A Model Based Approach for Generating Modular Manufacturing Control Systems Software

Mahmoud El Hamlaoui
ENSIAS, Mohammed V University in Rabat
Rabat, Morocco

Yassine Qamsane
Siemens Technology, Automation Engineering Software
Charlotte, NC 28273, USA

Youness Laghouaouta
National Institute of Posts and Telecommunications
Rabat, Morocco

Anant Mishra
Siemens Technology, Automation Engineering Software
Charlotte, NC 28273, USA

***Abstract*—Digitalization is transforming manufacturing systems to become more agile and smart thanks to the integration of sensors and connection technologies that help capture data at all phases of a product's life cycle. Digitalization promises to improve manufacturing flexibility, quality, productivity, and reliability. However, there is still a significant need of effective methods to develop models that could enhance the capabilities of manufacturing systems. Formal methods and tools are becoming essential to achieve this objective. Within this context, this paper introduces a formal software solution for the automatic generation of modular manufacturing control systems software. The proposed solution leverages software Model-Based Design (MBD) techniques to reduce development effort, time, and human error by automating several manual steps.**

Keywords-component; Industry 4.0; Cyber-Physical Manufacturing; Formal methods; MDE; DSL; Grafcet

I. INTRODUCTION

The world is in the midst of a new industrial revolution, referred to as Industry 4.0 or Smart Manufacturing (SM), driven by the rapid advancement in Information Communication Technologies (ICT) in terms of speed, power, autonomy, and mobility. The proliferation of ICT in the manufacturing domain has brought about Cyber-Physical Manufacturing Systems (CPMS), that combine physical production components (e.g., robots, machines, conveyors, and parts to be processed) with cyber components (e.g., logic controllers, networks, and data management infrastructures). CPMS gain their intelligence thanks to their connectivity to the Industrial Internet of Things (IIoT), which is an information network of physical objects (e.g., sensors and machines) that allows interaction and cooperation of these objects to reach common goals [1][2]. CPMS enable more flexibility, reactivity, proactivity, and adaptability to the dynamic global market shifts and fast changing customer requirements.

For manufacturers to remain competitive, CPMS are required to strategically adapt to factory changes associated with the responsiveness to the rapid changes in customer requirements and market conditions over time. In such agile environments, defining the interaction between the manufacturing control system and the physical manufacturing units using conventional development processes, which are characterized by an overhead of manual work, is effort and time-consuming and would have negative impacts on finances. Thus, to address uncertain and emerging situations, the manufacturing control system should be highly flexible, adaptable, and reconfigurable.

In previous work [3][4][5[6], we introduced a formal approach to automatically synthesize modular / distributed supervisory control for CPMS. The approach divides the overall control problem into local and global controls. Local Controllers (LCs) are developed for individual subsystems, then global interactions are added to the LCs to cooperatively execute the overall control actions. This approach ensures the flexibility required in CPMS to adapt to rapid changing conditions. For instance, in case of redesign or system extension, the practitioner would modify only the LCs for the impacted production modules and extend the global specifications with the new requirements, then automatically synthesize the new control logic. In contrast to a centralized controller approach where the entire control system needs to be rebuilt, the proposed approach requires to update only the affected manufacturing units. This approach enables the manufacturing units to be versatile and reusable in different environments.

The contribution of this paper is to derive a systematic software solution that is based on the previously proposed formal approach to support CPMS control designers in automatically generating control systems. The software solution leverages Model-Based Design (MBD) techniques to reduce development effort, time, and human error by automating several manual steps.

The rest of this paper is organized as follows. Section II recalls the theoretical proposition. Section III details the design and development of the proposed software solution. Finally, Section IV summarizes the contributions of this paper and presents some future research avenues for this work.

DOI reference number: 10.18293/SEKE2022-151

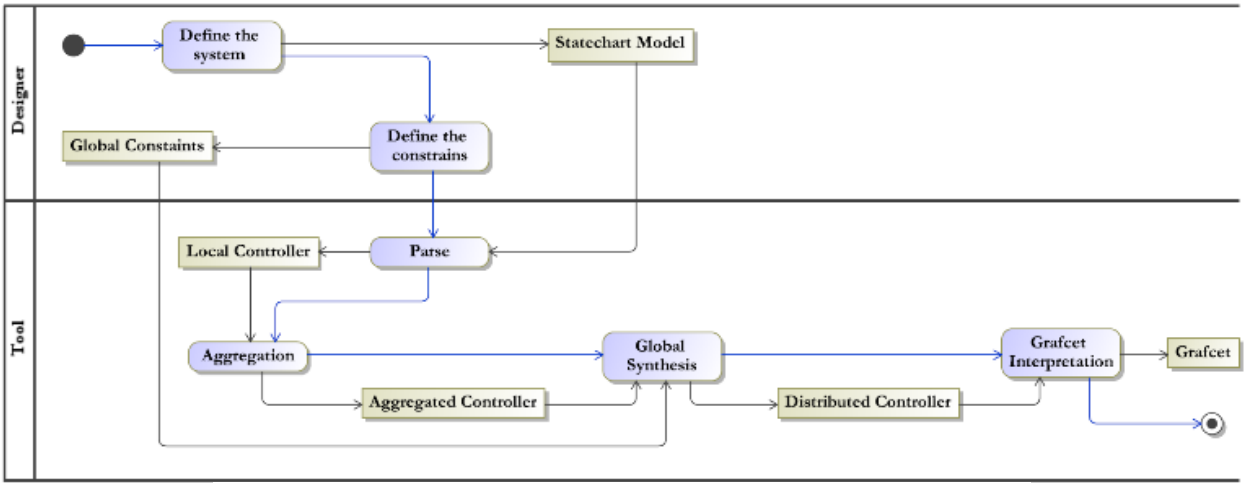

Figure 1. Activity diagram for the synthesis of distributed supervisory control

II. PROPOSED THEORETICAL APPROACH

The workflow shown in Fig. 1 displays the different steps of the supervisory controller synthesis process.

First, the discrete operation physically realizable by the system is modeled using discrete automata models in a local modular way according to its mechanical and functional characteristics (e.g., sensors and actuators). A Local Controller (LC) is derived for each local plant component from its automaton model. Second, to enable the overall system coordination, global specifications over the local components are defined and formalized as logical Boolean expressions. Then, the LC are aggregated to enable applying the logical Boolean expressions to the automata models. The aggregation procedure is carried out such that the states reached by controllable (actuator) events ($\sigma \in \Sigma_c$) are merged into macro-states that are interconnected with uncontrollable (sensor) events ($\sigma \in \Sigma_{uc}$) as explained in Section III. The output of this endogenous transformation is saved as an ALC (Aggregated Local Controller).

The Global synthesis activity consists of applying the formal global specifications to the corresponding ALC models, which allows the local components to coordinate among each other to reach the global control objective. At the output of this activity is the DC (Distributed Controller) models that implement both the local and coordination controls. A single DC is generated for each local manufacturing unit in the system. The last activity transforms the DCs into Grafcet specification in order to exploit the resulting model in the PLC (Programmable Logic Controller). The implemented transformation produces a JSON model visualized as Grafcet into GoJS [7]. The latter is a JavaScript library for creating and manipulating diagrams, charts, and graphs.

III. PROOF OF CONCEPT

According to the activity diagram of Fig. 1, after defining the system and control specification models, the proposed tool consists of four automated steps. First, the designer defines the system using UML Statechart models. Our statechart models are defined using PlantText [8], which is an Open-source tool, based on PlantUML [9], that uses a textual DSL to create graphical UML diagrams. Thus, our proposal is generic and not oriented for a particular software vendor (e.g., MagicDraw, Rational Software Modeler, Modelio, Visual Paradigm, etc.). In order to have a fully integrated MDE approach [10], we have proposed

```
7  Plstatemachine:
8      BEGINFILE (transitions+=Transition)* ENDFILE ;
9
10
11 Transition:
12     NL source= (State | InitialState) SourceToTarget  target=(State | InitialState) SymboleEvent event=ID ;
13
14 State:
15     name=ID ;
16
17 InitialState:
18     name='[*]' ;
19
20 terminal NL: ('\n')+;
21 terminal BEGINFILE:'@startuml';
22 terminal ENDFILE:NL'@enduml';
23
24 terminal SourceToTarget: '-->';
25 terminal SymboleEvent : ':';
```

Figure 2. Proposed grammar for PlantText

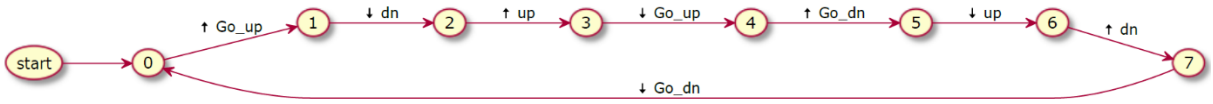

Figure 3. Local Controller (LC) model example

an alternative to the online PlantText editor. For this, we propose a textual editor based on Xtext. Figure. 2 shows the implemented grammar of the textual editor. Second, the aim of the "Parse" activity is to take the UML Statechart model established by the designer and automatically produce a LC model (Fig. 3) that is conform to a General Controller Metamodel.

The transformation is expressed as graph transformation unit implemented using Henshin [11]. Henshin is a transformation language and tool environment based on graph transformation concepts and operating on EMF models [11]. It provides features needed to express complex transformation such as negative application conditions (NACs), which specify the non-existence of model patterns in certain contexts and transformation units to control the rules application sequence. We have to notice that the rule declaration in the Henshin formalism does not explicit the description of the left- and right-hand sides. Instead, it is based on the following stereotypes to depict the rule application semantic: preserve, create, require, and forbid

Third, the "Aggregation" activity is an endogenous transformation that takes as input the LC model and produces the corresponding ALC model. The aim of this transformation is to apply corresponding control specifications to the latter. The ALC is produced through a graph transformation implemented with Henshin. Figure. 4 shows the rules being used to aggregate states respectively according to *Inh* and *Ord*. Transformations consist of removing the controllable evolutions from the LC model, and joining them into macro-states as follows: if the controllable event is associated with a rising edge, then the order is authorized and belongs to the set $Ord^{(DC)}$; otherwise, the order is inhibited and belongs to the set $Inh^{(DC)}$. Figure. 5 shows the resulted ALC model.

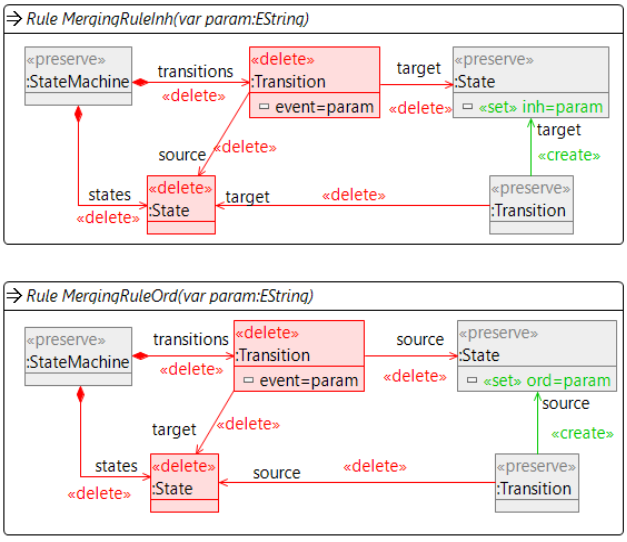

Figure 4. Aggregation transformation rule in Henshin

The basic data structure of a global constraint is a Boolean expression of the following form:

If (Condition) **Then** (Action)

Formally, the set of global constraints is defined by the pair $Spec = (C^{(spec)}, Act^{(spec)})$, where $C^{(spec)}$ is the set of *conditions*; and $\text{Act}^{(\text{spec})} = \{\text{Ord}^{(\text{spec})}, \text{Inh}^{(\text{spec})}\}$ is the set of activation/deactivation *actions*. Due to space limit, we choose to not present the textual editor grammar as it is based on the same idea of Fig. 2.

Example global constraints to be applied to the LC of Fig. 3 are stipulated as presented in Fig. 6. The global constraints are added to the ALC as follows. Check all the constraints for each ALC state. If an authorized (resp., inhibited) order of an ALC state is similar to that authorized (resp., inhibited) within a global constraint, then the constraint's condition should be associated with the actual state to condition the authorization (resp., the inhibition) of the corresponding order. The resulting controller is a DC model (conforms to the General Controller Metamodel). To avoid any confusion between the LC and ALC metamodels, the values regarding the conditions *Ord_If* and *Inh_If* are only represented in the DC model.

The rules implied in the transformation unit that takes as input the ALC and constraint models and produces the DC model as output (Fig. 7) have been implemented using Henshin. Dedicated rules have been expressed in order to retrieve *Inh* and *Ord* nodes from the constraint models and set the adequate values of *Inh_If* and *Ord_If* attributes.

The purpose of the last activity is to create a Grafcet model to be used in a running system (the PLC). To do so, we have used the metamodel proposed in [12]. It defines the steps and the different transitions between them.

To avoid defining a concrete representation for the Grafcet model, we decide to visualize the output into GoJS [5]. For that, the transformation is defined to a JSON model. For the sake of brevity, we don't explicit the transformation rules in this paper. Figure. 8 presents the visual model being generated for the LC of Fig. 3.

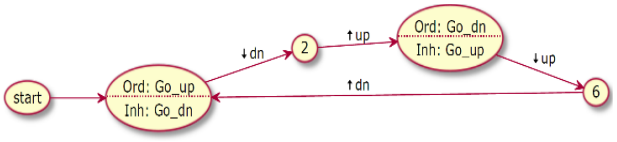

Figure 5. Aggregated Local Controller ALC model example

Id	If	Then	
	$C^{(spec)}$	$Ord^{(spec)}$	$Inh^{(spec)}$
1.	$free \wedge \neg Eject \wedge wpa2$	Go_up	
2.	$(\neg Ok \wedge \neg Eject) \vee ((Ok \wedge Eject) \rightarrow \uparrow ebp)$	Go_dn	

Figure 6. Aggregation transformation rule

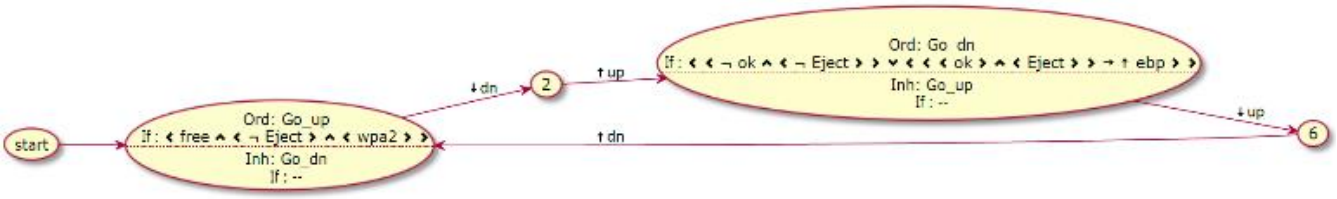

Figure 7. Resulting DC model example

IV. CONCLUSION & PERSPECTIVES

This paper presents a tool for automating the process of generating distributed supervisory control interpreted as Grafcet specification (IEC 60848 [13]) for CPMS. The tool framework decreases the state space explosion problem inherent to formal supervisory control theory methods by using a modular/distributed structure that avoids the synchronous composition of subsystem models. It also increases the flexibility required in manufacturing systems through distributed control models that enable a simple and adaptive control strategy, i.e., in the case of a redesign, only a small amount of data related to the corresponding subsystem controllers will be updated.

The founding framework of the tool uses model-checking technique for the verification of absence of deadlocks and making sure that the system safety and liveness requirements are met before the Grafcet implementation of the distributed control models. Model-checking technique allows tracing the sequences altering the system behavior, which enables easy update of the distributed control models. After the verification phase, simulation is used to validate the distributed control behavior.

We believe that additional investigations of the verification/validation process are of significant importance and would bring improvements to our software tool.

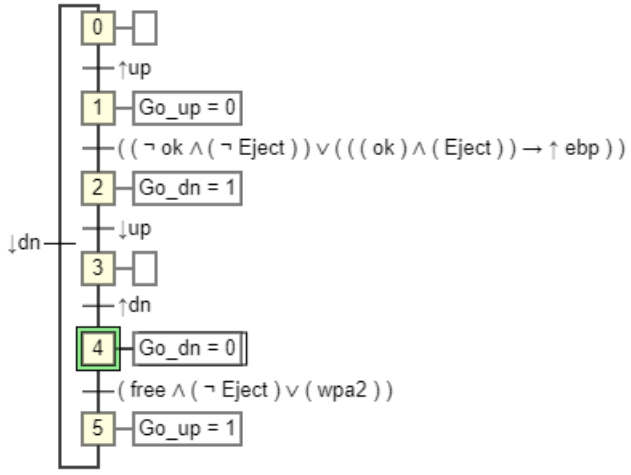

Figure 8. The generated Grafcet

Extensions of this work would introduce a linkage between our software tool and model-checking technique. Another avenue of research is the introduction of time-domain to the framework of this paper to evaluate quantitative control problems and measure system performance.

REFERENCES

[1] E. Sisinni, A. Saifullah, S. Han, U. Jennehag, and M. Gidlund, "Industrial internet of things: Challenges, opportunities, and directions," IEEE transactions on industrial informatics, vol. 14, no. 11, pp. 4724–4734, 2018.

[2] Y. Qamsane, E. C. Balta, J. Moyne, D. Tilbury, and K. Barton, "Dynamic rerouting of cyber-physical production systems in response to disruptions based on sdc framework," in 2019 American Control Conference (ACC). IEEE, 2019, pp. 3650–3657.

[3] Y. Qamsane, A. Tajer, and A. Philippot, "A synthesis approach to distributed supervisory control design for manufacturing systems with grafcet implementation," International Journal of Production Research, vol. 55, no. 15, pp. 4283–4303, 2017.

[4] Y. Qamsane, M. E. Hamlaoui, A. Tajer, and A. Philippot, "A tool support to distributed control synthesis and grafcet implementation for discrete event manufacturing systems," IFAC-PapersOnLine, vol. 50, no. 1, pp. 5806–5811, 2017.

[5] Y. Qamsane, M. E. Hamlaoui, A. Tajer, and A. Philippot. "A model-based transformation method to design PLC-based control of discrete automated manufacturing systems." 4th International Conference on Automation, Control Engineering and Computer Science (ACECS-2017). Vol. 19. 2017.

[6] Y. Qamsane, A. Tajer, and A. Philippot. "Towards an approach of synthesis, validation and implementation of distributed control for AMS by using events ordering relations." International Journal of Production Research 55.21 (2017): 6235-6253.

[7] N. Software, "Gojs, a javascript library for html diagrams," 2019.

[8] P. U. editor, "Uml editor - an online tool that generates images from text," https://www.planttext.com/, 2008.

[9] A. Roques, "Plantuml: Open-source tool that uses simple textual descriptions to draw uml diagrams," 2015.

[10] M. E. Hamlaoui, S. Bennani, M. Nassar, S. Ebersold, and B. Coulette, "Heterogeneous design models alignment: from matching to consistency management," in 33rd ACM/SIGAPP Symposium On Applied Computing (SAC 2018). ACM Digital Library, 2018, pp. 1695–1697.

[11] T. Arendt, E. Biermann, S. Jurack, C. Krause, and G. Taentzer, "Henshin: advanced concepts and tools for in-place emf model transformations," in International Conference on Model Driven Engineering Languages and Systems. Springer, 2010, pp. 121–135.

[12] P. Guyard, "Atl transformation example : Bridging grafcet, petri net, pnml and xml," 2005.

[13] IEC-60848, "Grafcet specification language for sequential function charts," 2013.