A Tool Support to Distributed Control Synthesis and Grafcet Implementation for Discrete Event Manufacturing Systems

Y. Qamsane * **M. El Hamlaoui** ** **A. Tajer** * **A. Philippot** ***

* *LGECoS Laboratory, ENSA-M, Cadi Ayyad University, Marrakech, Morocco (e-mail: qamsaneyassine@gmail.com, a.tajer@uca.ma)*
** *SIME Laboratory, IMS Team, Mohammed V University in Rabat, Rabat, Morocco (e-mail: mahmoud.elhamlaoui@gmail.com)*
*** *CReSTIC, University of Reims, Reims, France, (e-mail: alexandre.philippot@univreims.fr)*

Abstract: Current production systems are becoming more complex: manufactured products are increasingly technical, production components are more specific, and control specifications are rapidly changing. Thus, formal methods and tools are becoming essential to support the automated development of control systems. We propose to develop a tool for the synthesis and implementation of modular/distributed supervisory control for Automated Manufacturing Systems (AMS). To reduce the computational complexity, we divide the control problem into local and global controls. Local Controllers (LCs) are designed for the individual subsystems, then global dependencies are added to the LCs to cooperatively execute the control actions. The tool provides a distributed control, interpreted as Grafcet (standard IEC 60848), that can be lately converted into any suitable IEC 61131-3 standard programming language for PLC programming purposes. It is based on Model-to-Model (M2M) transformations implemented in an Eclipse Modeling Framework (EMF) environment.

Keywords: Supervisory Control, Automated Manufacturing Systems, Synthesis Method, Grafcet, Model-Driven Development.

1. INTRODUCTION AND RELATED WORKS

Programmable Logic Controllers (PLC) have occurred to support sequential control of Automated Manufacturing Systems (AMS) due to their low cost, reliability, and ease of programming Lu and Liao (2009). The programming of PLC is supported by the two international standards: IEC-60848 (2013) and IEC-61131-3 (2013). Among the different languages defined within these standards, Grafcet is widely used in the manufacturing industry as a specification and an implementation tool of PLC-based controllers.

In nowadays industrial practices, the control development rely on simple, intuitive, and direct interpretations of the control specifications into contol programs. In the case of small size systems this may produce valid solutions. However, when the size and complexity of a process increase, the time and cost of the control logic development rise as well. Thus, improved development methods and associated tools are increasingly required to support the control practitioner. Two well-known methods : the verification and validation (V&V) and the synthesis method may help the control practitioner in his task. The V&V method see, e.g. Biallas et al. (2012) consists of checking that a control program, intuitively constructed, fulfills given specifications by executing it against a simulated process or against the physical one. A major drawback of this method is the need for prior writing of the control programs. The synthesis method is based on constructing models of the system and their expected properties, then use algorithms to automatically generate a control model, which guarantees that the specified properties are never violated. Supervisory Control Theory (SCT), Ramadge and Wonham (1987) is a favorable formal method used to design control of AMS. The SCT aims at synthesizing a monolithic supervisor that satisfies a legal specification language of a target system. The objective of the supervisor is to disable the occurrence of a subset of events such that the given specifications are fulfilled. Nevertheless, because existing tools are unable to efficiently handle significant size problems, the computational complexity problem hinders the use of SCT methods in the industry. To deal with this problem, the following divide-and-conquer approaches, among others, have been introduced in the literature: the modular supervisory control see, e.g. Wonham and Ramadge (1988) aims at designing a set of small supervisors meeting various individual specifications, rather than constructing a single global supervisor that simultaneously meets the specifications altogether; the decentralized approach see, e.g. Shu and Lin (2014) divides the monolithic supervisions goal into several subgoals. The resulting individual sub-supervisors are simultaneously run to implement a solution for the initial problem; the hierarchical control architecture see, e.g. Hill et al. (2010) uses simplified process models to synthesize high-level supervisors capable of taking overarching decisions. To control the real process, these decisions are transmitted

to the low-level supervisors; and the distributed control approach see, e.g. Hu et al. (2015) assumes that a process is composed of several interconnected subsystems, which are required to exchange data with each other in order to reach a global goal. Local Controllers (LCs) are designed to control each subsystem individually, then request enough information sharing to cooperatively execute the control actions.

Modular/distributed approaches have sparked particular interest in the DES community because of their reduced complexity and implementation flexibility. In previous works Qamsane et al. (2014), Qamsane et al. (2016a) and Qamsane et al. (2016b), we have proposed a distributed control structure, where the plant is modeled by a collection of local modular automata, and the control specifications are modeled by a collection of logical Boolean expressions. First, LCs are constructed by applying the local control specifications to their corresponding subsystem models. Second, to establish global couplings, the global control specifications are applied to the obtained LCs. The resulting Distributed Controllers (DCs) allow achieving a non-blocking optimal closed-loop behavior, which is adaptive (in the case of a redesign, a small amount of data will be updated), and reduce the computational problem. The approach uses a model-checking technique (not discussed here due to limited space) to verify the non-blockingness and optimality of the DCs. Ultimately, these latter are converted into Grafcet formalism using a straightforward methodology. The methodology provides a basic Grafcet David and Alla (2010) which can be easily translated into one of the programming languages defined in the IEC-61131-3 (2013) for PLC programming purposes. Based on these previous results, this paper presents a software tool framework which assists control practitioners in automatically generating distributed control interpreted as a Grafcet specification for the purpose of PLC controllers implementation.

The relevance of Grafcet for PLC-based control solutions, has motivated the development of tools that support control development. A tool for the editing of compliant Grafcet is presented in Di-Meglio (2010). The tool allows to create Grafcet specifications using all basic elements defined in IEC-60848 (2013), including time dependencies, macro-steps, enclosing steps, and forcing orders. The tool prototype presented in Schumacher et al. (2013) supports control practitioners in automatically generating control programs compliant to IEC-61131-3 (2013) starting from a Grafcet specification. In Provost et al. (2011), a Grafcet specification model with no time-dependent elements can be translated into an equivalent Mealy machine for conformance test purposes. Let us mention that all these methods and tools assume to have a Grafcet specification from the start. Currently, a method and an appropriate support tool which automatically generates a Grafcet specification, starting from informal requirements are still lacking. The software tool framework we present in this paper makes a breakthrough on this issue. The tool is based on Model-Driven Development (MDD) from the software engineering domain. MDD process is a favorable approach for the development of complex software systems. It can be more easily used thanks to the advent of languages and tools dedicated to Model-Driven Engineering (MDE). MDE allows considering models as data and then used as first class entities in dedicated transformations. Java applications based on structured data models written in a domain-specific modeling language can be assessed by Eclipse Modeling Framework (EMF), which provides modeling and code generation capabilities. Using MDE terminologies, we present the development of our tool based on transformations written in with Java EMF libraries.

The rest of this paper is organized as follows. We present our tool in Section 2 and we illustrate it by using an AMS example in Section 3. Finally, Section 4 discusses the results of the paper and draws conclusions and ideas for further work.

2. THE TOOL

The tool we present in this section is built to evaluate the control synthesis and implementation method presented in detail in Qamsane et al. (2016b). The proposed architecture therin, assumes that a plant is modularly modeled according to its mechanical characteristics and the resulting local models are called plant elements (PEs). Local and global controls are treated separately. First, local safety and liveness constraints (defined as logical Boolean equations by a system expert) are applied to the corresponding local PEs according to a local synthesis algorithm, which provides an LC for each PE. Second, the application of global constraints, defined in the form of logical Boolean implications, to the LCs provides DCs allowing a cooperative interaction among the modular PEs. Third, the resulting DCs are reproduced to a model-checker (Uppaal) which verifies that the global control fulfills the safety and functional properties (deadlock-freeness and liveness). In the last step the obtained distributed control is interpreted into Grafcet for the purpose of PLC-based implementation.

We view a manufacturing system as a nesting of PEs (e.g. a single-acting cylinder controlled by a pneumatic monostable 3/2 valve, a one rotation direction electrical motor, etc.). Regardless of the environment in which they evolve, the PEs local operation does not change, thus, a set of immutable local constraints is retained for each PE. The application of these constraints to the PE models allows obtaining immutable LCs. Currently, we assume that the tool has as inputs: an LC and a set of its related global constraints. The basic data structure of an LC is a deterministic finite automaton, which is represented by the quintuple $G^{(LC)} = (Q^{(LC)}, \Sigma^{(LC)}, \delta^{(LC)}, Q_m^{(LC)}, q_0^{(LC)})$, where $Q_m^{(LC)}$ is a finite set of states, with $q_0^{(LC)} \in Q^{(LC)}$, the initial state and $Q_m^{(LC)} \in Q^{(LC)}$, the set of marked states, $\Sigma^{(LC)}$ is a finite set of events called an alphabet, and $\delta^{(LC)}$ is a transition function $\delta^{(LC)} : Q^{(LC)} \times \Sigma^{(LC)} \rightarrow Q^{(LC)}$. Events are divided into two disjoint sets, controllable and uncontrollable events. All events are assumed to be observable. 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 $Act^{(spec)} = \{Ord^{(spec)}, Inh^{(spec)}\}$ is the set of

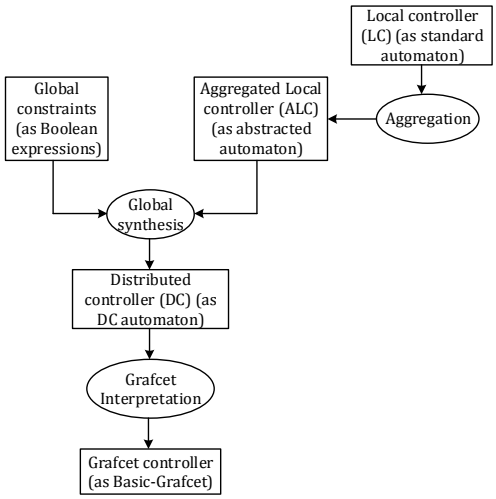

Fig. 1. The tool workflow

PEs' activation/deactivation *actions*. The tool workflow is presented in Fig. 1. First, the tool provides an abstraction of the LCs. In this step, the states reached by controllable events ($\sigma \in \Sigma_c$) are merged into macro-states that are connected with uncontrollable ones ($\sigma \in \Sigma_{uc}$). The method consists of hiding the controllable evolutions of the LC, and associating them into macro-states. Transitions between the merged states are ultimately checked as follows: if a controllable event is associated with a rising edge, it is interpreted as an authorized order and belongs to the set $Ord^{(DC)}$; Otherwise, it is interpreted as as inhibited order and belongs to the set $Inh^{(DC)}$. The resulting automaton is denoted as Aggregated Local Controller (ALC). ALC is a 4-tuple $G^{(ALC)} = (Q^{(ALC)}, \Sigma^{(ALC)}, \delta^{(ALC)}, q_0^{(ALC)})$, where $Q^{(ALC)}$ is the set of states; $\Sigma^{(ALC)} = \Sigma^{(LC)}$, the set of events; $\delta^{(ALC)}$ is the new transition function; and $q_0^{(ALC)}$ is the new initial state. The ALC is produced through a graph transformation implemented with Henshin Arendt et al. (2010). A graph transformation consists in applying specific transformation rules to a source graph. Each transformation specifies the modification of certain parts of this graph by another. The definition of this kind of replacement comprises two parts: a Left Hand Side (LHS) and a Right Hand Side (RHS). The LHS, LC in our case, is a graph which is used to determine the source graph's fragments concerned with the application of the rule, while the RHS, ALC graph in our case, specifies the replacement structure. Thus, a graph transformation rule, also known as a graph rewriting rule, is a couple of graphs (LHS, RHS). Essentially, the application of a rule on a given graph G comes down Andries et al. (1999):

(1) Detecting a subgraph of G isomorphic to the LHS part,
(2) Removing elements (nodes and edges) of the subgraph captured by the LHS part which are not present in the RHS part. The resulting graph is called context graph,
(3) Gluing the RHS graph into the context graph by following the connection points (elements that have been captured by the LHS but remain in the context graph).

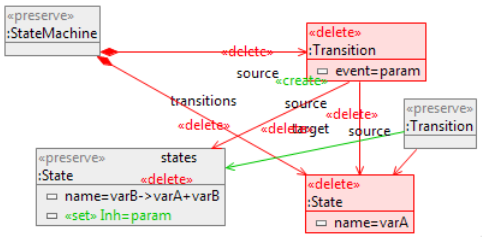

Fig. 2. General rule for merging states.

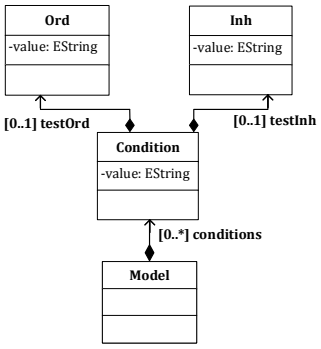

Fig. 3. Global constraints metamodel.

Fig. 2 shows an example of the implemented rule. Second, the tool considers the global constraints to the ALCs. The global constraints are written by using a textual editor developed with Xext Eysholdt and Behrens (2010). Xtext is a framework for the development of programming languages and DSLs based on EMF for handling models. It provides a definition language similar to BNF grammar (EBNF: Extended Backus-Naur Form Wirth (1996)) for the description of language syntax. The grammar that we have used for expressing the global constraints is based on a proposed metamodel (Fig. 3). The metamodel defines a set of conditions. Each one can have one or two possible values for the action. The produced Xtext editor offers a set of features: syntax highlighting, auto-completion, auto-indentation, navigation and search through the outline. Considering the global constraints to the ALCs consists of checking all the constraints for each ALC state. If an authorized (resp., inhibited) order of an ALC state matches to that authorized (resp., inhibited) within a global constraint, then the constraint's condition ($C^{(Spec)}$) should be associated with this state to condition the authorization (resp., the inhibition) of the corresponding order. The resulting controller is a DC. Formally, a DC automaton is syntactically defined by $G^{(DC)} = (Q^{(DC)}, \Sigma^{(DC)}, \delta^{(DC)}, Act^{(DC)}, C^{(DC)}, q_0^{(DC)})$, where $\Sigma^{(DC)}$ is a non-empty set of events such as $\Sigma^{(DC)} = \Sigma_c^{(DC)} \cup \Sigma_{uc}^{(DC)}$; $Q^{(DC)}$ is the set of states, to every state $q^{(DC)} \in Q^{(DC)}$ is associated a set of actions $Act_q^{(DC)}$ (which can be empty), and a set $C_q^{(DC)}$ (which can be empty) of logical conditions that monitor the authorized and inhibited orders; $q_0^{(DC)}$ is the initial state; $Act^{(DC)} = \{Ord^{(DC)}, Inh^{(DC)}\}$ is the set of actions associated with the states of $Q^{(DC)}$; $C^{(DC)} = \{C_{Ord}^{(DC)}, C_{Inh}^{(DC)}\}$is the set of logical conditions that monitor these actions; and $\delta^{(DC)} : Q^{(DC)} \times \Sigma_{uc}^{(DC)} \rightarrow Q^{(DC)}$ is the transition function. A transition of $G^{(DC)}$ is defined with the triple $(q, \sigma, q') \in \delta^{(DC)}$,

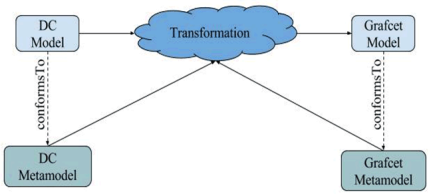

Fig. 4. The transformation concept.

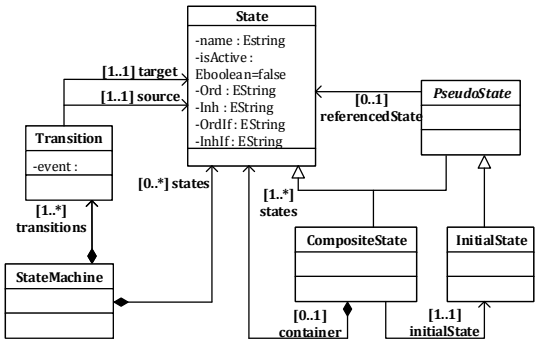

Fig. 5. The DC metamodel.

where q is the origin state, σ is an uncontrollable event and q' is the destination state. Technically, a DC is obtained from a Model-to-Model (M2M) transformation that takes as input the ALC and the global constraints models as well as their metamodels and produces a DC model. The transformation is implemented in Java/EMF environment. There are two reasons behind choosing Java/EMF. First, because EMF is being used by several tools, as it is the basis of all transformation languages, e.g. M2M, Model to Text (M2T), Model Development Tools (MDT). Second, using Java/EMF will enable us to use object-oriented practices to structure our code.

In the final step, the tool allows to interpret the DCs into partial Grafcet controllers. A straightforward method for the interpretation of any DC configuration into Grafcet was defined in Qamsane et al. (2016b)(section 3.5). Therein, altogether 11 interpretation rules have been defined to ensure a correct transformation of the DCs into Grafcet. The basic data structure of a Basic-Grafcet according to David and Alla (2010) includes steps, transitions, junctions, Boolean transition conditions, and actions. Thus a Basic-Grafcet $G^{(basic)}$ is described as a 9-tuple $G^{(basic)} = (S, T, W, sit_0, I, O, r, a, \Omega)$, where S is the set of steps; T is the set of transitions; W is the flow relation which contains all junctions; sit_0 is the initial situation; I is a finite set of inputs; O is a finite set of outputs; a is the set of a basic-Grafcet actions; and Ω is the output function which contains all executable actions with regard to a situation $sit(m)$. According to Fig. 4, our tool produces a Grafcet model by taking the following artefacts as inputs: a DC metamodel, a Grafcet metamodel and a DC model. The DC metamodel is described in Fig. 5. It defines a concept called StateMachine, which includes the set of states and the set of state-transitions. The metamodel represented in Fig. 6 illustrates the Grafcet metamodel concepts. A Grafcet contains a set of elements, where an

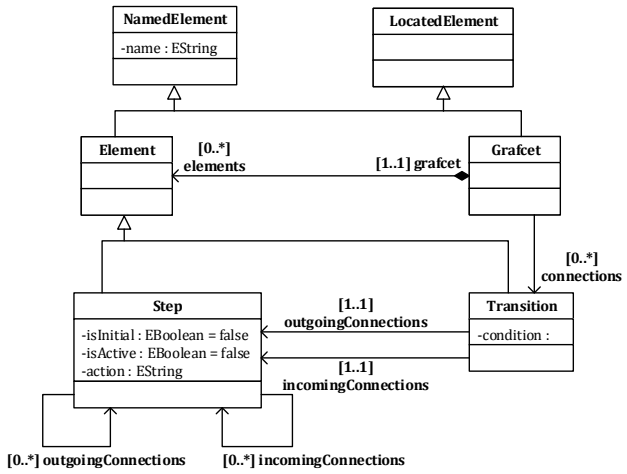

Fig. 6. The Grafcet metamodel.

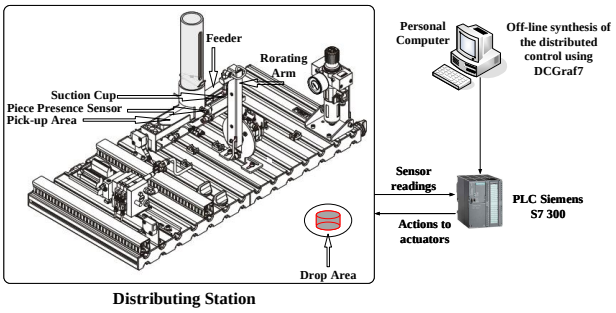

Fig. 7. The experimental AMS

element is an abstract concept that can determine one specialized step and transition.

3. APPLICATION

3.1 The experimental AMS

In this section an example of an industrial AMS illustrates our software tool framework. The industrial AMS (Fig. 7) is a modular production station controlled by a PLC Siemens S7-300, which aims at distributing workpieces from a magazine barrel to a downstream station. It is divided into three PEs: A feeder (single-acting cylinder), a swivel drive (double-acting cylinder), and a suction cup. The feeder feeds the workpieces from a buffer to the pickup area; the swivel drive transports the workpieces towards a drop area; and the suction cup catches the workpieces while the swivel drive is moving. The station uses a set of sensors to detect the arrival of workpieces, their colour and material, and working area freeness. The LCs models of the distributing station are given in Fig. 8. The reader can find detailed explanations about their construction in Qamsane et al. (2016b). In order to reach a global goal, ten global control specifications are stipulated for the experimental AMS as follows: (1) The feeder doesn't extend if the swivel drive is in the magazine position; (2) The feeder doesn't extend without a workpiece; (3) The feeder doesn't extend while the swivel drive is moving to the magazine position; (4) The swivel drive only goes to the magazine position when necessary; (5) The swivel

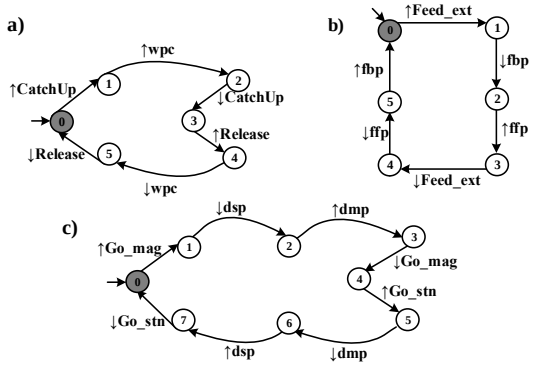

Fig. 8. LCs of the experimental AMS

Table 1. The global constraints

If	Then	
	Ord	***Inh***
¬dmp∧wpa∧¬Go_mag	Feed_ext	
¬wpc∧ffp∧¬CatchUp	Go_mag	
wpc∧testing_vacant	Go_stn	
¬Go_mag∧(ffp→dmp)	CatchUp	
dsp	Release	CatchUp
dmp∧wpc		Feed_ext

drive moves towards the testing station position only if the latter is not occupied; (6) The swivel drive moves towards the testing station with a securely caught workpiece; (7) Mutual exclusion between the suction cup activation, and the swivel drive moves towards the magazine position; (8) The suction cup activates only when a workpiece is in the pick-up area; (9) The feeder retracts when a workpiece is picked up; (10) The suction cup releases the workpiece when the swivel drive reaches the testing station position.

These constraints are formally presented in Table. 1. For instance, the first three constraints relate to the authorization of the action: "*Feed_ext*" (Feeder extend). These allow the feeder to extend when the drive is not in the magazine position ($\neg dmp$), a workpiece is available in the buffer (wpa), and the swivel drive is not moving to the magazine position ($\neg Go\_mag$), respectively. This can be written in Boolean algebra as follows: **if** $(\neg dmp \wedge \neg Go\_mag \wedge wpa)$ **then** Ord($Feed\_ext$).

3.2 Tool support

This section shows how the different models of the studied system have been produced with the aid of the proposed software tool. The latter receives the set of LCs automata shown in Fig. 8 together with their related global Boolean constraints shown in Table. 1. It uses the procedures explained in section 2 to generate the DCs and the partial Grafcet models as shown in Fig. 9.

4. DISCUSSION AND CONCLUSION

Supervisory control and performance analysis are two fundamental needs to design AMS. The former aims to constrain the system behavior to its legal specifications in a logical domain, whereas the latter needs time-domain work by aiming to evaluate system performance Hu and Liu (2015). Within supervisory control context, we presented a tool to distributed controller synthesis and Grafcet implementation for AMS. The tool is based on Model-to-Model transformations implemented in Java/EMF environment. Our tool framework makes two contributions. First, it decreases the state space explosion problem by using a modular/distributed structure which avoids the synchronous composition among the subsystems' models. Second, it increases the flexibility required in AMS through the DCs, which are simple and adaptive, i.e., in the case of a redesign, only a small amount of data related to the corresponding subsystems will be updated. Let us mention that this approach uses a model-checker (Uppaal) to verify the absence of deadlocks and ensure the system liveness before the Grafcet implementation level. The model-checker allows to trace the sequences disrupting the system's behavior, which enables swift update of the models. The Grafcet interpretation only comes once the DCs satisfy the requested properties. We believe additional investigations of the verification/validation are of interest and would bring improvements to our software. Extensions of this work would introduce a linkage between our software tool and Uppaal model-checker. Another avenue of research is the introduction of time-domain to the framework of this paper in order to treat some quantitative control problems and to evaluate the system performance. Future applications work would also consider the issues with moving from the synchronous event-based DES-world, to the asynchronous, signal-based PLC-world, outlined by Fabian and Hellgren (1998) and Hellgren et al. (2001).

REFERENCES

Andries, M., Engels, G., Habel, A., Hoffmann, B., Kreowski, H.J., Kuske, S., Plump, D., Schürr, A., and Taentzer, G. (1999). Graph transformation for specification and programming. *Science of Computer programming*, 34(1), 1–54.

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

Biallas, S., Brauer, J., and Kowalewski, S. (2012). Arcade. plc: A verification platform for programmable logic controllers. In *Automated Software Engineering (ASE), 2012 Proceedings of the 27th IEEE/ACM International Conference on*, 338–341. IEEE.

David, R. and Alla, H. (2010). *Discrete, continuous, and hybrid Petri nets*. Springer Science & Business Media.

Di-Meglio, S. (2010). Sfcedit: Un éditeur de grafcet. `http://stephane.dimeglio.free.fr/sfcedit/en`.

Eysholdt, M. and Behrens, H. (2010). Xtext: implement your language faster than the quick and dirty way. In *Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion*, 307–309. ACM.

Fabian, M. and Hellgren, A. (1998). Plc-based implementation of supervisory control for discrete event systems. In *Decision and Control, 1998. Proceedings of the 37th IEEE Conference on*, volume 3, 3305–3310. IEEE.

Hellgren, A., Fabian, M., and Lennartson, B. (2001). Modular implementation of discrete event systems as sequential function charts applied to an assembly cell. In *Control Applications, 2001.(CCA'01). Proceedings of*

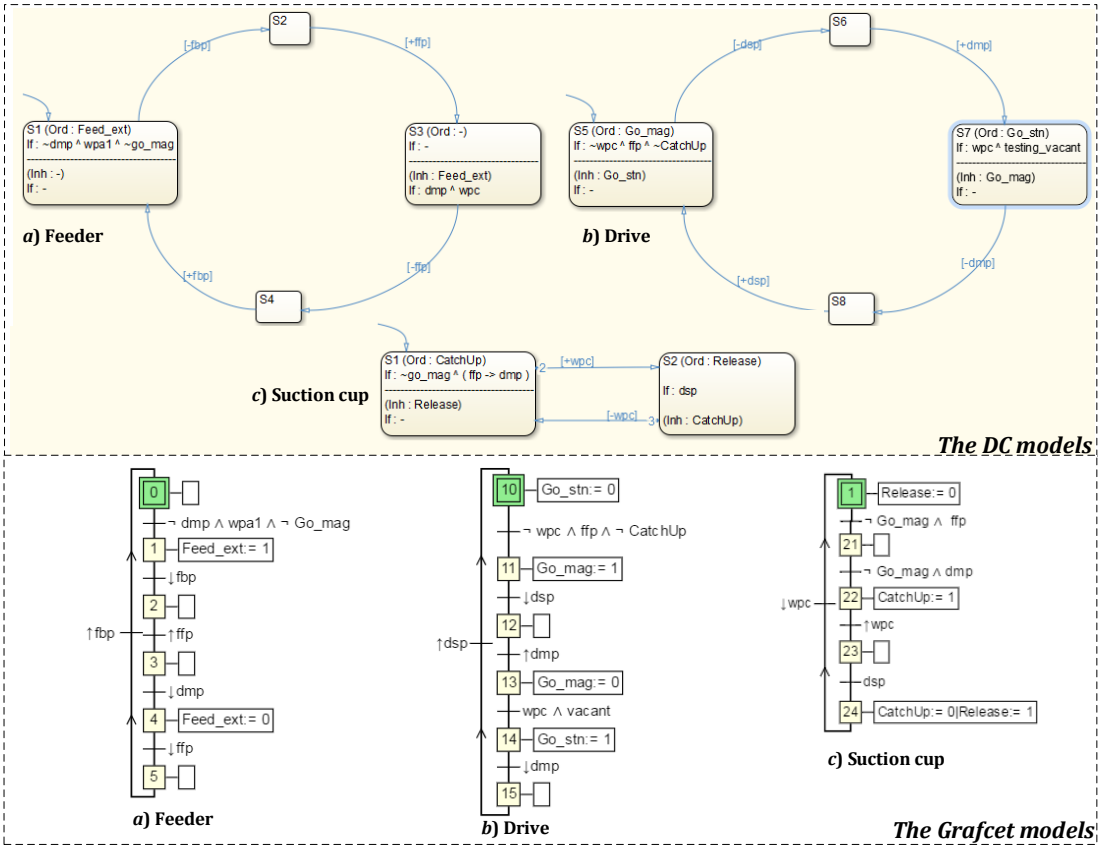

Fig. 9. The generated DC and Grafcet models.

the 2001 IEEE International Conference on, 453–458. IEEE.

Hill, R.C., Cury, J.E.R., de Queiroz, M.H., Tilbury, D.M., and Lafortune, S. (2010). Multi-level hierarchical interface-based supervisory control. *Automatica*, 46(7), 1152–1164.

Hu, H. and Liu, Y. (2015). Supervisor synthesis and performance improvement for automated manufacturing systems by using petri nets. *IEEE Transactions on Industrial Informatics*, 11(2), 450–458.

Hu, H., Liu, Y., and Zhou, M. (2015). Maximally permissive distributed control of large scale automated manufacturing systems modeled with petri nets. *IEEE Transactions on Control Systems Technology*, 23(5), 2026–2034.

IEC-60848 (2013). Grafcet specification language for sequential function charts.

IEC-61131-3 (2013). Programmable controllers-part 3: Programming languages, (3rd ed.).

Lu, M.S. and Liao, S.F. (2009). An integrated idef0-3/ctpn/sfc approach for design and analysis of discrete event control systems. *International Journal of Production Research*, 47(22), 6433–6453.

Provost, J., Roussel, J.M., and Faure, J.M. (2011). Translating grafcet specifications into mealy machines for conformance test purposes. *Control Engineering Practice*, 19(9), 947–957.

Qamsane, Y., Tajer, A., and Philippot, A. (2014). Synthesis and implementation of distributed control for a flexible manufacturing system. In *Complex Systems (WCCS), 2014 Second World Conference on*, 323–329. IEEE.

Qamsane, Y., Tajer, A., and Philippot, A. (2016a). Distributed supervisory control synthesis for discrete manufacturing systems. *IFAC-PapersOnLine*, 49(12), 396–401.

Qamsane, Y., Tajer, A., and Philippot, A. (2016b). A synthesis approach to distributed supervisory control design for manufacturing systems with grafcet implementation. *International journal of Production Research*, 1–21.

Ramadge, P.J. and Wonham, W.M. (1987). Supervisory control of a class of discrete event processes. *SIAM journal on control and optimization*, 25(1), 206–230.

Schumacher, F., Schröck, S., and Fay, A. (2013). Tool support for an automatic transformation of grafcet specifications into iec 61131-3 control code. In *2013 IEEE 18th Conference on Emerging Technologies & Factory Automation (ETFA)*, 1–4. IEEE.

Shu, S. and Lin, F. (2014). Decentralized control of networked discrete event systems with communication delays. *Automatica*, 50(8), 2108–2112.

Wirth, N. (1996). Extended backus-naur form (ebnf). *ISO/IEC*, 14977, 2996.

Wonham, W.M. and Ramadge, P.J. (1988). Modular supervisory control of discrete-event systems. *Mathematics of control, signals and systems*, 1(1), 13–30.