Application of an Intelligent Network Architecture on a Cooperative Cyber-Physical System: An Experience Report

Uwe Pohlmann
Fraunhofer IPT, Project Group
Mechatronic Systems Design
D-33102 Paderborn, Germany
Email: uwe.pohlmann@ipt.fraunhofer.de

Henning Trsek, Lars Dürkop
inIT - Institute Industrial IT
Ostwestfalen-Lippe UAS
D-32657 Lemgo, Germany
Email: henning.trsek@hs-owl.de
lars.duerkop@hs-owl.de

Stefan Dziwok, Felix Oestersötebier
Heinz Nixdorf Institute
University of Paderborn
D-33102 Paderborn, Germany
Email: stefan.dziwok@hni.uni-paderborn.de
felix.oestersoetebier@hni.uni-paderborn.de

***Abstract*—Cooperative cyber-physical systems (CCPS) are driven by the tight coordination between computational components, physical sensors and actuators, and the interaction with each other over system bounds. The software development of CCPS is getting more complex because of the tight integration, heterogeneous technologies, as well as safety and timing requirements. Therefore, new engineering approaches, such as model-driven development methods, are required, along with communication architectures with self-* capabilities. Both will support the developer in specifying such a system effectively and efficiently. However, the application of such techniques for the development of CCPS has not been addressed sufficiently so far. This paper presents an experience report of developing a cooperative delta-robot system that juggles a ball without a central control or camera system. For the development, an intelligent network architecture and model-driven development method for CCPS are applied.**

I. Introduction

Cooperative cyber-physical systems (CCPS) require a tight combination of and coordination between computational components and physical devices. As a result, they are able to sense their environment, to take "intelligent" decisions, and to interact with their environment. CCPS have to interact and coordinate with each other over system bounds to fulfill advanced tasks. Accordingly, the tight coupling between controlling and controlled components is broken-up and complex communication sequences over networks must handle the coordination. Therefore, the CCPS development requires interdisciplinary teams composed of system, software, control, and communication engineers.

The adoption of reconfigurable manufacturing systems (RMS) in industrial automation is one example of CCPS. To react fast and cost-effective on new challenges, a RMS shall allow the adding or removing of modules and machines from the production process with none or little manual configuration effort. As a result, the process has to reconfigure itself after each physical modification. The general approach in research to realize such systems is based on the modularization of production systems in distributed, autonomous, and intelligent objects. Therefore, methods and components from the information technology merge with traditional automation devices. In order to reduce the manual setup efforts to enable RMS, such systems require self-configuration capabilities [1].

In this paper, we focus on the software development of such systems. For example, applications use coordination protocols that negotiate the message-based interaction between system parts. Moreover, received messages influence the reference values of continuous controllers that use advanced algorithms and control physical actuators. Further, applications are bound to physical laws like communication delay, but have to fulfill safety and hard real-time requirements. Additionally, the usage of heterogeneous technologies by discipline-spanning development teams result in a challenging development process. To cope with these challenges, new engineering approaches, such as model-driven development methods, are required, along with network architectures which are able to reconfigure themselves. The self-configuration capabilities are necessary for a simple integration of the application and the communication layer. By today, both layers are strongly coupled – especially in the field of real-time communication systems where the application developer has always to configure the real-time network simultaneously. To handle the complexity of CCPS, an application development without considering the underlying network technology is desired. As a result, the developer only needs to formulate the requirements on the communication system. Afterwards, the communication layer configures the network automatically with respect to the specified demands and provides an appropriate communication channel to the application.

However, the application of such techniques for the development of CCPS has not yet been addressed sufficiently. Especially, there are very few examples for the development of CCPS in closely collaborating, discipline-spanning development teams who are using a common architecture and discipline-specific development methods. One example for an existing report by Derler et al. [2] describes how to model a fuel management system of an aircraft with Ptolemy. Another report by Zhang et al. [3] elaborates on how to develop a robot car. However, both reports do not address the challenges of discipline-spanning teams, cooperative behavior, and a real-time network architecture. Ricken and Vogel-Heuser [4] discuss the challenges of interdisciplinary devel-

 http://dx.doi.org/10.1109/ETFA.2014.7005358

opment for manufacturing systems, but do not consider that manufacturing systems need an underlying intelligent network architecture. Other research activities address only partially the technical challenges for developing CCPS. The existing work can be mainly categorized in two classes, component-based development methods for the application layer and real-time communication networks with self-configuration capabilities. State of the art component-based development methods [5] define the meaning of a component and the reason of the operations *construct*, *compose*, and *deploy* on components [6]. Self-configuration capabilities in order to provide plug-and-play (PnP) functionalities were already the objective of several research papers [7].

The key novelties of our approach include the tight integration of a model-based software engineering approach with an industrial-scale control engineering development method [8]. Moreover, an underlying real-time communication network with self-configuration capabilities provides the required flexibility for CCPS. In contrast to other approaches, our verification and simulation techniques allow the analysis of models supporting reconfiguration. Timing constraints as well as a time-dependent asynchronous message-exchange including message loss, message delays, and several message buffers with replacement policies can be considered.

In the German research project ENTIME[1] and the succeeding project it's OWL[2], we created and evaluated methods for the design of (industrial) CCPS. Both projects have a close collaboration between research and local industry. Within these projects, test beds for CCPS have been developed. One of them consists of two cooperating delta robots that juggle a ball by passing it to each other. The cooperating delta robots have parallel manipulators that are often used for pick-and-place applications in manufacturing systems. The system is comprised of two identical, autonomous robots, which are equipped with a movable racket on their tool-center-points. The rackets are used to strike the ball. Fig. 1 shows the physical construction of the cooperating delta robots. In contrast to similar systems, no optical sensors are used to detect the ball. As a resulting challenge, the striking robot has to simulate and predict the ball flight trajectory. Three piezo force sensors under each racket plate constitute the only source of information to enable stable playing. This denotes an ambitious control task, which requires model-based observers. Another challenge is that no central control unit for both robots exists. Thus, the robots have to coordinate with each other via an intelligent network architecture considering hard real-time requirements. Therefore, the robots shall exchange messages with information about the current gaming strategy and a prognosis for the next strike. To summarize, the cooperating delta robots are a representative of a new class of CCPS that lead to new requirements on a systematic design process considering different disciplines.

The contribution of this paper is an experience report about the application of an intelligent network architecture on the software of the cooperating delta robots. We describe how we have used our method in a discipline-spanning team, the challenges we had to cope with and the lessons that we have learned. By taking our experience into account, other discipline-spanning teams from industry and academics can improve their efficiency and the quality of their resulting CCPS software.

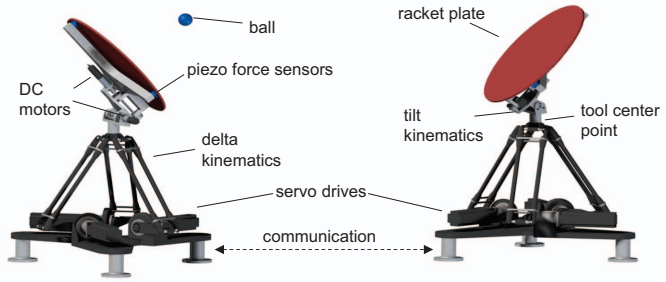

Fig. 1. Cooperating Delta Robots[3]

The remainder of the paper is structured as follows: The next section provides relevant background in the area of model-driven engineering methods and real-time communication networks. Section II summarizes our intelligent network architecture that addresses the requirements of CCPS with hard real-time tasks. Afterwards, in Section III, we describe the software development of the cooperating delta robots and discuss our experiences in Section IV including the faced challenges and our lessons learned. Finally, the paper concludes with its main findings and provides an outlook towards future work.

II. Intelligent Network Architecture for CCPS

An intelligent communication network is one of the enablers for CCPS. Hence, a software architecture for CCPS has been specified [9]. Fig. 2 shows the architecture that follows a layered approach. The architecture has the main objective of providing services for a self-configuration of the real-time communication network and to decouple the application from the communication network. Therefore, three architectural layers have been defined: application, middleware, and connectivity.

A. Application Layer

The application layer provides the required functionality for the system. It consists of a set of application components that is an abstraction of the real application structure. A component has its own activities and an active discrete or continuous state. Every component has an interface description. This description has a syntactic part that defines which messages or signals may be sent and received. Further, the description has a semantic part that describes the meaning of the signals and messages as well as quality-of-service requirements. The semantic description is a basic requirement for implementing a self-configuration. For instance, the semantic description of a simple temperature sensor could include the measured unit and its range.

B. Middleware Layer

The middleware layer is responsible for decoupling the application layer and the underlying connectivity layer as well as selecting a suitable communication technology. It offers a standardized communication interface to the application, independent of the underlying real-time communication systems.

[1] http://entime.upb.de
[2] http://t.co/AKfVwUpOxV
[3] http://twitpic.com/d1u0gh

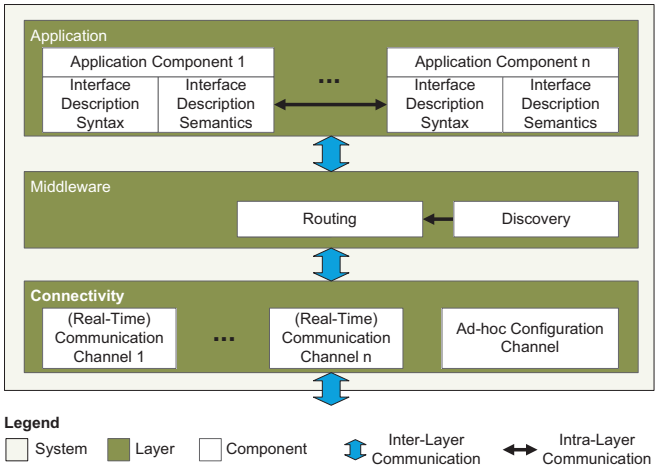

Fig. 2. Intelligent Network Architecture for CCPS

The middleware consists of several services being responsible for application data exchange and self-configuration of the nodes. For the exchange of data, the middleware has to select a suitable communication technology, which is able to meet the application requirements. The requirements are described within each application component and are provided to the middleware via a standardized interface. The core component for the self-configuration is a discovery service, which allows to discover other existing nodes and to recognize their semantics. For both functionalities, it requires an ad-hoc communication channel as described in Section II-C. An example bootstrapping would consist of three steps: *(i)* discovery of other nodes, *(ii)* configuration of the real-time communication systems, and *(iii)* the logical interconnection of application components to their relevant peers. After these steps are completed, the routing block is responsible for selecting valid paths to exchange information between different systems, especially if several real-time communication systems are used.

C. Connectivity Layer

The connectivity layer is capable of providing different real-time communication channels depending on the requirements of the application. They are used to exchange process data of the application with deterministic temporal boundaries. Besides real-time traffic, it is also possible to exchange best effort traffic, which is used for any non-critical data such as monitoring data or network management data. The connectivity layer also provides an ad-hoc communication channel. As soon as a new CCPS is physically connected to the network, the ad-hoc channel is automatically established and can be used to exchange configuration data during the bootstrapping phase. Moreover, depending on the requirements of the application, the connectivity layer is able to offer various communication technologies ranging from deterministic wired hard real-time communication systems to wireless technologies providing only soft real-time guarantees.

III. Developing the Software of the Cooperative Delta Robots

In the following, we report about our development of the software of the cooperative delta robots by applying our intelligent network architecture for CCPS. At first, we define a conceptual design to check the feasibility of the system and to retrieve requirements on the software. Afterwards, we develop the application layer, before developing the middleware and the connectivity layer.

A. Conceptual Design

In this stage, we built up idealized simulation models of the dynamics in order to be able to evaluate the principle feasibility of the concept. Therefore, models that represent the physical principles of an abstract technology are necessary. We use the idealized model of the system to design the first controller concept. Herein, we define the height of the ball trajectory to be the controlled variable. In addition, the distance of the point of impact from the racket center is controlled to zero. The robot that strikes gets to know the actual position and velocity of the ball by evaluating the measured forces at the time of the contact. Feeding a model-based observer with this information, it calculates a prognosis of when and where the ball will reach the other robot. The robot sends this prediction to the other robot. As a result, the receiver robot can compute a strike trajectory and thus can strike the ball properly. Considering the distance between the two robots, the minimum height of the ball trajectory, the gravity on earth, and the time needed for the alignment of the receiver's plate, the maximum allowed time for the transmission of the prognosis is 336 ms [10].

B. Application Layer

We define the software of the application layer using MechatronicUML [11] which is a software engineering method that is especially designed for developing this software. MechatronicUML is model-driven. That means it uses models for the whole development process from the requirements to the source code and supports the developer with automatic transformation to combine the different development phases. MechatronicUML provides a domain-specific modeling language that enables formal verification of the discrete-event application parts via timed model checking.

For specifying the structure, MechatronicUML provides instantiable software components that define ports as interaction points. Ports that are connected can exchange information, e.g., discrete-event ports may exchange asynchronous messages. MechatronicUML enables to define contracts called Real-Time Coordination Protocols for defining, which messages may be sent and received under hard real-time requirements via discrete-event ports. The semantics of these protocols is defined by so-called Real-Time Statecharts (a combination of UML state machines and timed automata).

Considering our cooperating delta robots, Fig. 3 shows an extract of the component structure of the application layer. Here, the discrete-event ports of two instances of software component RobotSW are connected to each other over the protocol ExchangePrognosis. It is used while the game is active and coordinates that after each strike, the prognosis for the next strike is calculated and sent to the other robot. Furthermore, ExchangePrognosis specifies the real-time requirements that we defined in the conceptual design phase, e.g., it specifies that r1 has at most 3 ms of computation time after strike before sending the prognosis and that r2 at most 151 ms of computation time after receiving the prognosis before its strike.

Real-Time Coordination Protocols make assumptions concerning the quality-of-service of the port connector. In our case, ExchangePrognosis assumes the following: (i) the maximum message delay from application layer to application layer is 336 ms, (ii) all messages eventually arrive to the receiver, (iii) messages are not reordered, (iv) no message is delivered twice, (v) corrupted messages will not be delivered, (vi) man-in-the-middle attacks are not possible, and (vii) the sender and the receiver can synchronize their system time with a precision of a few milliseconds. As long as all assumptions are fulfilled, we can formally verify our protocols via timed model checking, e.g., we can prove that there will never be a deadlock within the message exchange.

The component model of MECHATRONICUML enables to specify continuous-time as well as discrete-event software components such that control and software engineers have one common model with clearly defined interfaces. In particular, continuous-time components represent controllers that are developed by the control engineers. Such components contain continuous ports that transfer signals of a specific data type, e.g., Real, or Integer values. In contrary, discrete-event components are developed by software engineers. Their component behavior is defined in a state-based manner using Real-Time Statecharts and may be formally verified, e.g., via timed model checking. Discrete-event components contain the already mentioned discrete-event ports for exchanging messages as well as so-called hybrid ports that transfer time-continuous signals values into sampled local variable values. Thus, hybrid ports can be connected to continuous ports of a continuous component.

We show the internal software structure of a cooperating delta robot in the right side of Fig. 3. It contains the discrete-event component CoordinationModule that is able to exchange messages with other discrete-event components. The CoordinationModule adheres to the protocol ExchangePrognosis. As a result, both robots can coordinate the exchange of the prognosis. CoordinationModule is connected via hybrid ports to the continuous components BallDetection, StrikeControl, and LocalRacketControl. Thus, CoordinationModule can send or receive new prognosis data for the transmission to or from the other robot and can send or receive input values for the controllers. The component-based structure has the benefit that parts of the software can be easily exchanged. The clear separation between event-discrete components and continuous-time components has the advantage that safety properties for event-discrete components can be proved via timed model checking. Moreover, the clearly defined interfaces between the discrete-event and continuous-time components can reduce misunderstanding of the disciplines.

C. Middleware and Connectivity Layer

As shown in Fig. 3, the top-level components of both the robots are using ports to communicate with each other. The behavior of the ports is restricted by Real-Time Coordination Protocols. However, these protocols describe the platform independent communication process that assumes certain properties of the platform dependent network architecture.

In this architecture, the middleware is responsible for the real-time delivery of messages between the ports of

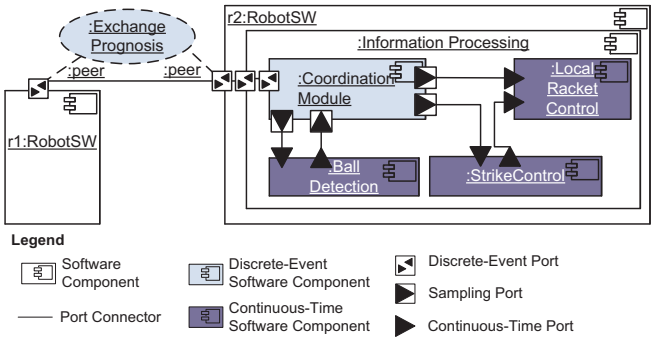

Fig. 3. Application Component Structure

distributed software components. Therefore, the middleware accepts messages from a port, transforms the messages according to the underlying real-time capable network, and finally transmits them. The details of this process are described as follows. The basis of the communication process is a real-time data exchange between the components. Here, real-time Ethernet (RTE) is used, because it will be mainly deployed in future automation systems [12]. RTE is a general term for different communication network standards like Profinet IO, EtherNet/IP, or EtherCAT. However, the advantage of real-time communication is at the expense of an increased manual configuration effort for these systems. To overcome this disadvantage the architecture includes a self-configuration mechanism for Profinet IO. A detailed description of this self-configuration mechanism is provided in [1].

Each port owns a semantic description as well. When the middleware of robot 1 receives an *ExchangePrognosis* with a *send*-attribute, its discovery block scans the network for the appropriate counterpart – i.e. a port which can receive this message type. The semantic descriptions of the available ports are retrieved by OPC UA over the ad-hoc channel. We provide more details of the discovery process in [9]. The latency of a Profinet IO system (from a controller to a device) in an installation with line topology – which is very common in industrial manufacturing systems – can be expected to be less than 35 ms [13]. The delay of the middleware components is smaller than 10 ms. So, the `ExchangePrognosis` message reaches the other robot within the maximum transmission time of 336 ms. In conclusion, we have successfully applied our model-driven development method and the network architecture to our example of the cooperating delta robots. However, we still have to improve the quality of our detection of the ball position by piezo force sensors. Currently, we only support seven strikes per game as the ball detection does not consider the stiffness of the racket plate. As the mass of the ball is small in comparison to the mass of the plate, it is difficult to detect the ball position on the plate. Small variances influence the precision of the model-based observer and the prognosis. As a result, the cooperating delta robots currently drop the ball after several strikes.

IV. EXPERIENCE

In the following, we state the challenges we have faced and the lessons that we learned while developing the software of the cooperating delta robots in the context of the ENTIME and the it's OWL projects.

A. Software Developing Challenges

The development of CCPS gets more complex because of such advanced cooperation and control behavior. In our opinion, their development has to face the following five challenges that extend and refine the modeling challenges presented by Schäfer and Wehrheim [14].

(I) The application architecture gets more complex because it has to monitor the sensors, analyze the sensor values, plan appropriate actions, and execute them by controlling the actuators. The application consists of several software components that handle these different tasks. Properties of sensors, actuators, and ports influence how the application architecture is being built. "This concerns, in particular, timing aspects which must be specified in the model as well, to be able to analyze the system model appropriately and to fully automatically generate code from the model specification." [14]

(II) The controller has to consider information from another system and its control strategy. The information from the other system could be available only at discrete points in time and not continuously/sampled. In our example, the prognosis message of the other robot has to arrive before the robot calculates its strike trajectory.

(III) The (message-based) communication is getting much more complex. The communication depends on physical dynamics, calculations from the controller, environmental changes, and the current internal state of the system. It is not only important that the system works in a logically correct way. In addition, the timing must be considered accordingly. Therefore, such interacting systems and distributed control loops must be able to rely on real-time communication networks to coordinate and control their complex and time-dependent behavior. In our example, the maximum message delay depends on the minimal ball flight duration.

(IV) While developing the system, control engineers have to consider different system parts like the application and the underlying communication infrastructure. In our example, the distributed control has to handle long dead-times as a result of the event-based sensoring and the communication delay, which may lead to a decreased performance of the control loop.

(V) A discipline-spanning collaboration between software engineers, control engineers, and communication engineers is required to develop these CCPS. The software engineers develop the application architecture and the coordination behavior, the control engineers develop the controller concept, and the communication engineers provide the concept for communication. Thus, the different engineers are expert for their domain, but do not necessarily have a holistic understanding of the system. A system engineer might have a holistic understanding, but lack discipline knowledge.

B. Lessons Learned

Considering the software architecture, all three intelligent network layers must be considered from the beginning. Additionally, the interfaces and assumptions within each layer are as important as the interface and assumptions between the layers, e.g., the quality-of-service assumptions of the application layer that the underlying layers have to fulfill. The middleware and connectivity layer fulfill these assumptions and allow a decoupling of the application and the real-time communication system.

A discipline-spanning development process like the one of Heinzemann et al. [15] improves the efficiency of the development. Before developing the software, a conceptual design phase where all disciplines participate is important, e.g., to identify additional real-time requirements. Moreover, early simulations via idealized models help to determine the feasibility of the system and to identify requirements imposed by the environment.

Considering the project management, it is important that the engineers can use the languages and tools of their own discipline. Small task forces should be established where each participant represents one discipline and one system engineer that moderates. These task forces need to meet often and have to document decisions and reasons that led to these decisions. To force progress, the project manager needs to define hard regular deadlines. Team building exercises help to reduce prejudices against other disciplines. Training the engineers helps but does not necessarily imply that all engineers follow the process and architecture. One good training method is to review successful projects. In our opinion, adhering to these lessons helps to improve the development of a CCPS but is no silver bullet. Some of these lessons confirm general project management guidelines.

V. Conclusion

This paper presents an experience report on the development for CCPS using two cooperating delta robots as an example. Due to the tight integration of different system parts, the development process of such systems is very complex. Hence, model-driven development methods, as an engineering approach, have been deployed. In combination with a new intelligent network architecture with self-configuration capabilities, an interdisciplinary team of engineers was able to specify and develop the system in an efficient way.

Engineers need to consider the identified challenges during the software development of a CCPS. The tight coupling of different parts within a CCPS and heterogeneous requirements from different disciplines makes the development more complex than the sum of its parts. This challenge is also known as emergence [16]. New insights after completion of this work are the necessity to consider all three intelligent network layers from the beginning. Moreover, a discipline-spanning development process increases the efficiency and avoids time consuming and costly iterations in the final phase of the project.

Currently, the implementation and integration of all system components of the cooperating delta robots is the result of the first iteration. Future work will incorporate the lessons learned into the development process of all different engineering disciplines, leading to a further improvement of it. Another important aspect of our future work will be the evaluation of the whole system in specific case studies. This will lead to a validation of the system with respect to the identified requirements.

ACKNOWLEDGMENT

This work was partially funded by the German Federal Ministry of Education and Research (BMBF) within the Leading-Edge Cluster "Intelligent Technical Systems OstWestfalenLippe" (it's OWL) and by the state of North Rhine-Westphalia (NRW), Germany, within the project 'ENTIME: Design Methods for Intelligent Mechatronic Systems'.

REFERENCES

[1] L. Dürkop, J. Imtiaz, H. Trsek, L. Wisniewski, and J. Jasperneite, "Using OPC-UA for the Autoconfiguration of Real-time Ethernet Systems," in *11th IEEE International Conference on Industrial Informatics (INDIN)*, July 2013.

[2] P. Derler, E. Lee, and A.-S. Vincentelli, "Modeling Cyber-Physical Systems," *Proceedings of the IEEE*, vol. 100, no. 1, pp. 13–28, Jan 2012.

[3] L. Zhang, H. Sun, X. Ma, C. Xu, and J. Lu, "Challenges in Developing Software for Cyber-physical Systems," in *Proceedings of the 5th Asia-Pacific Symposium on Internetware*, ser. Internetware '13. New York, NY, USA: ACM, 2013, pp. 15:1–15:10. [Online]. Available: http://doi.acm.org/10.1145/2532443.2532450

[4] M. Ricken and B. Vogel-Heuser, "Modeling of Manufacturing Execution Systems: An interdisciplinary challenge," in *15th IEEE Conference on Emerging Technologies & Factory Automation (ETFA)*, Sept 2010.

[5] P. Hošek, T. Pop, T. Bureš, P. Hnětynka, and M. Malohlava, "Comparison of component frameworks for real-time embedded systems," in *Component-Based Software Engineering*, ser. LNCS. Springer Berlin Heidelberg, 2010, vol. 6092, pp. 21–36. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-13238-4_2

[6] K.-K. Lau and Z. Wang, "Software Component Models," *IEEE Transactions on Software Engineering*, vol. 33, no. 10, pp. 709 –724, october 2007.

[7] A.-W. Colombo, S. Karnouskos, and J.-M. Mendes, "Factory of the Future: A Service-oriented System of Modular, Dynamic Reconfigurable and Collaborative Systems," in *Artificial Intelligence Techniques for Networked Manufacturing Enterprises Management*. Springer, 2010, pp. 459–481.

[8] J. Gausemeier, F.-J. Rammig, and W. Schäfer, *Design Methodology for Intelligent Technical Systems*, ser. Lecture Notes in Mechanical Engineering. Springer, 2014.

[9] L. Dürkop, H. Trsek, J. Otto, and J. Jasperneite, "A field level architecture for reconfigurable real-time automation systems," in *10th IEEE International Workshop on Factory Communication Systems (WFCS)*, May 2014.

[10] J. Gausemeier, A. Trächtler, and W. Schäfer, *Semantische Technologien im Entwurf mechatronischer Systeme: Effektiver Austausch von Lösungswissen in Branchenwertschöpfungsketten*. Carl Hanser, 2014.

[11] S. Becker et al., "The MechatronicUML Method: Model-Driven Software Engineering of Self-Adaptive Mechatronic Systems," in *36th International Conference on Software Engineering (ICSE)*. ACM, May 2014.

[12] Thilo Sauter, "The Three Generations of Field-Level Networks—Evolution and Compatibility Issues," *IEEE Transactions on Industrial Electronics*, vol. 57, no. 11, pp. 3585–3595, 2010.

[13] H. Kleines, S. Detert, M. Drochner, and F. Suxdorf, "Performance Aspects of Profinet IO," *IEEE Transactions on Nuclear Science*, vol. 55, no. 1, pp. 290–294, Feb. 2008.

[14] W. Schäfer and H. Wehrheim, "The Challenges of Building Advanced Mechatronic Systems," in *Future of Software Engineering*, ser. FOSE '07. IEEE Computer Society, May 2007, pp. 72–84.

[15] C. Heinzemann, O. Sudmann, W. Schäfer, and M. Tichy, "A Discipline-Spanning Development Process for Self-Adaptive Mechatronic Systems," in *International Conference on Software and System Process (ICSSP)*, May 2013.

[16] G. H. Lewes, *Problems of life and mind (First Series)*. London: Trübner, 1875, vol. 2, no. ISBN 1-4255-5578-0.