based development methods it in the context of cyber-physical systems [He16]. Furthermore, the authors provide recommendations for model-driven engineering for self-adaptive (i.e. self-configured), agent-based cyber-

[14] https://www.unified-automation.com/de/produkte/entwicklerwerkzeuge/uamodeler.html
[15] https://github.com/FreeOpcUa/opcua-modeler

physical systems. This work is providing further guidelines for elaborating on the engineering process.
An example of a full-fledged toolchain are the engineering tools developed by Harrison, Vera and Ahmad [HVA16]. They introduce component-based software development principles in a use-case that includes industrial robots. Their solution package “vueOne” is based on virtual simulation and virtual integration techniques for production equipment. Approaches that enable dynamic and self-adaptive systems have been successfully researched in the automotive context. For example, Braun et al. [Br12] introduced a formal approach that can be used to model evolution in automation engineering (i.e. component upgrading and an interdisciplinary compatibility supervision technique).

6 Conclusion

In this paper, we have justified that software engineering processes known from component-based software development are currently not applied in practical “Industrie 4.0” projects, especially for collaborative embedded systems. For example, current interface descriptions are primarily driven by code and not by models (e.g. OPC UA). However, such engineering approach makes it hard to change production setups quickly. As production life cycles get shorter each year and technological advances reduce the “time-to-live” for software and hardware, current engineering processes are not supporting future requirements efficiently. Hence, an adapted top-down engineering process has been suggested that places models as a device independent abstraction back at the core of embedded systems engineering. As a consequence, production lines are integrated from upside down. Nevertheless, current interface description techniques are not designed to support such processes. To realize dynamic and self-adaptive production lines efficiently, this paper introduces first ideas towards fulfilling such system qualities.

Bibliography

[BA05] Broy, M., and Andreas R., "Das neue v-modell® xt." Informatik-Spektrum 28.3 (2005). S. 220-229.

[Ba05] Bartelt, C., et al. Dynamic integration of heterogeneous mobile devices. In: ACM SIGSOFT Software Engineering Notes. ACM, 2005. S. 1-7.

[Br12] Braun, S., et al. Requirements on evolution management of product lines in automation engineering. IFAC Proceedings Volumes, 2012, 45. Jg., Nr. 2, S. 340-345.

[BRE15] Bartelt, C.; Rausch, A.; Rehfeldt, K., Quo vadis cyber-physical systems: research areas of cyber-physical ecosystems: a position paper. In: Proceedings of the 1st International Workshop on Control Theory for Software Engineering. ACM, 2015. S. 22-25.

[CCL06] Crnkovic, I.; Chaudron, M.; Larsson, S. Component-based development process and component lifecycle. In: Software Engineering Advances, International Conference on. IEEE, 2006. S. 44-44.

[He16] Hehenberger, P., et al. Design, modelling, simulation and integration of cyber physical systems: Methods and applications. Computers in Industry, 2016, 82. Jg., S. 273-289.

[HVA16] Harrison, R.; Vera, D.; Ahmad, B. Engineering methods and tools for cyber–physical automation systems. Proceedings of the IEEE, 2016, 104. Jg., Nr. 5, S. 973-985.

[Po16] Pohl, K., et al. "Advanced Model-Based Engineering of Embedded Systems." Advanced Model-Based Engineering of Embedded Systems. Springer International Publishing, 2016. S. 3-9.

[Va16] Vale, T., et al. Twenty-eight years of component-based software engineering. Journal of Systems and Software, 2016, 111. Jg., S. 128-148.