A Descriptive Engineering Approach for Cyber-Physical Systems

Steffen Henning
and Oliver Niggemann
inIT - Institute Industrial IT
OWL University of Applied Sciences
Lemgo, Germany
Email: {steffen.henning, oliver.niggemann}
@hs-owl.de

Jens Otto
and Sebastian Schriegel
Fraunhofer IOSB-INA
Industrial Automation
Lemgo, Germany
Email: {jens.otto, sebastian.schriegel}
@iosb-ina.fraunhofer.de

Abstract—**Plug and produce (PnP) aims at reducing system engineering effort. Therefore several PnP aspects have to be solved. This paper gives an overview and classifies process types and PnP aspects. Furthermore it identifies coupled processes as the most challenging ones and provides a new engineering approach for such systems. A new two-step descriptive engineering approach is applied to automatically synthesise control code from a given product specification. The main idea of the approach is to use a descriptive view rather than a prescriptive. This means that the engineer specifies *what* he wants to produce and no longer *how* he wants to produce it. Thus, it reduces engineering effort and releases more resources to optimize the process.**

I. Motivation and State of the Art

Plug and produce (PnP) is a major goal in the research area of Cyber-Physical Production Systems: Its goal is the fast adaptation of production systems to new requirements such as new product variants [1].

A major bottleneck for the implementation of PnP lies within the automation system: Significant manual engineering efforts are needed after the production system layout has been modified. For example, the network configuration must be modified, new automation software has to be implemented and HMI layouts must be changed. So from an automation perspective, the core question of PnP is how the automation software and the automation configuration can be adapted automatically to new or modified production systems.

Different types of production systems require different solution approaches in the automation. Two process types can be identified: coupled and non-coupled processes.

For discrete manufacturing industry, such as car manufacturing, production modules can work independently. Resulting in no need for communication between production modules, overall control and synchronization between these modules. These processes are non-coupled or in other words: There is no function in the structure of the production modules. The no-function-in-structure principle goes back to de Kleer [2]. Each module performs its task autonomously, independent of where it is located. Here, a solution may be smart products, i.e. products that carry their process description with them [3].

For other types of systems, e.g. bulk good production or process industry, an overall control is needed. This is also the case if the process must be adapted frequently during a production cycle of a product, e.g. to optimise the utilisation of the factory. These processes are coupled, meaning that the system behaviour depends on how the modules are combined. For such scenarios, PnP requires that a new overall control is generated. In this paper, strategies for such scenarios are given.

Also, PnP requires different solutions on different abstraction layers, which are shown in Figure 1. The three layers are a communication layer for basic connectivity, a modularisation layer for modular design and interfaces and a control code synthesis layer for the superordinated control software applications.

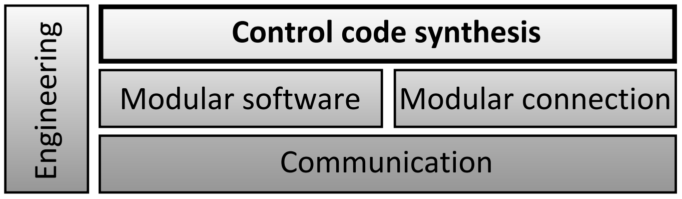

Fig. 1. Layers of plug and produce

As a first step for PnP, a communication connection between the production modules has to be established, which is done in the communication layer. This includes tasks such as physically connecting the modules and specifying communication protocols. In this layer, approaches for auto-configuration [4] are researched.

Secondly, to enable PnP, modular components are required, which is represented by the middle layer. Most hardware components are already constructed in a modular way, such that they are encapsulated from other modules. This also has to be done for the software by modularising the control code. Additionally, interfaces have to be defined to allow communication between modules. In this layer, works on distributed engineering [5] and modelling [6] are being researched.

The top layer is the control code synthesis, on which our approach will focus. In general, it represents approaches to generate the control code for the production system. Control code synthesis is an important aspect of PnP, because a manual change of control code after system modifications is no longer a feasible option. The systems become more complex and so does the control code. For this layer, three main approaches exist, depending on the process type: Service orchestration [7] for non-coupled processes and agent-based [8] and superordinated control [9] for coupled processes. Service orchestration runs into problems if an overall control is required, since only functionalities are concatenated and no connection logic is specified. Agent-based systems learn cooperation and coordination but this takes time. Descriptive engineering can synthesise overall control code.

II. Descriptive Approach

In contrast to a prescriptive approach, a descriptive approach specifies the goal of a task, rather than the individual steps of the task to reach the goal. In a manufacturing context, this means that instead of programming the behaviour of a production system, only the goal of the production system is specified, i.e. the product. The structure of our descriptive engineering approach is shown in Figure 2. Compared to function orchestration approaches, like [7] which start in level 1, our approach raises the abstraction by one level. The novelty of our approach is to use only the product descriptions of level 2 to automatically deduce the production process and then the control code.

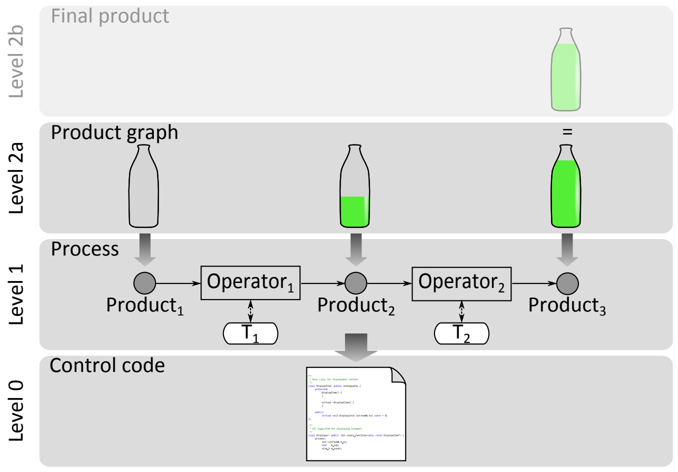

Fig. 2. Descriptive engineering with its design levels: product definition, process model, control code

The starting point for our approach is the intermediate product level (level 2a). However, towards a future engineering approach for PnP, level 2b is the next logical step. Next, the three levels will be explained in detail.

Level 2 conceals the two levels below and defines the final product and intermediate products. Since a product is the only reason for any production system, the final product contains all information that is needed to construct it. A product can formally be defined as a rule set that unravels all possible ambiguities that might occur during construction:

Definition 1 (Product): A product p is defined as $p \in A_1 \times A_2 \times \ldots \times A_n$, where A_i is the set of all possible values for the i-th attribute. Special products are final products P_{final}, which represent the goal of the production process. Products can also be bound to other products to form composite products, e.g. $p_c = p_1 \cup p_2$. A list of products and their corresponding values can be defined alongside the national German standard DIN 4002.

Example 1 (Product): An example of a product would be a bottle, which can be defined by: volume = $0.1l$ and basic material = “glass”. A binding is used to denote that two or more products are bound together to form a new, composite product. An example for a binding would be a filled bottle, which consists of the individual products: bottle and water. This is denoted by a binding “$\cup$” between the two products, i.e. filled bottle = bottle $\cup$ water.

In level 2a, a product graph is defined, which represents the order in which the product is modified. This graph connects all products together such that every product has a path to the final product.

Definition 2 (Product graph): A product graph is a directed graph $G = (P, T)$, with P being the products and $T \in P \times P$ being the transitions between the products. A product transition $t = p_1 \rightarrow p_2$ denotes that the source product p_1 can be transformed into the target product p_2.

Example 2 (Product graph): The most simple product graph would consist of one input product with a transition leading to the final product. This product graph would represent a single product transformation.

Level 1 adds meaning to the transitions between the products of level 2a, such that a graph edge represents a product modification. It represents the production process which consists of steps and their corresponding input- and output-products. The process model we use is syntactically defined in German guide line VDI/VDE 3682. An example of a process can be seen in Figure 3. Each individual process step consists of an operator, which represents the functionality, i.e. the product modification, and at least one input product and one output product. Furthermore it is related to a technical resource that represents the actual hardware of the process.

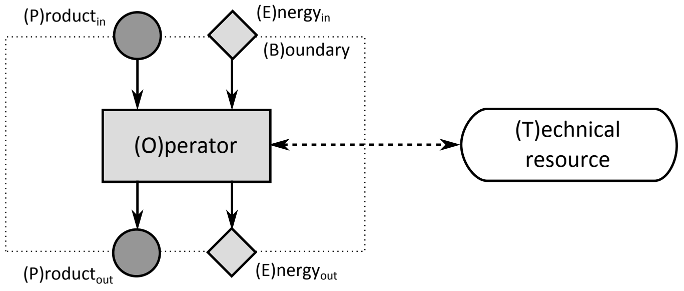

Fig. 3. Graphical description of a process from VDI/VDE 3682

In the following definition, the VDI/VDE 3682 model is formally specified. Elements that are not required for our approach, like the boundary and energy are omitted here and no extension to the guide line is required. However, the guide line offers a means of modelling them as well if desired.

Definition 3 (Process): A *process* is defined as the ordered sequence of multiple process steps. It is denoted as a triple $process = (P, O, T)$, consisting of products P, operators O and technical resources T. Products are defined in Definition 1 and can be input (P_{in}) and output products (P_{out}), with $P_{in}, P_{out} \subseteq P$. Operators perform the product transformation and have a modifier. Modifiers can change attribute values, i.e. increase ">" and decrease "<" values or product bindings, i.e. combine "$\cup$" and separate "$\overline{\cup}$". Technical resources represent the hardware, that realises the operator.

Example 3 (Process): An example process is heating, e.g. to heat a liquid. This process has a liquid of a certain temperature as input and another liquid with a higher temperature as output.

Level 0 represents the target level, i.e. the control code that runs directly on the controller. This is the basic level on which the two advanced levels (1 and 2) are built on. For most cases this is already formally defined in standards like IEC 61131-3 and is not the focus of this paper.

III. SYNTHESIS

The required control code synthesis steps can be identified from Figure 2. Between every two levels a transformation step is necessary. Since we have not dealt with the transformation from level 2b to level 2a yet, two transformations remain: Algorithm 1 creates a process model from a product (P) and a set of available processes, i.e. it transforms level 2a into level 1. Algorithm 2 generates control code from the previously created process model, i.e. it transforms level 1 into level 0.

A. Process Generation

Algorithm 1 creates a $Process$ from the set of all involved products (P) and a set of available $Processes$.

```
1: for all $(p_a, p_b) \in \{P : p_a \rightarrow p_b\}$ do
2:     $Modifier \leftarrow$ GETMODIFIER$(p_a, p_b)$
3:     $step \leftarrow$ GETPROCESS$(Modifier)$
4:     CONNECT$(Process, step)$
5: end for
6: return $Process$
```

Algorithm 1 takes a sequence of products and a set of available processes as input. It then computes a valid process model that can produce the product. In line 1, all connected product pairs are iterated, i.e. all products and their corresponding successors. The function call *getModifier* in line 2 looks for attribute and binding changes and returns the modifiers necessary to perform these changes. Next, *getProcess* returns a process step based on the modifiers in line 3. Then in line 4, the newly found process step is added to the overall process. The function *getModifier* returns the required modifiers for any two given products. It therefore compares all attributes and product bindings. If a discrepancy is found, a corresponding modifier is added.

B. Control Code Generation

Algorithm 2 constructs possible paths from the process model. A path is one possible concrete cycle of the production system, e.g. the normal behaviour. This can be done by a depth first search with loop detection. From these paths sequences are generated and their timing is estimated. A sequence is based on a path with additional timing information on when to switch between production system states. The timing estimation is not part of this paper, here a constant timing for each state is assumed. One timing approach with pre-learned timed automata

Algorithm 2 creates control code from a multiple execution path $Process$

```
1: for all $path \in$ DEPTHFIRSTSEARCH$(Process)$ do
2:     $sequences_{path} \leftarrow$ GETSEQUENCES$(path)$
3:     ADD$(sequences_{sys}, sequences_{path})$
4: end for
5: $h \leftarrow$ HYBUTLA$(sequences_{sys})$
6: $code \leftarrow$ GENERATECODE$(h)$
7: return $code$
8:
9: function GETSEQUENCES$(Path)$
10:     for $i \leftarrow 1, |Path|$ do
11:         $state_t \leftarrow (x_1, \ldots, x_n) | x_{1\ldots n} = 0, x_i = 1$
12:         $S \leftarrow S \cup$ PREDICT$(state_t)$
13:     end for
14:     return $S$
15: end function
```

is described in [10]. The HyBUTLA algorithm [11] uses the set of sequences as input to learn a combined automaton. The combined automaton contains the information on all process paths that were entered. From this automaton, the source code can be generated [12].

IV. CASE STUDY

Our approach was tested on the Lemgo Model Factory (LMF). It is a modular production system that can fill maize or water into small bottles. Furthermore, it can produce popcorn.

To start the engineering, a product graph is required as explained in section II. Exemplary product definitions for the LMF are shown in the product graph in Figure 4. Two specialities can be observed in the product graph: Firstly, there are two final products, i.e. two products that have no outgoing edge, so the production system can produce two products. And secondly, it is a cyclic graph, since p_2 and p_4 can be transformed into the composite product p_5, which can be transformed back into p_7 and p_8. The same holds for $(p_2, p_3) \rightarrow p_6 \rightarrow (p_7, p_{10})$. Thus a cycle in the product graph could be $p_2 \rightarrow p_5 \rightarrow p_2 \rightarrow \ldots$ followed by a path to the final product p_9.

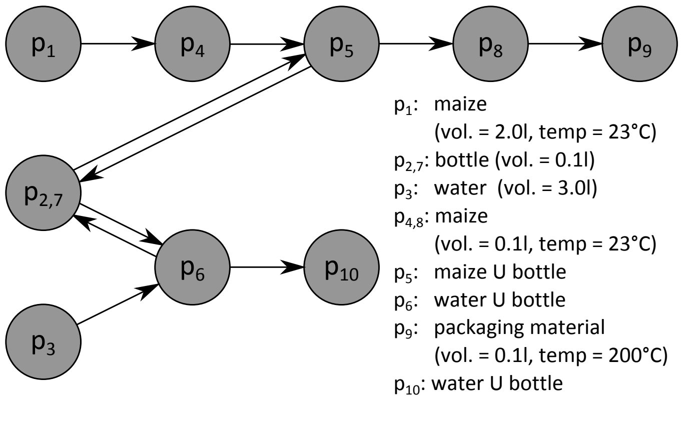
p1
p4
p5
p8
p9
p2,7
p6
p10
p3
p1: maize (vol. = 2.0l, temp = 23°C)
p2,7: bottle (vol. = 0.1l)
p3: water (vol. = 3.0l)
p4,8: maize (vol. = 0.1l, temp = 23°C)
p5: maize U bottle
p6: water U bottle
p9: packaging material (vol. = 0.1l, temp = 200°C)
p10: water U bottle

Fig. 4. Product graph of the LMF

With the product graph available, the next step is to derive the process from the product graph. Therefore, the set of processes should at least include the following process steps from VDI 2860 and DIN 8580: divide, combine, move and heat. The process model, as result of Algorithm 1 for the LMF is shown in Figure 5.

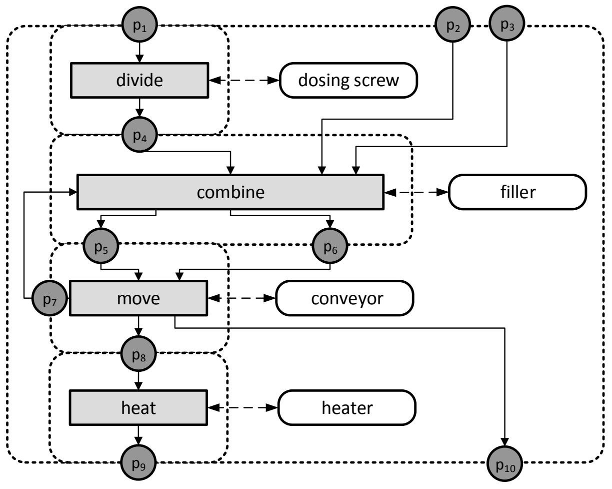

Fig. 5. Derived LMF process model

Algorithm 2 creates paths and then sequences of this process, which are possible ways the process can run. For example one path would be *combine* and then *move*. The sequence of this path is further refined with timing information. Afterwards, the sequences are generated and provided to the HyBUTLA algorithm, which learns an overall timed automaton from these sequences. The timed automaton is then used as input for a code generator, which produces the final control code.

V. Summary

Plug and produce has the aim to significantly decrease engineering effort for production systems. In this paper, two classes of plug and produce processes have been identified: Non-coupled processes with no-function-in-structure and coupled processes with structure to function. The latter one have the characteristic that the connection of components itself is not enough, but the logical meaning of this connection has to be provided as well. Currently, no solution exists for this class of processes despite its omnipresence.

Our approach aims at synthesising a superordinated control with descriptive control code synthesis. It shifts the classical engineering view towards a more intuitive goal-driven view. The engineer can now think in terms of products and attributes instead of programming machine code, which is a much more intuitive view. The presented descriptive engineering approach uses product descriptions as input and generates control code from it. A product model can be specified by the engineer and can contain geometrical and physical aspects as well as bindings to other products. This way composite products can be defined, representing even the most complex kinds of products. The process model does not need to be specified by the engineer since it can be generated automatically. However, if desired, the engineer still has the opportunity to make changes in the process model or final control code as well. The two algorithms described in this paper, realise the transformations between these models. As a final step the control code is generated from an automaton representing all possible sequences of the production system.

Our descriptive engineering approach was verified on the Lemgo Model Factory. A production system consisting of six modules, which can handle multiple types of products. A product graph was defined, which represents product modifications. Given the handling and manufacturing processes of VDI 2860 and DIN 8580, the production process can be generated from the product graph. Results have shown that this case can be modelled with our approach and the transitions work as intended. Future steps will be the refinement of the algorithms and application to other use cases. Also, to compare our approach with already existing engineering approaches, we will investigate and apply metrics to them.

Acknowledgments

This research and development project is funded by the German Federal Ministry of Education and Research (BMBF) and by the German Federal Ministry of Economy and Energy (BMWI). The authors are responsible for the contents of this publication.

References

[1] U. E. Zimmermann, R. Bischoff, G. Grunwald, G. Plank, and D. Reintsema, “Communication, configuration, application - the three layer concept for plug-and-produce,” in *ICINCO-RA (2)*, 2008, pp. 255–262.

[2] J. De Kleer and J. S. Brown, “A qualitative physics based on confluences,” *Artificial intelligence*, vol. 24, no. 1, pp. 7–83, 1984.

[3] W. Wahlster, Ed., *SemProM - Foundations of Semantic Product Memories for the Internet of Things*, ser. Cognitive Technologies. Springer, 2013.

[4] L. Dürkop, J. Imtiaz, H. Trsek, L. Wisniewski, and J. Jasperneite, “Using opc-ua for the autoconfiguration of real-time ethernet systems,” in *11th International IEEE Conference on Industrial Informatics*, Bochum, Germany, Jul 2013.

[5] A. Zoitl, G. Kainz, and N. Keddis, “Production plan-driven flexible assembly automation architecture,” in *Industrial Applications of Holonic and Multi-Agent Systems*, ser. Lecture Notes in Computer Science, V. Mak, J. Lastra, and P. Skobelev, Eds. Springer Berlin Heidelberg, 2013, vol. 8062, pp. 49–58.

[6] C. Secchi, M. Bonfe, C. Fantuzzi, R. Borsari, and D. Borghi, “Object-oriented modeling of complex mechatronic components for the manufacturing industry,” *Mechatronics, IEEE/ASME Transactions on*, vol. 12, no. 6, pp. 696–702, Dec 2007.

[7] M. Loskyll, J. Schlick, S. Hodek, L. Ollinger, T. Gerber, and B. Pirvu, “Semantic service discovery and orchestration for manufacturing processes,” in *ETFA*, 2011, pp. 1–8.

[8] S. Ulewicz, D. Schütz, and B. Vogel-Heuser, “Design, implementation and evaluation of a hybrid approach for software agents in automation,” in *ETFA*, 2012, pp. 1–4.

[9] J. Pfrommer, M. Schleipen, and J. Beyerer, “Pprs: Production skills and their relation to product, process, and resource,” in *ETFA*, 2013, pp. 1–4.

[10] J. Otto, S. Henning, and O. Niggemann, “Why cyber-physical production systems need a descriptive engineering approach a case study in plug produce,” in *2nd International Conference on System-integrated Intelligence (SysInt)*, Bremen, Germany, Jul 2014, to be published.

[11] O. Niggemann, B. Stein, A. Vodenčarević, A. Maier, and H. Kleine Buning, “Learning behavior models for hybrid timed systems,” in *Twenty-Sixth Conference on Artificial Intelligence (AAAI-12)*, Jul 2012.

[12] T. Amnell, E. Fersman, P. Pettersson, H. Sun, and W. Yi, “Code synthesis for timed automata,” *Nord. J. Comput.*, vol. 9, no. 4, pp. 269–300, 2002.