Automatic Parameterization of Automation Software for Plug-and-Produce

Jens Otto [1] and **Oliver Niggemann**[1,2]
[1]Fraunhofer IOSB-INA, 32657 Lemgo, Germany
email: {jens.otto, oliver.niggemann}@iosb-ina.fraunhofer.de
[2]inIT-Institut Industrial IT, University of Applied Sciences, 32657 Lemgo, Germany
email: oliver.niggemann@hs-owl.de

Abstract

Cyber-Physical Production Systems' (CPPSs) main feature is adaptability, i.e. they can adapt quickly to new production goals such as new products or product variants. Today, the bottleneck of such approaches is the automation system, which still requires high manual engineering efforts for every adaptation step. Many recent solutions for a more adaptable automation software have focused on the automatic orchestration of software systems: for a new product and production configuration, a software solutions is created by putting together reusable software components. But such solutions come with a price: reusable software components must be, by definition, applicable to wide range of configurations. For this, software components come with free parameters that must be set according to the current configuration. Typically, the main problem is not the orchestration of software components but their correct parameterization.

This paper presents, to the best of our knowledge for the first time, a solution to the parameterization problem of adaptable, CPPS-enable software systems. Due to the nature of CPPSs, no direct computation of parameters is possible. Instead, an iteration-based approach using a model of both the plant and the automation system is needed. An example from process industry illustrates the ideas.

Introduction

Cyber-Physical Systems (CPS) combine computation with physical processes (Lee 2008). A subset of CPS are Cyber-physical Production Systems (CPPS) whose main feature is adaptability, i.e. the system automatically adapts to new production goals such as new products or product variants (Zimmermann et al. 2008; Reinhart et al. 2010). But while the plants are becoming more and more modular and adaptable, the automation software is still changed manually. Therefore, the research question is how the next generation of automation software can fulfill the promise of adaptability and plug-and-produce.

Classical vs. CPPS-enabled Automation Software Development

Classical automation has a long established development process: as shown in figure 1, normally informal descriptions –often provided verbally– of the product and the production process are the starting point. The automation engineer then uses automation software engineering tools to write software for the controllers, often separately for each controller. This software is written using classical procedural languages such as IEC 61131-3.

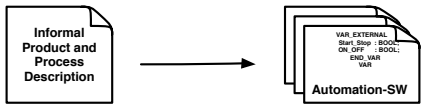


Figure 1: Classical Automation Development Process.

CPPS-enabled automation software development works differently: the automation engineer no longer writes the automation software manually, but solely defines the automation goal (product, process), with the result that the engineering no longer defines the "how" but the "what". Production goals are product features, costs, throughput or the energy consumption. As, in contrast to the classical automation, these automation goals usually remain unchanged during system modifications (e.g. replacement of a system module), the automation engineer is no longer continually involved. Moreover, it is easier to define the automation goal (e.g. in the form of a description of the final product) than the complete automation logic (i.e. the software). This results in a reduction of the complexity as perceived by the automation engineer.

As a prerequisite for such adaptability, a transparent communication connection between the production modules and their respective controllers has to be established, which is done in the communication layer. This includes tasks such as physically connecting the hardware and software modules and specifying communication protocols. For this, approaches for auto-configuration (Reinhart et al. 2010; Dürkop et al. 2013) and middleware solutions (Cannata, Gerosa, and Taisch 2008; Deugd et al. 2006; Zoitl et al. 2007; Schleipen 2008) are researched. For this paper, we presume that the automation platform and the communication layer provide these features and therefore support adaptability.

Figure 2 now outlines the main steps of a CPPS-enabled automation software development: First, a production pro-

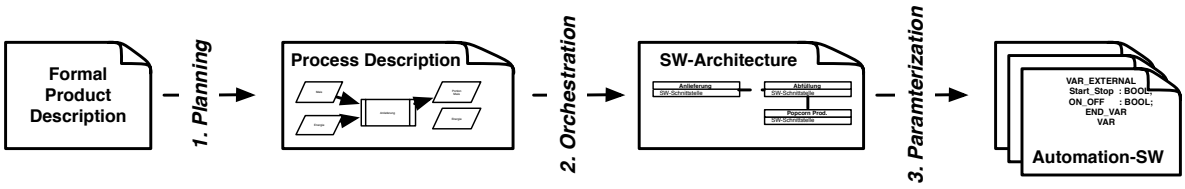


Figure 2: CPPS Automation Development Process.

cess is computed, which transforms the provided raw materials into the required product. In the second step, a software architecture is generated that automates this process. In the last step, the software is parameterized. In the following these steps are explained in detail.

Step 1: Computation of a production process
Ultimately, the final goal of all activities is the product. Manufacturing engineering, automation and mechanical engineering are all only means to this end. So as a vision, CPPS-enabled automation should start from one single set of requirements: product and raw material descriptions. Planning and scheduling methods are then used to generate a suitable production process (Anis, Schaefer, and Niggemann 2014; Bannat et al. 2011; Loskyll et al. 2011), these solutions use the product and raw material descriptions and also descriptions of production ressources (robots, transportation systems, reactors, etc.) and compute an optimal sequence of production steps, i.e. the production process.

The reader may note that such a planning step requires that production steps can still be rearranged. Since the plant, from a mechanical point of view, is usually already installed at the time of automation software creation, a corresponding dynamic plant layout is required—a prerequisite seldom true in real-world production.

Step 2: Orchestration of a software architecture
Many research projects take such a production process as an input and compute a suitable software architecture (Loskyll et al. 2011; Pfrommer, Schleipen, and Beyerer 2013). Normally, this is done by combining pre-defined software components (Witsch and Vogel-Heuser 2009; Papakonstantinou, Sierla, and Koskinen 2011) whereat software components usually correspond to production modules. This step is called software orchestration.

If software orchestration is used, the process description has to be some-how related to the description of the software components. In most projects, this is done by creating two taxonomies of process steps and of software components– often formally modeled as ontologies. Elements of both taxonomies are then mapped onto each other, allowing for an automatic choice of software components for process steps. E.g. a production step "transportation" is mapped onto a software component "transporting".

The reader may note that the behavior of software components and process steps are modeled as symbols/ontologies only and no dynamic system features are therefore captured, i.e. it is an open research question whether this type of modeling is sufficient for real-world production plants.

Only few projects try to generate the software from the scratch instead of using pre-defined software components (Henning et al. 2014; Otto, Henning, and Niggemann 2014).

Step 3: Parameterization of a software architecture
The usage of software components in step 2 comes with a price: the automation software must be abstracted into reusable and capsulated software components. While in classical automation, software is written for each production scenario individually, such software components must be now applicable to a large set of scenarios. This is only possible if software components can be adapted to individual scenarios by means of free parameters in the software. Examples for parameters are machine timing, controller parameters such as PID, physical dimensions of products, sensitivity of diagnosis functions, etc.

So the usage of software orchestration and of software components raises a new key research question: *How can we choose automatically optimal parameters of the automation software for a specific production scenario?* This paper tries to give one possible answer.

Software Parametrization

As outlined before, software parametrization is a key research question for CPPS-enabled automation. In general, two types of software parametrization approaches can be differentiated: Figure 3 shows the first type: Based on product

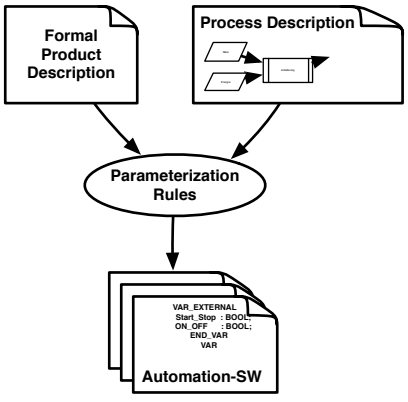


Figure 3: Rule-based Software Parameterization.

and process features, the software can be parameterized by means of simple rules. An example is a robot that is adapted to physical dimensions of the product or a filling station that chooses the amount of liquid according to bottle sizes. One

might notice that this approach only works *(i)* if the production machine can be adapted locally and local parameters do not influence other parts of the production line and *(ii)* if the rules are rather straightforward and simple.

If the rule-based approach does not work, an iterative approach as shown in figure 4 must be used: In an optimization loop, sets of parameters are analyzed virtually using a model of both the plant and the automation system. If a good set of parameters is identified virtually, this parameter set is used for the real world plant.

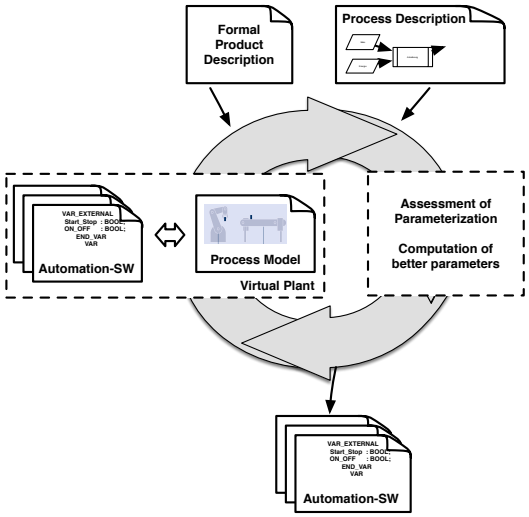


Figure 4: Iteration-based Software Parameterization.

Unlike the rule-based approach, this approach allows for a system-wide optimization of parameters. This approach is therefore used in this paper.

Smart Products and Agents

In this context, the approach of smart products must be mentioned (Loskyll et al. 2011; Abramovici and Stark 2013): Smart products store product IDs, product information and process information on the product itself—e.g. in form of an RFID chip. Often the smart product contains software parameters such as physical dimensions, i.e. it supports the rule-based parameterization described above. In other cases, it contains a product-specific production process, this requires a dynamic plant layout as described in step 1 of figure 2.

Agent-based automation software (Ulewicz, Schutz, and Vogel-Heuser 2012; Urbas et al. 2011) is another way to solve the software parameterization challenge. This approach also uses distributed software components–called agents–but the parameter optimization is done online within the agents.

Solution Approach

In this section, our approach for an iteration-based software parameterization is introduced.

The iteration-based software parameterization requires a model to simulate the plant. More precisely, the simulation model must be able to simulate the physical behavior and the automation software. This simulation model allows the evaluation of parameter configurations virtually. Furthermore, a measure rating the result of an evaluation and a method to find a better parameter configuration is defined.

In the following section, a formal problem definition is given, the used simulation model for evaluation of parameter configuration is explained and the method to find better parameter configurations is introduced.

Problem Definition

An automation software $\mathcal{A}$ has a set P of parameters, where P is defined as $P = \{p_1, \ldots, p_i, \ldots, p_k\}$. These parameters adapt the orchestrated software components to a specific production scenario. An element p_i of P is a tuple $p_i = (v_i, r_i)$, where v_i is the parameter's name and $r_i = (v_{min}, v_{max})$ is the range of the possible values for v_i. Examples for parameters are machine timing, controller parameters such as PID, physical dimensions of products, sensitivity of diagnosis functions, etc. For an instance of the software, a parameter configuration $\theta = \{\theta_1, \ldots, \theta_i \ldots, \theta_k\}$ where $\theta_i \in r_i$ is the value of the parameter p_i. We denote r_i by $range(p_i)$. The overall set of possible parameter configurations is defined as $\Theta = r_1 \times \ldots \times r_i \times \ldots r_k$.

The problem can be defined as follows:
Find a good parameter configuration θ in the parameter configuration space Θ for an automation software $\mathcal{A}$ using a simulation model $\mathcal{M}$.

Next, the measure for a good parameter configuration is defined. Our hypothesis is that a plant operates optimal in a specific production scenario, if the production output rate is high, without using more energy as needed. This means that a good parameter configuration θ for an automation software $\mathcal{A}$ results in a high production output rate and an optimal energy consumption. So, the objective function is defined as:

$$\min_{\theta \in \Theta} f(\theta), f(\theta) = \frac{e(\theta)}{o(\theta)}, o : \Theta \rightarrow \mathbf{R}, e : \Theta \rightarrow \mathbf{R} \quad (1)$$

where $e(\theta) \in \mathbf{R}$ is the value of consumed energy and $o(\theta) \in \mathbf{R}$ is the production output rate in a specific time window t_{window}. For example, f defines how many bottles can be filled in 10 minutes and how much energy is needed for this. Both $e(\theta)$ and $o(\theta)$ are computed by analyzing the parameter set θ with the system model $\mathcal{M}$.

Simulation Model

The simulation model is formally defined as a hybrid automata, more precisely a hybrid timed automata. Hybrid automata describes both discrete and continuous dynamics (Alur et al. 1995). In (Niggemann et al. 2012), hybrid timed automata are identified as a crucial class of models for technical systems, such as production plants. It also introduces an algorithm called HyBUTLA, which can be used to learn a hybrid timed automaton automatically from observations. Furthermore, a compositional interchange format for hybrid systems exists, which allows the development of tools to describe production plants manually (van Beek et al. 2008).

In the automation domain, the time behavior and the energy consumption of the plant are relevant information. Mostly, they are the only measurable behavior, for example a water tank has only binary sensors to measure the filling level. The continuous behavior is hidden. So, our definition of a hybrid timed automaton has a function that returns the relative time and the used energy consumption for each transition (see definition 1).

Definition 1 *A hybrid timed automaton is a tuple* $A = (\Sigma, S, s_0, F, \delta, T)$*, where*

- Σ *is the alphabet,*
- S *is a set of states,*
- $s_0 \in S$ *is the initial state,*
- $F \subseteq S$ *is the set of final states,*
- $\delta : S \times \Sigma \rightarrow S$ *is the state-transition function, and*
- T *is a set of functions* $f : p \rightarrow (t, e)$ *that returns the relative time and energy consumption* e *of a transition, given the parameter* p*.*

For a better understanding, the simulation is encapsulated as a mathematical function: $m(\theta, t_{window})$. The parameters of m are the parameter configuration θ and a time window t_{window}. The time window describes that how long should the simulation runs. The return value of m is the optimization criteria f, which is defined above.

Parameter configuration

The exploration of the parameter configuration space Θ is separated into the following two algorithms:

Algorithm 1 samples $n \in \mathbf{N}$ initial parameter configurations I and evaluates each parameter configuration with the simulation, i.e. with function $m(\theta, t_{window})$. The value of t_{window} is use case specific. The function $uniform$ denotes a uniform sampling.

Algorithm 1: creates initial parameter configurations

Input: Set of parameters P, n number of initial parameter configurations, t_{window} the time window of the simulation
Output: a set $I \subseteq \Theta$ of parameter configurations

```
1 I ← ∅
2 for i ← 1...n do
3 |   θ ← ∅
4 |   foreach p ∈ P do
5 |   |   r ← range(p)
6 |   |   θ ← θ ∪ uniform(r)
7 |   v ← m(θ, t_window)
8 |   I ← I ∪ (θ, v)
9 return I
```

Algorithm 2 tries to find a good parameter configuration. The set I of initial parameter configurations is assigned to set H (line 1). The set H is used for a function approximation of the objective function (line 3). An optimization algorithm tries to find a good parameter configuration θ' (line 4). The expected parameter configuration θ' is evaluated by the simulation model (line 5). The set H of all evaluated parameter configurations is extended by θ' (line 6). The termination criterion is defined by the algorithm parameter k.

For the $optimize$ procedure, the basin-hopping algorithm is used (Wales and Doye 1997). The basin-hopping algorithm finds the global minimum of a function and is an efficient method for a wide variety of problems. The $approximate$ procedure is a Nearest-neighbour interpolation in N dimensions.

Algorithm 2: explores Θ

Input: Set I of initial parameter configurations, k the number of iterations, t_{window} the time window of the simulation, $approximate : H \rightarrow v'$, $optimize : f \rightarrow minf, min : H \rightarrow \theta_{opt}$
Output: θ_{opt}

```
1 H ← I
2 for i ← 1...k do
3 |   f ← approximate(H)
4 |   θ' ← optimize(f)
5 |   v ← m(θ', t_window)
6 |   H ← H ∪ (θ', v)
7 return min(H)
```

Results

In this section, a case study and our experimental results are presented. The case study shows how to create a simulation model with a real world scenario. The experimental results illustrate the gain of the algorithm in general by evaluating a standard test functions for optimization.

Case Study

The filling process, which is shown in figure 5, is used to show the creation of a simulation model with a real world scenario. The goal of this filling process is to fill water into bottles. The water must be heated to 70 degree Celsius to kill the bacteria before it can be filled. The filling process is realised by three technical resource: t_1: heater, t_2: filler and t_3: transporter. It consists of four actuators (a_1 – a_4) and four sensors (s_1 – s_4), which are connected by three IO-devices to a central programmable logic controller (PLC). The actuator a_1 is a heating element, s_1 measures the temperature of the water, whereby a_2 and a_3 are proportioning valves, s_2 and s_3 are filling level sensors and a_4 is a transporter with a light barrier s_4. The used field bus is PROFINET. The three technical resources can be substituted with other modules with the same or similar capabilities. The modular structure enables a simple modification of the system representing the general plug-and-produce idea.

The automation software has the following parameters: p_1 sets the electrical energy for heating element a_1, p_2 sets the valve opening a_2, p_3 sets the valve opening a_3 and p_4 set the speed of the transport a_4. For simplification, all ranges are normalized.

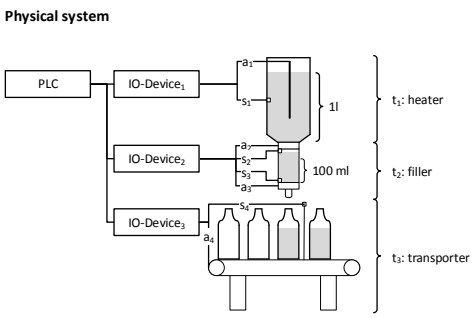


Figure 5: Filling process.

The hybrid timed automaton (shown in figure 6) of the filling process has four states. It starts with an enabled heating element a_1. If the specific temperature ($c_1 = 70$) is reached, the heating element is disabled and the first proportioning valve a_2 opens. If sensor s_2 is enabled ($c_2 = 1$), the proportioning valve a_2 is closed and proportioning value a_3 opens. If sensor s_3 is enabled ($c_3 = 1$), proportioning value a_3 is closed and the transport a_4 is activated. If sensor s_4 is enabled ($c_4 = 1$), the transport is deactivated. The process starts all over again.

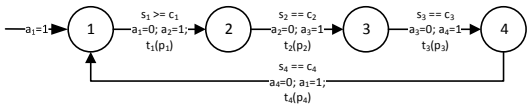
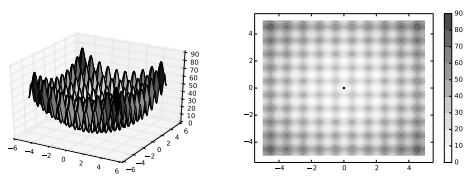


Figure 6: Hybrid timed automaton of the filling process.

Experimental Results

For the experimental results, the two dimensional rastrigin function is used. So, the set of parameters P has the elements p_1 and p_2. The range for p_1 and p_2 is set to $[-5...5]$.

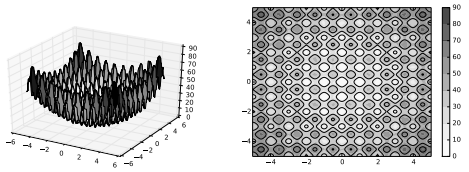
Figure 7 shows the rastrigin function on the left side and the contour of the rastrigin function on the right side. The global minimum at $[0, 0]$ is marked with a dot and is evaluated by the basin-hopping algorithm.



(a) Function (b) Contour

Figure 7: Two dimensional rastrigin function.

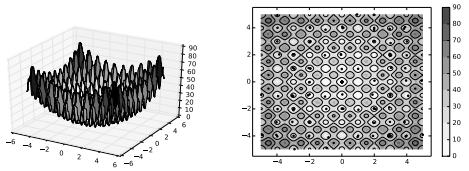
Figure 8 shows the approximated rastrigin function and the surface after algorithm 1 with $n = 10000$ initial parameter configuration.



(a) Function (b) Contour

Figure 8: Approximated rastrigin function after algorithm 1.

Figure 9 shows the approximated rastrigin function and the contour. The evaluated values are marked with dots. The algorithm 2 found an optimum function value $v = 0.08445$ at $p_1 = 0.02023$ and $p_2 = 0.00412$ after $k = 100$ iterations.



(a) Function (b) Contour

Figure 9: Approximated rastrigin function after algorithm 2.

Summary and Future Work

This paper presents a new approach towards more adaptable automation software for CPPSs. For such software, the question of finding optimal parameters for the software components is a key challenge. As a solution, a new iteration-based algorithm using a model of both the plant and the automation system is given. Our approach finds a suitable parameter configuration. The simulation model for evaluating the parameters is formalized as a timed hybrid automaton and a case study from a real world scenario is used to illustrate the solution approach.

Future steps will be the refinement of the algorithms and the application to other more complex use cases, for example to the Lemgo Smart Factory. Such further applications will allow for a optimized choice of optimization methods and for an evaluation of the used model formalism.

Acknowledgement

This research and development project is funded by the German Federal Ministry of Education and Research (BMBF) within the leading-edge cluster "Intelligent Technical Systems Ostwestfalen-Lippe" (it's OWL) and managed by the Project Management Agency Karlsruhe (PTKA). The author is responsible for the contents of this publication.

References

Abramovici, M., and Stark, R., eds. 2013. *Smart Product Engineering*, volume Proceedings of the 23rd CIRP Design Conference. Springer Verlag.

Alur, R.; Courcoubetis, C.; Halbwachs, N.; Henzinger, T. A.; Ho, P.-H.; Nicollin, X.; Olivero, A.; Sifakis, J.; and Yovine, S. 1995. The algorithmic analysis of hybrid systems. *THEORETICAL COMPUTER SCIENCE* 138:3–34.

Anis, A.; Schaefer, W.; and Niggemann, O. 2014. A comparison of modeling approaches for planning in cyber physical production systems. In *19th IEEE International Conference on Emerging Technologies and Factory Automation*.

Bannat, A.; Bautze, T.; Beetz, M.; Blume, J.; Diepold, K.; Ertelt, C.; Geiger, F.; Gmeiner, T.; Gyger, T.; Knoll, A.; Lau, C.; Lenz, C.; Ostgathe, M.; Reinhart, G.; Roesel, W.; Ruehr, T.; Schuboe, A.; Shea, K.; Stork genannt Wersborg, I.; Stork, S.; Tekouo, W.; Wallhoff, F.; Wiesbeck, M.; and Zaeh, M. 2011. Artificial cognition in production systems. *Automation Science and Engineering, IEEE Transactions on* 8(1):148–174.

Cannata, A.; Gerosa, M.; and Taisch, M. 2008. Socrades: a framework for developing intelligent systems in manufacturing. In *IEEE International Conference on Industrial Engineering and Engineering Management*, 1904 – 1908.

Deugd, S.; Carroll, R.; Kelly, K.; Millett, B.; and Ricker, J. 2006. Soda: Service-oriented device architecture. In *IEEE Pervasive Computing*, 94–96.

Dürkop, L.; Imtiaz, J.; Trsek, H.; Wisniewski, L.; and Jasperneite, J. 2013. Using opc-ua for the autoconfiguration of real-time ethernet systems. In *11th International IEEE Conference on Industrial Informatics*, 248–253.

Henning, S.; Otto, J.; ; Niggemann, O.; and Schriegel, S. 2014. A descriptive engineering approach for cyber-physical systems. In *19th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA 2014)*.

Lee, E. A. 2008. Cyber physical systems:design challenges. Technical Report UCB/EECS-2008-8, EECS Department, University of California, Berkeley.

Loskyll, M.; Schlick, J.; Hodek, S.; Ollinger, L.; Gerber, T.; and Pirvu, B. 2011. Semantic service discovery and orchestration for manufacturing processes. In *Emerging Technologies Factory Automation (ETFA), 2011 IEEE 16th Conference on*, 1–8.

Niggemann, O.; Stein, B.; Vodenčarević, A.; Maier, A.; and Kleine Buning, H. 2012. Learning behavior models for hybrid timed systems. In *Twenty-Sixth Conference on Artificial Intelligence (AAAI-12)*.

Otto, J.; Henning, S.; and Niggemann, O. 2014. Why cyber-physical production systems need a descriptive engineering approach – a case study in plug & produce. *Procedia Technology* 15(0):295 – 302. 2nd International Conference on System-Integrated Intelligence: Challenges for Product and Production Engineering.

Papakonstantinou, N.; Sierla, S.; and Koskinen, K. 2011. Object oriented extensions of iec 61131-3 as an enabling technology of software product lines. In *Emerging Technologies Factory Automation (ETFA), 2011 IEEE 16th Conference on*, 1–8.

Pfrommer, J.; Schleipen, M.; and Beyerer, J. 2013. Pprs: Production skills and their relation to product, process, and resource. In *Emerging Technologies Factory Automation (ETFA), 2013 IEEE 18th Conference on*, 1–4.

Reinhart, G.; Krug, S.; Huttner, S.; Mari, Z.; Riedelbauch, F.; and Schlogel, M. 2010. Automatic configuration (plug & produce) of industrial ethernet networks. In *Industry Applications (INDUSCON), 2010 9th IEEE/IAS International Conference on*, 1–6.

Schleipen, M. 2008. Opc ua supporting the automated engineering of production monitoring and control systems. In *IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, 640–647.

Ulewicz, S.; Schutz, D.; and Vogel-Heuser, B. 2012. Design, implementation and evaluation of a hybrid approach for software agents in automation. In *Emerging Technologies Factory Automation (ETFA), 2012 IEEE 17th Conference on*, 1–4.

Urbas, L.; Krause, A.; Pech, S.; and Göhner, P. 2011. Function allocation for multi-agent systems and middleware in industrial automation systems. *16th IEEE Internation Conference on Emerging Technologies and Factory Automation (ETFA 2011)*.

van Beek, D. A.; Reniers, M. A.; Rooda, J. E.; and Schiffelers, R. R. 2008. Concrete syntax and semantics of the compositional interchange format for hybrid systems. In *Proc. 17th IFAC World Congress, Seoul, Korea*.

Wales, D. J., and Doye, J. P. K. 1997. Global optimization by basin-hopping and the lowest energy structures of lennard-jones clusters containing up to 110 atoms. *The Journal of Physical Chemistry A* 101(28):5111–5116.

Witsch, D., and Vogel-Heuser, B. 2009. Close integration between uml and iec 61131-3: New possibilities through object-oriented extensions. In *Emerging Technologies Factory Automation, 2009. ETFA 2009. IEEE Conference on*, 1–6.

Zimmermann, U. E.; Bischoff, R.; Grunwald, G.; Plank, G.; and Reintsema, D. 2008. Communication, configuration, application - the three layer concept for plug-and-produce. In *ICINCO-RA (2)*, 255–262.

Zoitl, A.; Strasser, T.; Hall, K.; Staron, R.; Sünder, C.; and Favre-Bulle, B. 2007. The past, present, and future of iec 61499. *Holonic and Multi-Agent Systems for Manufacturing* 4659:1–14.