On the applicability of model based software development to cyber physical production systems

Oliver Niggemann and Björn Kroll
Fraunhofer IOSB-INA,
Application Center for Industrial Automation, Lemgo, Germany
{oliver.niggemann, bjoern.kroll}@iosb-ina.fraunhofer.de

Abstract—The efficient handling of complex production systems and the implementation of more flexible and adaptable production lies at the heart of cyber-physical production systems and its german equivalent Industry 4.0. Such scenarios currently face one main difficulty: the creation, configuration and maintenance of the corresponding automation software is time-consuming and error-prone. Two main solutions exist for this problem: (i) model-based software development and (ii) intelligent automation, i.e. the usage of new knowledge-based solution approaches. This article compares these different solutions by applying them to three phases of the life-cycle: the planning phase, the operation phase and the plant modification phase.

I. MODEL-BASED DEVELOPMENT AND INTELLIGENT SYSTEMS

An initial glance would suggest that most experts do not agree on the contents of cyber-physical production (CPPS) and Industry 4.0. The focus in some cases is on information (Internet of Things), with the emphasis in other situations being placed on methods (self-diagnosis, self-configuration, etc.) or system capabilities (Intelligent Technical Systems), depending on the personal history of the experts involved. Moving away from the technical level, however, and return to the original question, and the picture becomes more homogeneous. The central concept is complexity reduction [1]. An increasing number of people evidently regard dealing with today’s production and automation technology to be excessively complex, far too prone to failure and overly inflexible. Examples include the automation specialist who is incapable of reconciling requirements relating to commissioning durations, energy savings and reliability within an appropriate period of time, the plant manager who no longer meets demands with regard to higher variability relative to low quantities (keyword: batch size 1) or the maintenance engineer who no longer repairs complex faults at a system level within a reasonable period of time. The objective in every case is to reduce the level of complexity perceived by people while maintaining the complexity of production and automation technology. In this sense, the focus is on the human factor in this work, and this is a further central feature of CPPS.

The classic response of computer science to these questions (and with regard to automation) is based on frontloading and, especially, in model-based software development [2], [3], [4], [5], [6], [7]. In the context of these approaches, modeling tools enable the expert to formalize (i.e. model) his know-how with regard to the software being developed at an abstraction level which he finds comfortable. A comfortable working point is achieved when the modeling level and the level of the mental model created by the person (i.e. his mental image of the software) closely converge. The term "frontloading" describes the trade-off hoped for: Modeling initially involves a greater investment of time and effort, but problems are later reduced through improved specification and exploitation of options for early virtual system verification (e.g. through simulation) and code generation. Figure 1 illustrates a typical model-based software development approach, with the expert specifying the software in a preferably abstract form with which he is most comfortable. Automation software and the appropriate configuration of automation are generated on the basis of these models. However, in contrast to model development in general, model-based software development always aims to describe the software. This also applies where software models are enhanced through supplementing with process components, hardware topology and basic software to form system models.

Fig. 1. A typical model-based software development procedure.

However, the more frequently systems and, consequently, automation are modified, the more often this engineering cycle needs to be realized. Moreover, the complete spectrum of knowledge relating to correct automation, such as system and control technology know-how, remains totally with the expert. This means that an increase in system complexity is reflected in an increase in modeling complexity. New model-based approaches can diminish this increase, but not completely prevent it.

Intelligent automation represents an alternative to model-based software development. In this approach, the expert no longer describes the automation procedure, but solely the automation goal, with the result that the engineering goal is no longer the definition of "how", but of "what". Classic production goals are product features, throughput or the maximum energy consumption. This procedure represents the transition from classic procedural automation to future...
descriptive automation.

As, in contrast to the automation procedure, the automation goal also usually remains unchanged in the case of system conversions (e.g. replacement of a system module), the expert is no longer continually involved in this approach. 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 procedure (i.e. the software). This results in a reduction of the complexity as perceived by the expert.

Figure 2 illustrates the difference between these two approaches in the case of a system conversion (i.e. with regard to requirements such as adaptivity and flexibility). Whereas the human factor is involved in every change in the case of model-based software development, descriptive automation reacts automatically in most cases to changes without the involvement of the expert.

These two approaches are compared below on the bases of three application cases. Section II compares model-based and descriptive approaches during planning of automation systems. Both approaches are compared for anomaly detection and diagnosis in Section III. Section IV illustrates the case of a system modification, while conclusions are drawn in Section V.

II. PLANNING AND COMMISSIONING PHASE

The contrasting approaches described in Section I are compared on the basis of several research projects for an application case involving planning and commissioning. The EFRE inITial project (higher productivity through model-based design and operation of complex automation systems) attempted to shorten commissioning through model-based software development methods and simulation. The German Federal Ministry of Education and Research (BMBF) EfA project (design methods for automation systems with model integration and automatic variation validation) adopts another method, with the user only modeling the requirements to be met by the automation solution in this approach, while the solution itself is generated automatically.

The inITial project developed a typical model-based approach to the planning of automated systems. This includes a structural model, involving a model of the structure of the production system (e.g. modules and their interconnection) and automation technology (e.g. controls, including software, sensors, actuators, networks). The structural model can be imported and stored as an AutomationML file. In addition, behavior models are stored for parts of the structural model. This is realized in the Modelica modeling language for system modules, while real IEC 61131 code is used for the control components.

These models are used for different tasks in the inITial project. On the one hand, they serve as methods to facilitate consultation and communication between the parties involved while, on the other, the models can be simulated on a PC to enable the early detection of automation errors. Ultimately, it is also possible to connect a real control to a simulated system in a hardware-in-the-loop (HIL) simulation, thus enabling examination of the real control prior to commissioning. Further details on the inITial project can be found in [8], [9].

The EfA project takes a completely different approach. The expert defines the process to be automated at the requirement level. The system automatically generates the automation solution from this, including the hardware topology and software function-blocks. From a theoretic viewpoint this approach tackles the challenge of supporting the formalization of expert knowledge in software implementations. Second, the precise formalisms allow new insights concerning handling the complexity of real world problems.

Both approaches are compared using the same plans, e.g. the Lemgo Smart Factory: The effort required for the descriptive approach is considerably less, as only the goal and not the solution needs to be described. A further consequence of this is that the automation expert can concentrate on process know-how in the descriptive approach, whereas the model-based approach demands that he also possess extensive knowledge of IT, the software and automation engineering.

Considerable differences exists for the verification of the automation software: In the case of the model-based approach, the whole approach aims at a virtual commissioning, with the controller (whether real or as a software-controller) being tested in a simulation of the system. However, this only functions if it is assumed that a good system model which has been aligned with reality exists. But it is precisely here that the problem arises. Generally speaking, neither the system integrator nor the system operator has the resources to develop models of this kind. An alternative is that machine and plant manufacturers also supply models of their systems and integrate these later through exchange formats such as AutomationML [10] to form an overall model. However, a variety of problems arise here: (i) Differing models are needed for different purposes, such as PLC programming or the creation of control systems (i.e. not only one, but several diverse models are required). (ii) Model parameterizing is only possible where knowledge of the overall system is available. However, a prerequisite for this is that the system integrator or system operator have the necessary knowledge and can also measure system behavior for the purpose of achieving an alignment between the model and reality. (iii) The business case for machine and plant manufacturers is currently ambiguous.
In the case of descriptive automation, no system models suitable for simulation exist (i.e. no model creation and parameterizing problems exist in this form). Instead, the know-how relating to the automatic generation of control software has to be modeled once from the description of the automation goal. This synthesis has to date not been researched in full and represents the greatest challenge during implementation of the descriptive approach. The EIA project, for example, compiles the control software from completed modules (i.e. software components in the form of IEC 61131 function blocks), with constraint-solving methods being employed for this purpose to assure the consistency of requirements and control systems.

However, in the event of this software synthesis being resolved, the need for tests conducted through simulation is dispensed with, because instead of the software, the software generation itself can be secured. This is analogous to compiler construction in which, similarly, the compiler and not the compilation is tested.

III. Operation Phase

Questions arise during the operation phase relating to the detection of anomalies, suboptimal energy consumptions or wear [11], [12], [13], [14]. The expert currently solves these questions in most cases through manual coding of fixed rules in automation code (see left-hand side of figure 2), and these rules draw on symptoms to detect anomalies or optimization needs [15]. However, all anomalies have to be thought out in advance for this purpose. In the context of CPPS, and with the requirement for support for frequent system conversions, this also means that these rules frequently require manual editing.

Another disadvantage of manual diagnostic rules arises from the complexity at the system level. While it is still possible to set up symptom → anomaly rules manually for individual units, the combinatorics for systems with numerous dependencies make this no longer feasible. An example can clarify this: A system with causal dependencies in the anomalies → symptoms form with \( m \) anomalies and \( n \) symptoms already requires a maximum of \( 2^n \) rules to differentiate between the symptoms.

Model-based approaches (see left-hand side of figure 2) therefore address this task in a different manner [11], [12], [13]. They compare the predictions of a behavior model with observations of the system and inform the user in the event of discrepancies being detected (e.g. poor energy consumption). For example, only \( n \cdot m \) rules are required for the behavior model in the above example for this purpose.

However, an immediate question arises concerning the origin of the behavior model. Manual modeling leads to all the disadvantages discussed in Section II. For this reason, the German Federal Ministry of Education and Research (BMBF) AVA project (abstraction of behavior models for distributed automation plants using observations) and the Anubis project (Analysis and monitoring of energy consumption in process and manufacturing engineering) funded by the ZVEI (Zentralverband Elektrotechnik- und Elektronikindustrie e.V) takes a different approach. The models are learned automatically on the basis of system monitoring in the form of hybrid timed systems.

Figure 3 illustrates an example of a behavior model for a module of the Lemgo Model Factory (see photo in figure 3) which was learnt automatically on the basis of system monitoring. The user only provides a descriptive specification of the type of anomaly (e.g. time, energy or sensor anomaly) in which he is interested and the sensitivity with which the system should react.

Fig. 3. A learned system with an unexpected event (e.g. caused by a software error).

The anomaly detection system now compares the learned model and system behavior. The behavior of the model and system is identical during states 0 to 4 in figure 3, but an unknown signal occurs during state 4. This is indicated to the user as an anomaly, with a programming error being the cause in this example.

IV. Modification Phase

The modification phase was already briefly addressed in Section I. A plant modification using the model-based software development approach (see left-hand side of figure 1) has up until now mainly been realized as follows: Following completion of the mechanical system modification, new devices such as sensors, actuators and controls are connected in the network, a task which in most cases involves a reconfiguration of the network itself. It is also necessary to modify all connected controllers at this stage to take the new network configuration into consideration and establish new communication relationships (e.g. to new system modules). Control algorithms are also frequently adapted and parameters such as task cycle times altered in the controls. In addition, adaptations are realized at higher levels, involving elements such as OPC servers, visualization software and MES systems. All these steps involve an extensive development and testing effort. Even where superior-quality models are used [5], [7] from which many of these settings can be generated, a manual engineering effort remains in each modification cycle.

Descriptive automation (see right-hand side in figure 1) adopts a different approach here, describing only the desired final product. This goal remains consistent (i.e. manual modification of automation is not necessary) during many plant modifications (e.g. replacement of a production module). In other cases, such as variation of the product, only the product description needs to be adapted. The effort involved is reduced enormously in each case.

Approaches of this nature are currently being researched in the it’s OWL IV leading-edge cluster project (intelligent networking cluster project) promoted by the BMBF. Fig. 4 illustrates the procedure in this project. The expert no longer models the automation software, modeling instead the final
product and the process. The process in this context consists of standardized process steps, with each process step featuring intermediate products and resources as inputs and outputs. Automation software is generated automatically from this description (e.g. in the form of an interconnection and parameterizing of pre-defined software components).

However, the examples illustrated also clearly demonstrate that model-based software development has matured and advanced further. Research needs to address a variety of unanswered research questions if descriptive automation is to be implemented: (i) Formalisms for products, processes and optimizing goals are still subjects of research and have not yet been standardized. (ii) Methods for automatic model learning do not exist. (iii) Automation software should be automatically generated on the basis of the descriptive description. Secured methods for software generation are missing here. (iv) The manner in which a migration from a classic engineering chain to a descriptive automation tool chain can succeed is unclear.

Fig. 4. Generation of automation software based on a product and process description.

The descriptive approach has clear advantages for this scenario: Instead of incorporating the software modifications in all models as in, for example, IEC 61131, the expert only formalizes the product and, as in the case of the it’s OWL IV project, also the process in the case of the descriptive approach. No effort for testing is needed when this solution approach is implemented. However, the advantages of the descriptive approach emanate from the secured and tested generation of software on the basis of product and process models. It is precisely here that questions currently arise during research, and planning algorithms and case-based approaches are presently being examined in this respect in the project “it’s OWL IV”.

V. CONCLUSION

Model-based software development and descriptive automation represent two fundamentally different approaches to the implementation of cyber-physical production systems and Industry 4.0. While the first approach develops methods to enable an expert to model the software in as comfortable and reliable a manner as possible, the descriptive approach is based on a description of the product goal, with the software being generated instead of modeled.

Models also play a central role in the descriptive approach, but here they describe the product and, to a degree, the process and specify optimization goals such as throughput and energy consumption. However, as they do not model the software in a static manner, the degrees of freedom gained as a result can be exploited to implement adaptivity (sections II and III), quality improvement (section III) and optimization (sections II and IV). It is precisely in this latitude between the descriptively defined production goal (“what”) and fixed process control in the automation (“how”) that knowledge-based methods such as self-configuration, self-diagnosis and self-optimizing work.

REFERENCES