Prod. Eng. Res. Devel. (2017) 11:687–694
DOI 10.1007/s11740-017-0765-0

COMPUTER AIDED ENGINEERING

Towards the co-evolution of industrial products and its production systems by combining models from development and hardware/software deployment in cyber-physical systems

Birgit Vogel-Heuser[1] · **Stefan Wildermann**[2] · **Jürgen Teich**[2]

Received: 10 April 2017 / Accepted: 13 September 2017 / Published online: 24 October 2017
© The Author(s) 2017. This article is an open access publication

Abstract Industrial production systems and industrially manufactured products are constantly evolving due to technical innovations and customers' demands. Therefore, ensuring an optimized co-evolution of both becomes an important yet challenging task. Approaches for model-based systems engineering have been widely investigated and have already had significant impact on industrial practice. However, existing model-based approaches mostly are focusing on particular aspects of a production system and do not provide a holistic approach for optimizing automated production systems, their deployment, and their co-evolution with the product. As a first step towards this direction, this contribution proposes the combination of an engineering approach for aPS based on systems modeling language with models and techniques for deployment of software and hardware to cyber-physical system architectures, which have been investigated in the field of computer science, and motivates the possibilities of a combined approach.

✉ Birgit Vogel-Heuser
vogel-heuser@tum.de

✉ Stefan Wildermann
stefan.wildermann@fau.de

Jürgen Teich
juergen.teich@fau.de

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

2 Department of Computer Science, Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU), Cauerstrasse 11, 91056 Erlangen, Germany

Keywords Model-based engineering · Automated production systems (aPS) · Multicore deployment · Cyber-physical systems (CPS) · Cyber-physical production systems (CPPS)

1 Introduction

Current trends for industrially made products like small lot sizes and mass-customized products result in a divergence of the life cycles of industrial production systems and the products they produce. Whereas automated production systems (aPS) represent big investments and therefore have to be operated for several decades, industrial products are evolving much faster and innovations or changing customer requirements may occur even on a daily basis [1]. In order to cope with this divergence, aPS need to be flexible and able to co-evolve with innovations or changes of the products they produce [2]. On the other hand, "aPS are comprised of mechanical parts, electrical and electronic parts (automation hardware) and software, all closely interwoven. They represent a special class of mechatronic systems and consist of mechatronic sub-systems like sensors and actuators" [2]. Consequently, it is also necessary that aPS co-evolve by changes of its parts and sub-systems. In order to cope with these co-evolutions, two concepts lately have had a significant impact: model-based systems engineering (MBSE) and cyber-physical systems (CPS) for administrating physical entities (products as well as electronic components of aPS like sensors) during their lifetime. Whereas model-based evolution of CPS has already been investigated for different applications, sufficient approaches for the co-evolution of industrial products and aPS have not been presented yet. The envisioned goal of the work presented in this paper are advanced methodologies and techniques to support the co-evolution of models describing industrial products as instances of CPS as well as models describing aPS as instances of cyber physical production systems (CPPS). In the application example the product is a fluid that will be bottled later and referred to as a CPS with an RFID storing all relevant information and enabling tracking of the product and availability of the information in the cloud. One of the main intends of this co-evolution is to enable in future to co-optimize the CPS as well as CPPS during all phases of their interlinked life cycles for resource efficiency.

Taking a first step towards this goal, this paper presents an approach for MBSE based on systems modeling language (SysML) (e.g., [6, 8, 10]). This approach intends to enable an optimized deployment of a production system's automation software to automation hardware resources taking the co-evolution of the products to be produced into account. This deployment represents a substantial part of a CPPS's model that has not been sufficiently addressed in existing works on production automation. Common optimization goals within the deployment of software to hardware are (a) even distributions of the computing loads under (b) consideration of the real-time requirements of a particular application while (c) minimizing heat generation to decrease the aging/wear of the hardware resources. In order to realize such an approach, this paper presents the previous advances of the authors' groups and provides an outlook on a combined approach. In particular, the authors' works towards the model-based development of aPS [4] are discussed in combination with the works towards the model-based code deployment to heterogeneous cyber-physical system architectures [5] and a combined approach is outlined. The remainder of the paper is structured as follows: in Sect. 2, related work in the area of model-based development of CPPS automation level and model-based approaches for hardware/software co-design of electronic CPSs will be discussed. After the instance of a CPPS is introduced in Sect. 3, prior works in MBSE and software deployment is subsumed in Sects. 4 and 5. An integration of the two groups' approaches is motivated in Sect. 6 and evaluated in Sect. 7. Section 8 concludes this paper.

2 Related work

First, existing model-based approaches for CPPS automation level, are given and, second, model-based approaches for optimization and code deployment on heterogeneous CPS architectures are discussed.

2.1 Model-based development of flexible automated production systems

The increasing complexity of modern industrial aPS and their development and optimization during their whole lifecycle has led to MBSE as a possible solution for handling this complexity. Backhaus et al. [21] introduce an approach based on digital production process models aiming to shorten the engineering process, e.g. when adapting plants to new products and requirements. An approach for the systematic model-based development of production automation systems has been proposed by Vepsäläinen et al. [5] with AUKOTON. This approach covers several aspects of an automation system but lacks methods to integrate user-defined control logic [3, 6]. Possibilities to explicitly specify information for automatic deployment of Function Blocks onto controllers with respect to multiple optimization objectives (cost, latency, load distribution, and heat generation) are missing too. Reuter et al. [7] present an approach to integrate sub-models for control configurations in MBSE tools to improve the interdisciplinary modeling of manufacturing systems during development phase. Other SysML-based approaches [8, 10] provide the modeling of (non-functional) requirements and the application of simulations for design validation. However, in contrast to the work in [3] these approaches do not provide a direct integration with automation system software. The same holds for Thramboulidis et al. [9] which are also based on SysML and provide the modeling of domain-specific components along with their functional and non-functional requirements but lack a sufficient coupling between the functional models and run-time environments. The approaches [10, 11] of Fantuzzi et al. are also based on SysML as well as on Unified Modeling Language (UML) intend to describe the different aspects of aPS and mechatronic systems in general. In [11], design patterns for the generation of aPS' software are presented still lacking a close integration between the modeled behavior and executed code on the target platform as in [3]. Estevez includes models from various domains in the development of aPS using XML schemes [12, 13] but lack an optimized deployment of the functionality onto controllers. Brecher et al. introduce an approach for developing a control logic based on a SysML model [14], including component behavior models integrated in the overall system model. The approach addresses the reuse of the behavior models for the development of system modifications using control agents. A promising approach in the domain of building automation is [15], hence, a coupling between product and production system models is not considered. Whereas the aforementioned approaches mainly focused on automation systems based on the IEC 61131 standard, other works which address event-driven implementations conforming to the IEC 61499 standard [16, 17] exist but have not been widely accepted in industry, yet. To describe automation hardware devices, several languages like EDDL and FDT [18] for describing field devices and networked architecture systems in a detailed way exist. The approach motivated in this paper may be extended by these descriptions to address a domain-specific hardware modeling.

2.2 Model-based deployment for heterogeneous CPS platforms

The task of hardware/software co-design is to concurrently design hardware and software components of complex electronic systems [19]. It exploits the synergy of hardware and software with the goal to optimize/satisfy design objectives and constraints such as cost, latency, and power consumption. The major design decisions are made on the electronic system level (ESL), where it is decided whether system tasks are implemented in software or hardware, how the system architecture is built (allocation), and on which core each software module is executed (deployment/binding) by considering multiple objectives and constraints. This optimization problem is generally NP-complete [20]. Therefore, various techniques for automatized ESL design have been proposed in the literature, see [19] for a survey. A survey of techniques that are tailored to application mapping onto multi-core architectures is given in [22]. The technique for ESL design that we present in Sect. 5 basically adheres to techniques presented in these surveys.

Thermal conditions of computing hardware in aPS as an instance of a CPPS are greatly affected by thermal conditions emerging from the production process. Consequently, influences coming from a production process need to be considered inside a CPPS's model in order to achieve feasible, temperature-aware, and optimal (resource-efficient) deployments. In case of innovations or changes which cause evolutions inside a CPS's model, also the model of the corresponding CPPS can be co-evolved eventually by re-optimizing the deployment of its automation software. Karl and Reinhart [23] introduce a methodology to update existing production systems based on new requirements. After comparing the new requirements to the system capabilities, alternative matching system reconfigurations are created, selecting the reconfiguration with the lowest adaption complexity and expenses. The approach, concentrates on mechanical system components and their interdependencies rather than CPPSs. Consequently, in this paper, we show how techniques need to be adapted and extended for using them for the design of complex CPPSs.

3 Application example

In order to provide a comprehensive model example of one aPS embedded in one production system out of a CPPS network which co-evolves with its produced product, a lab size part of an aPS is introduced. The plant section provides a tank in which the fluid is heat-treated with an endothermal chemical reaction (Fig. 1a), a heating element providing the thermal energy (Fig. 1b) and a temperature sensor (Fig. 1c) to measure the fluid's temperature.

The temperature of the fluid needs to be controlled by a closed-loop controller (TC Fig. 1d) and visualized for the operator (TI).

To implement these tasks, a network of different computing resources is available: the temperature sensor as well as the heating element represent an automation node with one or more micro controllers and a field bus interface and thereby providing (limited) computational resources. The control loop is implemented in software on a Programmable Logic Controller (PLC) with a single-core processor and a fieldbus interface.

Constraints on the deployment of the compute tasks arise from the limitations of the computation resources and data transmission rate of the field bus but also from the environmental constraints like heat. Whereas the PLC is housed inside a temperature-controlled control cabinet, the temperature sensor and heating element are directly exposed to the thermal conditions of the fluid (product). In order to minimize the wear inside the automation equipment and thus, expand its sustainable lifetime, an optimized solution for the deployment of the tasks to the automation hardware needs to be developed that also satisfies the given real-time requirements of the production process.

As evolution step of the product, i.e. the fluid, an increased temperature in the production process is assumed. To adapt to this changed and higher temperature requirements the automation hardware including the controllers need to evolve in accordance. In case of networked production facilities which need to produce the same product in a similar plant the evolution steps can be used immediately due to communication and information exchange in between both.

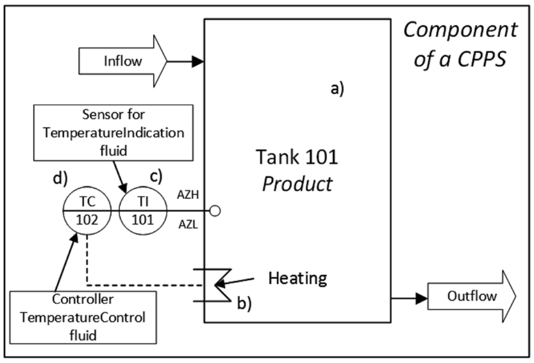

Fig. 1 Specification of Process and Instrumentation according to ISO 3511 [29]

4 Model-based development of aPS

For the model-based development of industrial aPS an SysML-based approach will be introduced for the application example [3] which particularly targets the design of networked aPS. The approach is based on a design procedure with multiple stages: based on the product specification at first the requirements of the production process are derived and modelled using the SysML requirements diagram (cp. Fig. 2).

For the application example it is assumed that the product model demands a constant temperature of 200 °C. From the functional requirement originating from this part of the product's model, other functional and non-functional requirements can be derived (cp. Fig. 2), e.g. regarding the operation of the sensor, actuator and the closed-loop control algorithm.

In the next step, these tasks are linked with the requirements they satisfy (cp. Fig. 3). The Block construct of SysML was extended in order to capture the information needed to describe the interfaces of a task: data which is generated and consumed by the tasks. By linking the provided and required information of the tasks, their data dependencies are integrated inside the model. In Sect. 5, different tasks can be derived from these requirements (cp. Fig. 3). Tasks A and E are responsible for the monitoring of the sensor's and, respectively, actuator's hardware condition. They do not generate data used by the other tasks. Consequently, they may run independently. In contrast, the tasks B, C, and D all contribute to maintaining a controlled temperature of the product inside the tank of the lab plant and therefore need to be executed in dependency to each other.

This dependency is given in the design model by the connections between the information generated and required by the tasks, i.e. task B pre-processes the value measured by the temperature sensor and generates data which is required by task C in order to execute the corresponding closed-loop control algorithm. The corrective action calculated inside task C is required by task D for the operation of the heating element. In addition to tasks, the available automation hardware can be described. The *Node* [3] encapsulates the information on computing hardware and its interfaces. In the application example used here, this part of the model contains the three nodes TemperatureSensor, *HeatingElement*, and *PLC* (Fig. 4).

In order to model the interfaces of nodes, the SysML element Port is used, e.g., EtherCAT fieldbus. (Details of SysML-based modeling of nodes and field-busses cp. [3, 24]). The fulfilling of requirements, e.g. like real time for specific tasks, using selected nodes with its characteristics, like storage and calculation power, could be proven by this approach. Next, approaches developed by the authors for the model-based deployment of applications in CPS platform architectures are described for the running example. Section 6 subsequently highlights the possibilities of a combined approach.

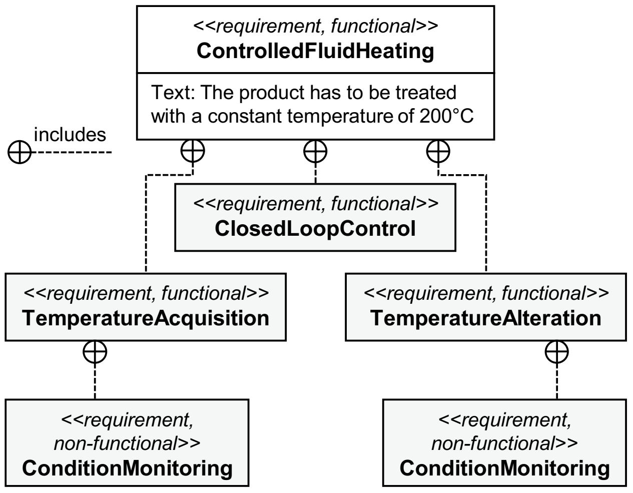
<<requirement, functional>>
ControlledFluidHeating
Text: The product has to be treated with a constant temperature of 200°C
includes
<<requirement, functional>>
ClosedLoopControl
<<requirement, functional>>
TemperatureAcquisition
<<requirement, functional>>
TemperatureAlteration
<<requirement, non-functional>>
ConditionMonitoring
<<requirement, non-functional>>
ConditionMonitoring

Fig. 2 Excerpt of the application example's requirements to guarantee the constant temperature derived from the product model

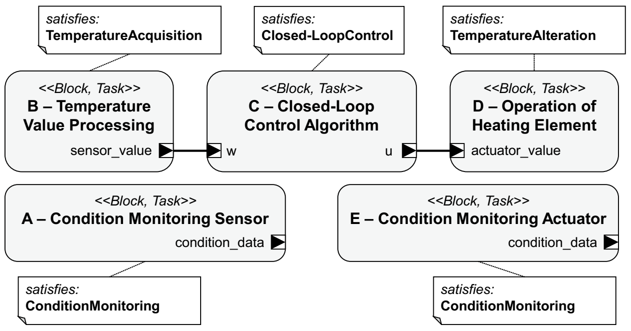
satisfies: TemperatureAcquisition
satisfies: Closed-LoopControl
satisfies: TemperatureAlteration
<<Block, Task>>
B – Temperature Value Processing
sensor_value
<<Block, Task>>
C – Closed-Loop Control Algorithm
w
u
<<Block, Task>>
D – Operation of Heating Element
actuator_value
<<Block, Task>>
A – Condition Monitoring Sensor
condition_data
<<Block, Task>>
E – Condition Monitoring Actuator
condition_data
satisfies: ConditionMonitoring
satisfies: ConditionMonitoring

Fig. 3 Derived computing tasks with requirements they satisfy

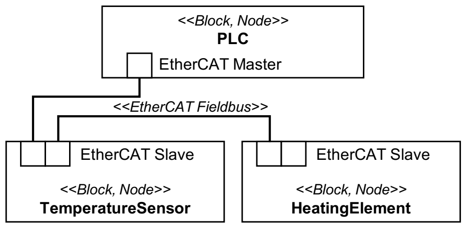

Fig. 4 Hardware model of the application example

5 Model-based approach for electronic system level design

ESL synthesis is the process of selecting an appropriate platform architecture (allocation), deploying the application tasks onto the allocated resources (binding), and scheduling the access of tasks onto shared resources. This process typically starts with an ESL specification given by a graph-based representation of (a) the application, (b) the architecture platform, and (c) the mapping options of application tasks onto architectural resources [28].

Figure 5 illustrates the specification graph of the application example in Sect. 3. The application graph consists of the five tasks with data dependencies between tasks B, C, and D. The architecture graph consists of the two microcontrollers, one PLC, and a connecting field bus. The dashed lines represent mapping options indicating which task could be assigned to which resource. A set of weights may be annotated to each mapping edge such as code costs, execution time, power consumption, etc. In this example, each task could be mapped to each of the three resources.

The design space of design candidates is defined by the set of mapping edges and grows exponentially with the number of tasks. Finding a design which is optimal with respect to multiple objectives, like cost and power consumption, and feasible for several design constraints, like upper bounds on worst-case latency and temperature, is an NP-hard multi-objective optimization problem [25]. Therefore, the task of design space exploration (DSE) (cp. Fig. 6) is to automatically explore and generate multiple such design candidates by allocating resources, binding tasks to processing resources and communications to routes in the architecture, as well as generating the schedules on shared resources. ESL not only considers the deployment of tasks to processors (CPUs, NCs, PLCs), but also to investigate hardware/software tradeoffs. If a task is mapped to such a unit, no software

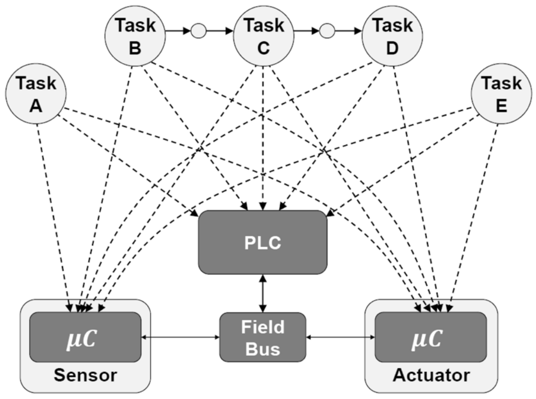

Fig. 5 Specification graph of the application from Sect. 3 and its data dependencies (top), the architecture (bottom) and the mapping options of tasks onto resources (dashed edges)

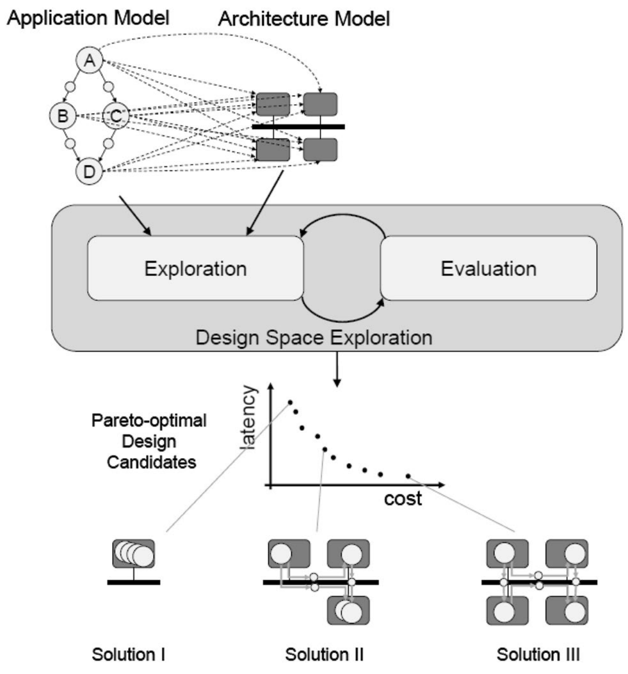

Fig. 6 DSE for finding feasible and optimized ESL designs. Based on the specification model, an iterative optimization heuristics generates and evaluates different implementation candidates and keeps the optimal (non-dominated) ones

but a dedicated hardware implementation will be generated through hardware synthesis tools. The non-functional properties of each candidate are then evaluated by applying analytical and/or simulation-based techniques.

Due to the NP-hardness, population-based multi-objective optimization heuristics are commonly used such as evolutionary algorithms [25]. Here, a population of design candidates is iteratively generated, evaluated and then evolved to form the next population generation. In ESL synthesis usually not a single but multiple objectives are independently optimized. The optimization results in multiple, so-called Pareto-optimal solutions; each representing a certain tradeoff between the objectives. The designer can then select the most promising design for subsequent implementation.

The choice of an evaluation technique that is appropriate for this optimization process considerably depends on the property to quantify. For example, hardware cost is often modeled by a linear cost function, whereas timing and power are tackled by simulation-based performance evaluation [4, 26]. In case of hard real-time requirements, the evaluation is performed by applying complex timing analysis techniques to obtain worst-case task execution times (WCET) [20]. In this case, a design candidate is only feasible if the WCET is smaller than a given deadline.

Besides real-time requirements, peak temperature is a major concern in ESL synthesis. Temperature evaluation is based on complex models and simulations. A widely applied methodology used for thermal modelling and simulation-based evaluation is HotSpot [27].

6 Opportunities of a combined approach

The models for aPS development (Sect. 4) and electronic system level design (Sect. 5) fit perfectly together to enable a (semi-)automatic development and—even more important—evolution of CPPS. Figure 7 presents the respective design flow which combines these approaches. The SysML model contains all information of the CPPS. Particularly, the task model and the hardware model can be directly translated to an application graph and an architecture graph required for DSE. When the mapping options are not available, they can be automatically generated by synthesizing each task for the target devices and profiling their execution on them (see, e.g., [4]) or by applying worst-case analysis tools like AbsInt [28]. The behavior of a task itself may be modelled, e.g., in SystemC that is used in [4] for functional and timing verification and code synthesis (hardware and software). Also Simulink models could be used and automatically converted for subsequent DSE.

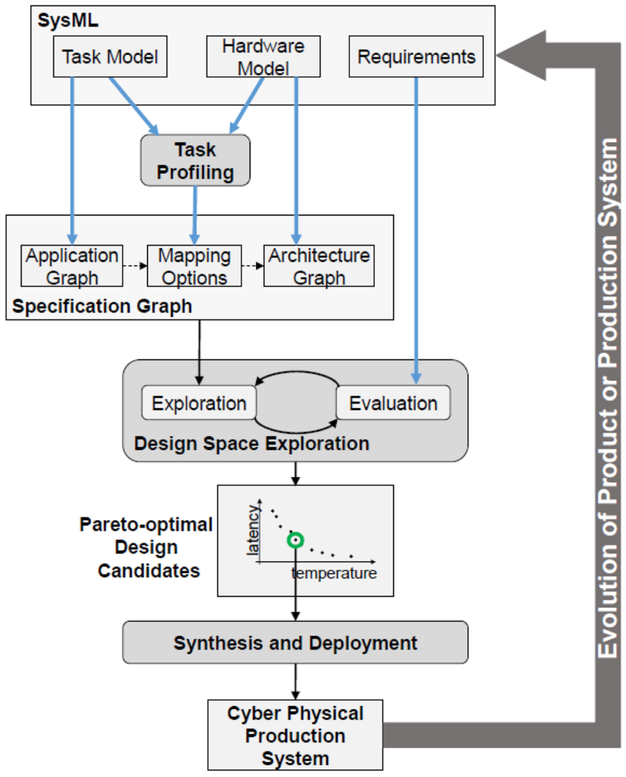

Fig. 7 Proposed design flow for evolution of CPPS by combining models from automation level design and electronic system level design for automatic deployment. Blue edges indicate model transformations that should happen automatically to alleviate the evolution of CPPSs

The requirements specify the frame conditions, design objectives and constraints of the product and, therefore, represent a possible link between a production system's model and the model of industrial products. Thus, they contain the information about which objectives have to be optimized and which constraints have to be verified during DSE and are therefore used for DSE configuration: An appropriate evaluation technique has to be instantiated for each objective and each constraint. DSE then performs allocation, binding, routing, and scheduling to determine optimized and feasible design points. The result is a set of Pareto-optimal solutions each with a different tradeoff between the multiple objectives. For instance, Fig. 7 illustrates tradeoffs between latency and temperature: a high-performance solution with low latency might cause a high temperature, whereas using a slow implementation might result in a lower peak temperature. In the end, a designer has to make a decision which tradeoff to go. The software and hardware modules of the selected solution are generated and finally deployed. In the presented application example, a suitable deployment of the closed-loop control task and for the condition monitoring tasks may be identified. Subsequently, the corresponding software parts can be deployed to the available nodes, including necessary field bus configurations, etc.

7 Strength of the combined approach

Designing, deploying, and evolving aPS is an interdisciplinary challenge requiring expertise of, amongst others, production software and hardware engineers. Already the presented example application with only minor complexity has 27 deployment candidates. Generally, the complexity grows exponentially in the number of tasks. Therefore it is almost impossible to anticipate which impact design decisions made during production engineering have on the non-functional properties of the final networked aPS embedded in CPPS.

Models used during the design process typically are subject to uncertainties on specifications and assumptions, e.g., on environmental conditions. Moreover, the product or aPS might change or be updated during the operation. For example, considering the application example from Sect. 3, when the peak temperature for treading the product has to increase, this influences also the peak temperature and temperature distribution on the microcontrollers of the temperature sensor and the heating element, potentially leading to a violation of the respective temperature constraint. Particularly, when industrial products and CPPSs evolve faster, such revisions have to be made (a) more frequently and (b) be implemented much faster in an existing CPPS without delaying the ongoing production process.

The proposed design flow is intended to tackle exactly this challenge. The SysML frontend represents a modelling technique that is intuitive for production engineers: it provides both a flexible specification framework guaranteeing conformity to production engineering standards and to consider environmental aspects of CPPS such as temperature of plant, sensor subsystems, etc.

The strength of the proposed flow is that, based on this specification, an automatic multi-objective DSE can be initiated. The respective methodology outlined in Sect. 5 deploys not only software to CPUs, but also hardware-synthesized functions, e.g., on application-specific integrated circuits or field-programmable gate arrays, are considered. This happens automatically and has proven high scalability for designing distributed systems, see e.g. [20]. The result is a set of Pareto-optimal CPPS design candidates. Thus, the production engineer gets feedback about the implications of his/her design decisions on the deployment and non-functional properties of the CPPS under design. The design flow is completed by an automatic generation of software code and hardware modules implementing the functions to be deployed through compiler and high-level hardware synthesis technologies for the proper targets.

8 Conclusion and outlook

Nowadays, the increasing speed of technical innovations and changing customer requirements, which may cause separate evolutions of a production system as well as of a product, causes the lifecycles of products and production systems to diverge. In order to cope with this challenges, new approaches for proactively handling the co-evolution of products and production systems need to be developed. Methods and techniques from MBSE have already had a great impact in academia and partially in industry. However, model-based approaches often do not provide holistic descriptions of all important aspects to sufficiently handle the evolution of a production system. In this paper, a modeling approach is motivated, which couples development models of aPS with model-based ESL design. Furthermore, by incorporating functional and non-functional requirements to the production process, a first possibility to couple a product's model with the model of an aPS is proposed. The methodology includes a design space exploration that performs a multi-objective optimization. This means that multiple deployments are evaluated with respect to objectives like resource costs, energy efficiency, latency, and throughput. The result of this process is a set of Pareto-optimal deployment options. They reflect different trade-offs. For example, one deployment includes many high-cost resources but has a high throughput. Whereas another one includes low-cost resource with lower throughput. A decision maker can then select the deployment that fits best his/her needs very early in the design. This exploration therefore helps a lot to avoid any late changes in the design often required due to either under-provisioning or under-utilizing resources. Moreover, for the addressed use-case of deployment of functionality in CPPS, a concept to co-evolve the model of a CPPS along with changing production process requirements and, hence, evolutions of a product's model is proposed. By the motivated concept (cp. Fig. 7) future works are outlined which include the elaboration and implementation of the discussed approach. Furthermore, evolvable model descriptions for products as CPS will be developed in close integration with the CPPS models in order to fully enable a co-evolution and optimization of both, CPPS and CPS.

Open Access This article is distributed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits unrestricted use, distribution, and reproduction in any medium, provided you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license, and indicate if changes were made.

References

1. Birkhofer R, Feldmeier G, Kalhoff J et al (2010) Life-Cycle-Management für Produkte und Systeme der Automation: Ein Leitfaden des Arbeitskreises Systemaspekte im ZVEI Fachverband Automation, Frankfurt
2. Vogel-Heuser B, Fay A, Schaefer I et al (2015) Evolution of software in automated production systems: challenges and research directions. J Syst Softw 110:54–84. doi:10.1016/j.jss.2015.08.026
3. Vogel-Heuser B, Schütz D, Frank T et al (2014) Model-driven engineering of manufacturing automation software projects—a SysML-based approach. Mechatronics 24(7):883–897. doi:10.1016/j.mechatronics.2014.05.003
4. Keinert J, Streubūhr M, Schlichter T et al (2009) SystemCoDesigner—an automatic ESL synthesis approach by design space exploration and behavioral synthesis for streaming applications. ACM Trans Des Autom Electron Syst 14(1):1–23. doi:10.1145/1455229.1455230
5. Vepsalainen T, Sierla S, Peltola J et al (2010) Assessing the industrial applicability and adoption potential of the AUKOTON model driven control application engineering approach. In: 8th IEEE INDIN, pp 883–889
6. Fay A, Vogel-Heuser B, Frank T et al (2015) Enhancing a model-based engineering approach for distributed manufacturing automation systems with characteristics and design patterns. J Syst Softw 101:221–235. doi:10.1016/j.jss.2014.12.028
7. Reuter A, Kircher C, Verl A (2010) Manufacturer-independent mechatronic information model for control systems. Prod Eng Res Devel 4(2):165–173. doi:10.1007/s11740-010-0220-y
8. Cao Y, Liu Y, Paredis CJ (2011) System-level model integration of design and simulation for mechatronic systems based on SysML. Mechatronics 21(6):1063–1075. doi:10.1016/j.mechatronics.2011.05.003
9. Thramboulidis K, Frey G (2011) Towards a model-driven IEC 61131-based development process in industrial automation. JSEA 04(04):217–226. doi:10.4236/jsea.2011.44024
10. Bassi L, Secchi C, Bonfe M et al (2011) A SysML-based methodology for manufacturing machinery modeling and design.

IEEE/ASME Trans Mechatron 16(6):1049–1062. doi:10.1109/TMECH.2010.2073480
11. Bonfè M, Fantuzzi C, Secchi C (2013) Design patterns for model-based automation software design and implementation. Control Eng Pract 21(11):1608–1619. doi:10.1016/j.conengprac.2012.03.017
12. Estévez E, Marcos M, Lüder A et al (2010) PLCopen for achieving interoperability between development phases. In: 15th IEEE ETFA, pp 1–8
13. Estevez E, Marcos M (2012) Model-based validation of industrial control systems. IEEE Trans Ind Inf 8(2):302–310. doi:10.1109/TII.2011.2174248
14. Brecher C, Nittinger JA, Karlberger A (2013) Model-based control of a handling system with SysML. Procedia Comput Sci 16:197–205. doi:10.1016/j.procs.2013.01.021
15. Fuentes DED, Becker U, Diekhake P et al (2016) Evaluation and simulation of building automation systems based on their AutomationML description. In: 21st IEEE ETFA, pp 1–6
16. Vyatkin V (2011) IEC 61499 as enabler of distributed and intelligent automation: state-of-the-art review. IEEE Trans Ind Inf 7(4):768–781. doi:10.1109/TII.2011.2166785
17. Hirsch M, Missal D, Hanisch H-M (2008) Design and verification of distributed industrial manufacturing control systems. In: 34th annual conference of IEEE industrial electronics, pp 152–157
18. Lugert S, Gisy S, Kato M (2010) Combination of FDT and EDDL technologies: Interpreter DTM to unify device integration. In: SICE annual conference 2010, proceedings of, pp 721–723
19. Teich J (2012) Hardware/software codesign: the past, the present, and predicting the future. Proc IEEE 100(Special Centennial Issue):1411–1430. doi:10.1109/JPROC.2011.2182009
20. Glaß M, Lukasiewycz M, Teich J et al (2009) Designing heterogeneous ECU networks via compact architecture encoding and hybrid timing analysis. In: ACM/IEEE design automation conference, vol 46, pp 43–46
21. Backhaus J, Scheib J, Pieloth R et al (2016) Automatic programming of processing machines—digital models as a basis for efficient engineering. Automatisierungstechnische Praxis (atp) 58(11):36–45
22. Singh AK, Shafique M, Kumar A et al (2013) Mapping on multi/many-core systems. In: Design automation conference (DAC), vol 50
23. Karl F, Reinhart G (2015) Reconfigurations on manufacturing resources: Identification of needs and planning. Prod Eng Res Devel 9(3):393–404. doi:10.1007/s11740-015-0607-x
24. Hashemi Farzaneh M, Feldmann S, Legat C et al (2013) Modeling multicore programmable logic controllers in networked automation systems. In: 39th IEEE IECON, pp 4398–4403
25. Blickle T, Teich J, Thiele L (1998) System-level synthesis using evolutionary algorithms. Des Autom Embedded Syst 3(1):23–58. doi:10.1023/A:1008899229802
26. Rosales R, Glass M, Teich J et al (2014) MAESTRO—holistic actor-oriented modeling of nonfunctional properties and firmware behavior for MPSoCs. ACM Trans Des Autom Electron Syst 19(3):1–26. doi:10.1145/2594481
27. Huang W, Ghosh S, Velusamy S et al (2006) HotSpot: a compact thermal modeling methodology for early-stage VLSI design. IEEE Trans VLSI Syst 14(5):501–513. doi:10.1109/TVLSI.2006.876103
28. Ferdinand C (2004) Worst case execution time prediction by static program analysis. In: International parallel & distributed processing symposium IEEE, pp 125–127
29. ISO 3511-3 (1987) Process measurement control functions and instrumentation; symbolic representation; part 3: detailed symbols for instrument interconnection diagrams (ISO 3511-3:1984-07)