4th International Conference on Automation, Control Engineering and Computer Science (ACECS - 2017)
Proceedings of Engineering and Technology – PET
Vol.19, pp.4-11

A Model-Based Transformation Method to Design PLC-Based Control of Discrete Automated Manufacturing Systems

Yassine Qamsane*, Mahmoud El Hamlaoui†, Abdelouahed Tajer* and Alexandre Philippot‡
*LGECoS, ENSA, Cadi Ayyad University, Marrakech, Morocco
Email: qamsaneyassine@gmail.com, a.tajer@uca.ma
†SIME Laboratory, IMS Team, Mohammed V University, Rabat, Morocco
Email: mahmoud.elhamlaoui@gmail.com
‡CReSTIC, University of Reims Champagne Ardenne, Reims, France
Email: alexandre.philippot@univ-reims.fr

***Abstract*—In the domain of Automated Manufacturing Systems (AMS), Supervisory Control Theory (SCT) is a general model-based framework, which contributes to the automated development of control systems. Although, SCT delivers important theoretical insights to deal with controllers design, its use in the industry is still weak, because of the discrepancy among its abstract controllers and their physical implementation. Whithin this context, this paper presents a practical method to automatically produce distributed supervisory control of AMS in the form of Grafcet, which is a graphical modelling tool widely used to specify controllers dynamic behaviors. This method is built to evaluate an authors' previous distributed approach to control synthesis and implementation. It is based on Model-to-Model (M2M) transformations implemented in a java/EMF environment. An experimental AMS illustrates its purposes.**

***Index Terms*—Automated manufacturing systems, Model-Driven engineering, control systems, Automation, PLC, Grafcet.**

I. Introduction

Rigorous methods are becoming increasingly required for the analysis, design, validation, implementation, control, and optimization of large scale Automated Manufacturing Systems (AMS). Supervisory Control Theory (SCT) initiated by Ramadge and Wonham (RW) [1], which is based on Discrete Event System (DES) methods, is a promising technique for the study of AMS. Typically, the SCT aims at synthesizing a single monolithic supervisor that satisfies a legal specification language of a target system. However, for large scale systems, the computational complexity, which is due to the exponential increase of eligible events constitutes a major drawback that hinders the use of SCT methods in the industry. Numerous divide-and-conquer approaches have been proposed in the literature to deal with this issue. The modular approach see, e.g. [2] aims at synthesizing small supervisors that satisfy various individual specifications, rather than synthesizing a single monolithic supervisor that satisfy all the specifications together. The hierarchical approach see, e.g. [3] uses simplified process models to develop high-level supervisors capable to take overarching decisions. These are provided to the low level supervisors that control the real process. The decentralized approach see, e.g. [4] consists of dividing the global control goal into several sub-goals, then, simultaneously execute the resulting individual sub-supervisors to implement a solution for the initial problem. The distributed approach see, e.g. [5] divides a system into several interconnected subsystems. To reach a global goal, the subsystems are required to share information with each other. First, loosely Local Controllers (LCs) are defined to control each subsystem individually. Second, the LCs extchange information to cooperatively execute the control actions.

Our recent research has focused on the distributed approach because of its reduced complexity and implementation flexibility. Computational complexity is reduced through local control design, and implementation flexibility refers to that a system modification might be made only over the corresponding subsystems, which results in updating only their corresponding controllers. As a result, we proposed a distributed control architecture [6], [7], [8], where the plant is modeled by a set of local modular automata, and the control specifications are modeled by logical Boolean expressions. First, LCs are designed by applying local control specifications to their corresponding subsystems automata. Second, Distributed Controllers (DCs) are obtained by applying global control specifications also stated as logical Boolean expressions to the corresponding LCs. The resulting DCs allow a nonblocking optimal closed-loop behavior. Moreover, one important step of our approach is the interpretation of the distributed control into Grafcet for implementation purposes. Grafcet is an international standard [9] used for logic controllers specification for AMS.

In this paper, we focus on the implementation issue. Actually, Programmable Logic Controllers (PLC) are implementation platforms widely used to perform control tasks in practically all nowday's modern industrial systems. Grafcet is a well-known discrete event modeling tool used for logic controllers' specification for PLC programing purpose. The main contribution here is to present a software solution, which allows an automatic translation of a DC model into a Grafcet one which respect the definitions of [9]. The method we propose constitutes an intermediate phase between SCT-based control, which is abstract, and the obtained Grafcet controller, which is supposedly the executable concrete and precise PLC program. I.e. it overcomes the problem of discrepancy between the SCT abstract controller and its physical implementation.

Actually, with the advent of languages and tools dedicated to Model-Driven Engineering (MDE) see, e.g [10], [11], Model Driven Development (MDD) process can be more easily used. MDE allows to consider models as data then use them as first class entities in dedicated transformations. Our software solution is based on methods from the software engineering domain. It uses MDE terminologies for the development of transformations written in with the Java EMF libraries .

The rest of this paper is structured as follows. Section 2 reminds the basic concepts of the distributed control synthesis approach. Section 3 details the contribution of this paper theoretically and technically. in Section 4 the proposed method is illustrated using a case study. Finally Section 5 concludes and draws perspectives.

II. Fundamental concepts of the distributed control synthesis approach

This Section recalls the basic concepts of our formal approach to distributed supervisory control synthesis. The obtained control by using this shall be translated into Grafcet using the method presented in this paper.

The control synthesis and implementation architecture [8] is illustrated in Fig. 1. First, the operation potentially realizable by the system is modularly modeled in the form of automata based on its mechanical characteristics (sensors/actuators). Each subsystem automaton model is called a Plant Element (PE) [12]. The application of local constraints, modeled as logical Boolean equations, to their corresponding PEs provides LCs. Second, the global constraints are applied to the corresponding LCs, which allows the PEs to synchronize and cooperate among each other through the obtained DCs. Third, a model-checking technique is used to verify and validate that the designed global control fulfills the required functional properties like the deadlock-freeness and liveness properties. Fourth, a straightforward method to the interpretation of the DCs into Grafcet for PLC-based implementation purposes is proposed.

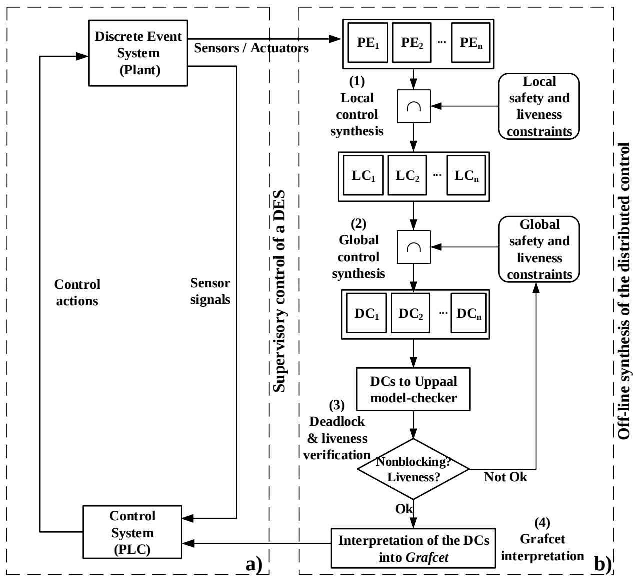
This paper is focused on the fourth step of the approach. We assume we have a DC model, the aim is to provide a software that automatically transform it into Grafcet. The objective is to assess AMS PLC-based control solutions.

III. Software design and implementation

In this Section, we first recall the rules of transforming a DC model into a Grafcet one. Then, the software design and implementation steps are presented.

A. From DC model to Grafcet

The syntax and semantics of DCs automata are described in more detail in [8]. A straightforward method for the



Discrete Event System (Plant)
Sensors / Actuators
PE1 PE2 ··· PEn
(1) Local control synthesis
Local safety and liveness constraints
LC1 LC2 ··· LCn
(2) Global control synthesis
Global safety and liveness constraints
DC1 DC2 ··· DCn
Control actions
Sensor signals
Supervisory control of a DES
Off-line synthesis of the distributed control
DCs to Uppaal model-checker
(3) Deadlock & liveness verification
Nonblocking? Liveness?
Not Ok
Ok
Control System (PLC)
Interpretation of the DCs into Grafcet
(4) Grafcet interpretation
a)
b)

Fig. 1. The proposed distributed supervisory control architecture.

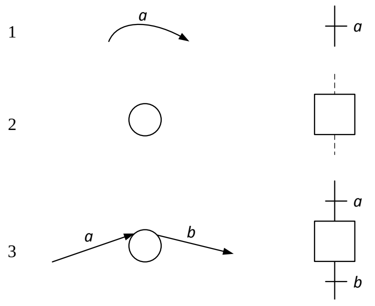


Fig. 2. Rules for the DC simple evolutions transformation.

interpretation of DCs into Grafcet follows. The method is based on the following rules:

- An ordinary state of a DC is represented by a step of Grafcet;
- An uncontrollable event associated with a DC transition is represented by a transition of Grafcet. The receptivity associated with the Grafcet transition is defined as the occurrence of the uncontrollable event;
- Macro-states have different configurations according to the authorized and inhibited orders, and their monitoring conditions, see Fig. 3;
- When an order is authorized and another is inhibited in a same DC macro-state, the priority goes to the inhibition i.e. one must disable the command already activated before enabling the authorized one to ensure safety.

Altogether 11 interpretation rules have been defined to ensure a correct transformation of the DCs into Grafcet, see Figs. 2 and 3.

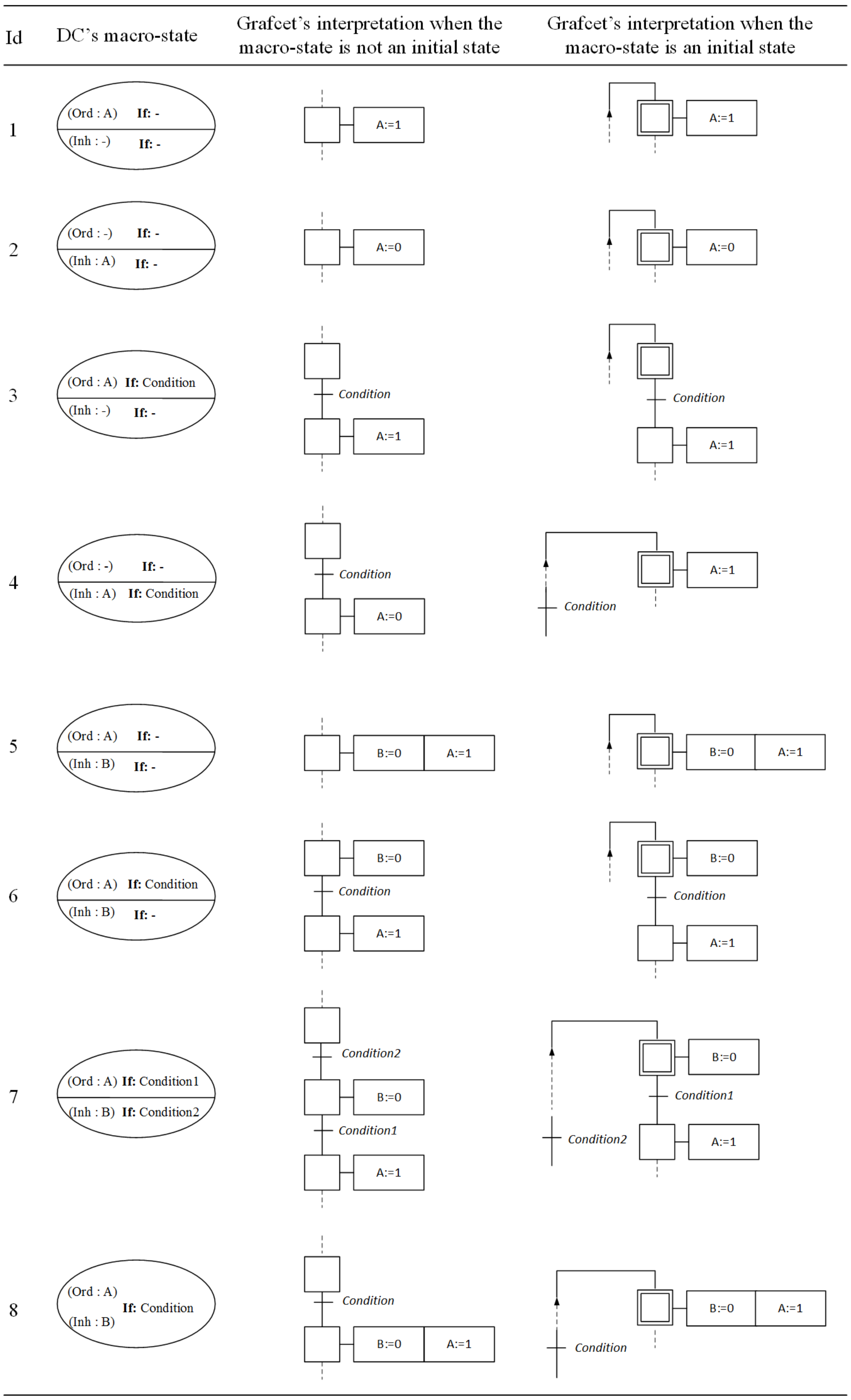


Fig. 3. Rules for the DC macro-states transformation.

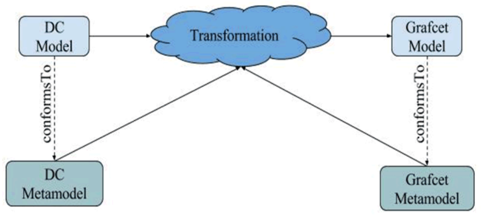


Fig. 4. Transformation concept.

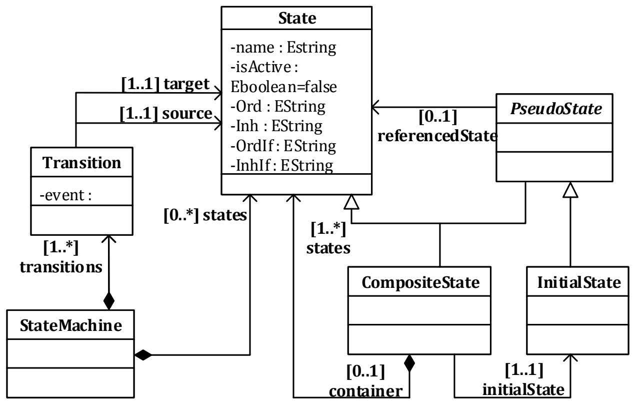


Fig. 5. The DC metamodel.

B. Software design and implementation steps

To validate the approach, we are developing a software tool using Eclipse [13], the open source platform of development, considered as the main incubator of development projects by the MDE community.

The software tool is an EMF-based transformation module. EMF is an environment of modeling and code generation facility for building tools and other applications based on a structured data model. 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. Model to Model (M2M), Model to Text (M2T), Model Development Tools (MDT). Second, using Java with EMF will enable us to use object-oriented practices to structure our code. Nevertheless, EMF lacks of a concrete syntax support. EMF has only a tree editor that relies on the countenance structure of metamodel elements. We shall soon fill this gap by using other frameworks like, Xtext and GMF.

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.

Fig. 5 describes the DC metamodel. 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. An element is an abstract concept that can determine one specialized step and transition.

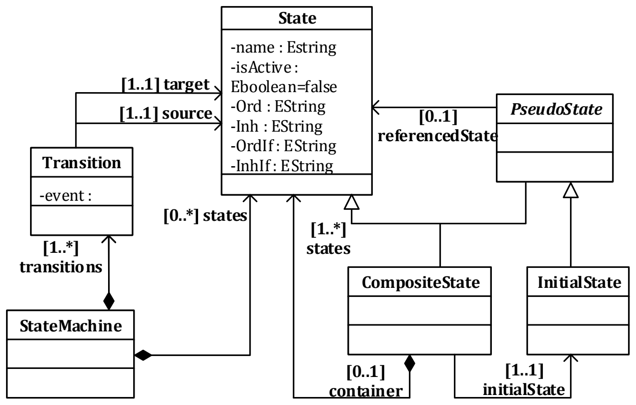


Fig. 6. The Grafcet metamodel.

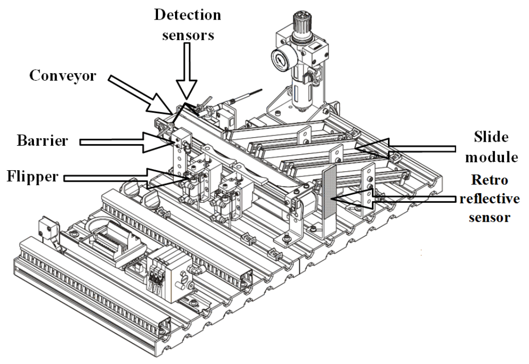


Fig. 7. The experimental AMS.

IV. Case study design

As a case study, the methodology has been applied to an industrial AMS (a workpieces sorting station), see Fig. 7. The system identifies the color of entering workpieces and deposes them onto one of three sliders according to their colors (black, red or silver). It consists of a sensing module; a pneumatic barrier; a conveyor belt; and two pneumatic cylinders in the form of flippers. The conveyor belt actuated by a direct current motor conveys the workpieces towards the sliders. The pneumatic barrier blocks the workpieces at the entrance of the station to identify their color and material. The pneumatic flippers are used to select the sliders, where the workpieces should be sorted. A set of sensors is used to detect the arrival of workpieces, their color, and their material.

Even if at this stage of our tool development, we assume we have DC models of a given system, and the aim is to automatically generate their Grafcet interpretation, we present nonetheless how these models are obtained by our approach.

A. Plant modelling

The sorting station comprises four PEs: two monostable double-acting cylinders (Flippers), a single-acting cylinder (Pneumatic barrier), and a conveyor. We do not explain the construction of the PE models shown in Fig. 8. The reader can find detailed explanations in our previous work [12].

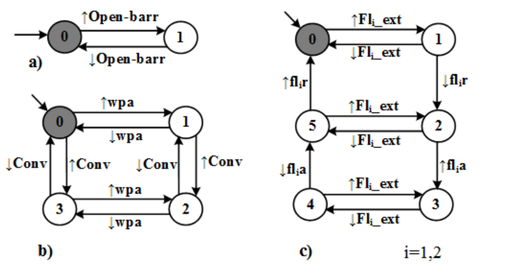


Fig. 8. Plant models. (a) Barrier, (b) Conveyor, (c) Flippers.

TABLE I
LOCAL SAFETY AND LIVENESS CONSTRAINTS

PE	Type	Constraints
Conveyor	Safety	↑Conv **And Not** ↑wpa=0
	Liveliness	↓wpa **And** ↓Conv =1
		↑wpa **And** ↑Conv =1
		↑Conv **And** ↓wpa =1
Flipper$_i$ (i=1,2)	Safety	↓Fl$_i$ **And Not** ↑fl$_i$a =0
		↑Fl$_i$ **And** ↑fl$_i$r =0
	Liveliness	↑Fl$_i$ **And** ↓fl$_i$a =1
		↓Fl$_i$ **And**↓fl$_i$a =1

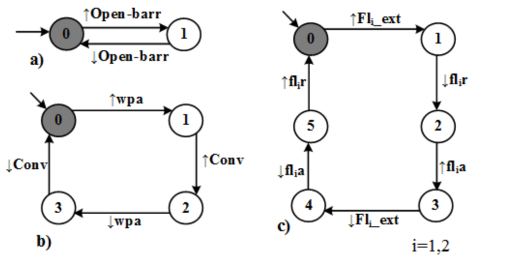


Fig. 9. Local controllers. (a) Barrier, (b) Conveyor, (c) Flippers.

TABLE II
GLOBAL SAFETY AND LIVENESS CONSTRAINTS

id	If	Then	
	C$^{(spec)}$	Ord$^{(spec)}$	Inh$^{(spec)}$
1.	red	Fl$_1$_ext	
2.	met	Fl$_2$_ext	
3.	blk+fl$_1$a+fl$_2$a	Open_barr	
4.	wpf		Conv
5.	wpf		Open_barr
6.	wpf		Fl$_1$_ext
7.	wpf		Fl$_2$_ext

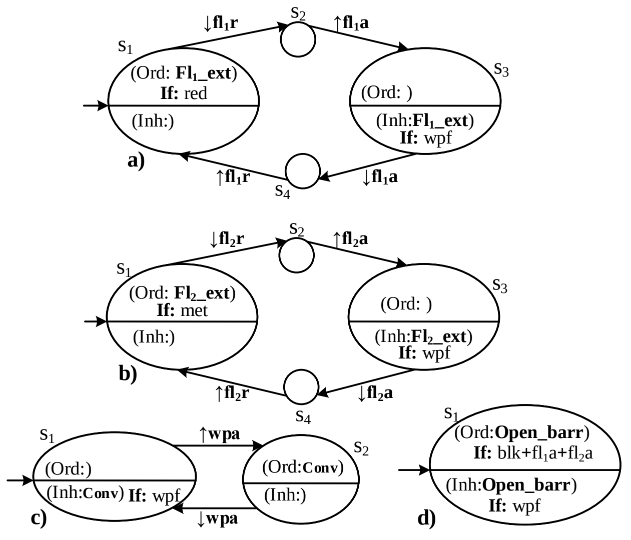


Fig. 10. Distributed controllers. (a) Flipper$_1$, (b) Flipper$_2$, (c) Conveyor, (d) Barrier.

B. Local safety and liveliness constraints

Local safety and liveness constraints restrict the PEs local behaviors in order to avoid the subsystems local functioning problems. Table I presents the local safety and liveness constraints to apply to the PEs of the sorting system. We note that local safety and liveness constraints are obtained by a system expert. For instance, the safety constraint "↑Conv **And Not** ↑wpa = 0" reflects the fact that the conveyor should not start (activate order "Conv") if no workpiece is detected by the sensor "wpa" in its input.

C. Local controllers

The application of the local constraints of Table I to the corresponding PEs of Fig. 8 allows obtaining the LCs shown in Fig. 9.

D. Global constraints

Global constraints allow a cooperative interaction among the sorting station PEs. The global constraints to be applied to the considered PEs are stipulated in Table II.

E. Distributed controllers

The global constraints presented in Table II are applied to the corresponding LCs according to the global control synthesis algorithm previously presented in [8]. For example, to establish the DC corresponding to the first flipper (Fig. 10(a)), the global constraints (1) and (6) from Table II are applied to the flipper$_1$'s LC (Fig. 9(c)). The constraint (1) allocates the flipper$_1$ to advance (activate order "Fl$_1$_ext") when a red workpiece is detected (sensor signal "red"). The condition "red" is then associated to the flipper$_1$ LC (after its aggregation), where the order "Fl$_1$_ext" is authorized. The constraint (6) allocates the flipper$_1$ to retract (deactivate order "Fl$_1$_ext") when a workpiece falls at one of the 3 sliders (sensor signal "wpf"). The condition "wpf" is then associated with the flipper$_1$ LC (after aggregation), where the order "Fl$_1$_ext" is inhibited.

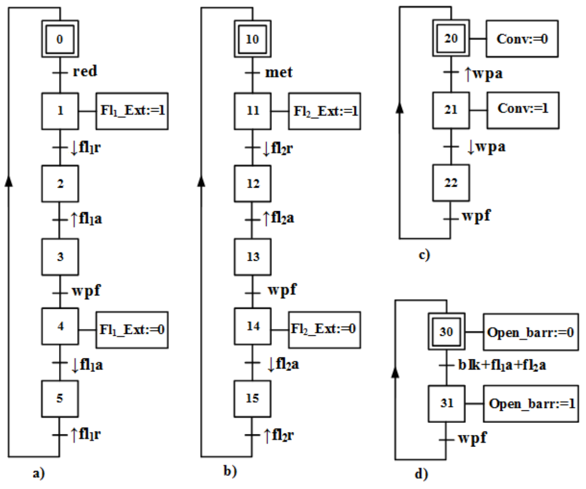
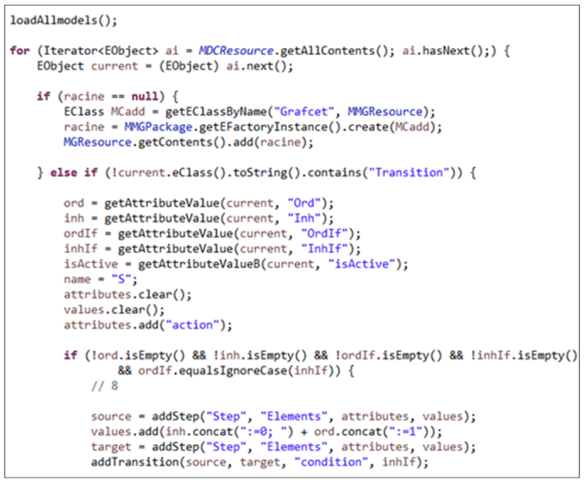


Fig. 11. Grafcet interpretation of the corresponding DCs. (a) Flipper$_1$, (b) Flipper$_2$, (c) Conveyor, (d) Barrier.


```
loadAllmodels();

for (Iterator<EObject> ai = MDCResource.getAllContents(); ai.hasNext();) {
    EObject current = (EObject) ai.next();

    if (racine == null) {
        EClass MCadd = getEClassByName("Grafcet", MMGResource);
        racine = MMGPackage.getEFactoryInstance().create(MCadd);
        MGResource.getContents().add(racine);

    } else if (!current.eClass().toString().contains("Transition")) {

        ord = getAttributeValue(current, "Ord");
        inh = getAttributeValue(current, "Inh");
        ordIf = getAttributeValue(current, "OrdIf");
        inhIf = getAttributeValue(current, "InhIf");
        isActive = getAttributeValueB(current, "isActive");
        name = "S";
        attributes.clear();
        values.clear();
        attributes.add("action");

        if (!ord.isEmpty() && !inh.isEmpty() && !ordIf.isEmpty() && !inhIf.isEmpty()
                && ordIf.equalsIgnoreCase(inhIf)) {
            // 8

            source = addStep("Step", "Elements", attributes, values);
            values.add(inh.concat(":=0; ") + ord.concat(":=1"));
            target = addStep("Step", "Elements", attributes, values);
            addTransition(source, target, "condition", inhIf);
```

Fig. 12. Transformation function

F. *Grafcet implementation*

The final step of the approach is to interpret the obtained DCs into Grafcet language according to the transformation rules presented in Section III.A.

Fig. 11 presents the partial Grafcets corresponding to the DCs of the studied system.

V. Tool support

In this Section, we show how the Grafcet models of the studied system DCs have been produced with the software tool support.

First, let us mention that the rules presented in Figs. 2 and 3 have been implemented in Java/EMF environment. The execution of the rules is defined in a method called Transformation (), which uses directly or indirectly several developed methods. In a general manner, the method loads both metamodels, the state machine model and an empty Grafcet model. Fig. 12 shows an example of the implementation of rule 8 from Table 1 inside the transformation method. This rule describes the creation of a Grafcet element. Second, we represent our DCs models. In Fig. 13 (Appendix) we have created four different models in a tree structure form representing the system's corresponding DCs. It is also possible to read each model in a textual form. Fig. 14 (Appendix) shows a screenshot of the four Grafcet models obtained from the previous DCs models through our developed software tool. The results correspond to what was expected in Section IV.F.

VI. Conclusions and prospects

This paper presented a MDE method, which aims at automatically generating distributed PLC-Based control of AMS in the form of Grafcet. The objective is to support control systems practitioners in realizing their control design and implementation tasks, based on a formal method. The approach was applied to an experimental AMS to evaluate its applicability and feasibility. Nevertheless, let us mention that other functionalities are still under development. Further development stages will emphasize on refining the tool by: (i) considering the DC design step based on a library of manufacturing components models and interfaces to formally specify the control specifications; and (ii) taking into account the graphical representation of the resulting Grafcet.

References

[1] P. J. Ramadge and W. M. Wonham, "Supervisory control of a class of discrete event processes," *SIAM journal on control and optimization*, vol. 25, no. 1, pp. 206–230, 1987.

[2] W. M. Wonham and P. J. Ramadge, "Modular supervisory control of discrete-event systems," *Mathematics of control, signals and systems*, vol. 1, no. 1, pp. 13–30, 1988.

[3] K. C. Wong and W. M. Wonham, "Hierarchical control of discrete-event systems," *Discrete Event Dynamic Systems*, vol. 6, no. 3, pp. 241–273, 1996.

[4] L. Feng and W. M. Wonham, "Supervisory control architecture for discrete-event systems," *IEEE Transactions on Automatic Control*, vol. 53, no. 6, pp. 1449–1461, 2008.

[5] H. Hu, R. Su, M. Zhou, and Y. Liu, "Polynomially complex synthesis of distributed supervisors for large-scale amss using petri nets," *IEEE Transactions on Control Systems Technology*, vol. 24, no. 5, pp. 1610–1622, 2015.

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

[7] ——, "Distributed supervisory control synthesis for discrete manufacturing systems," *IFAC-PapersOnLine*, vol. 49, no. 12, pp. 396–401, 2016.

[8] ——, "A synthesis approach to distributed supervisory control design for manufacturing systems with grafcet implementation," *International journal of Production Research*, pp. 1–21, September 2016.

[9] IEC, "60848, grafcet specification language for sequential function charts," 2013.

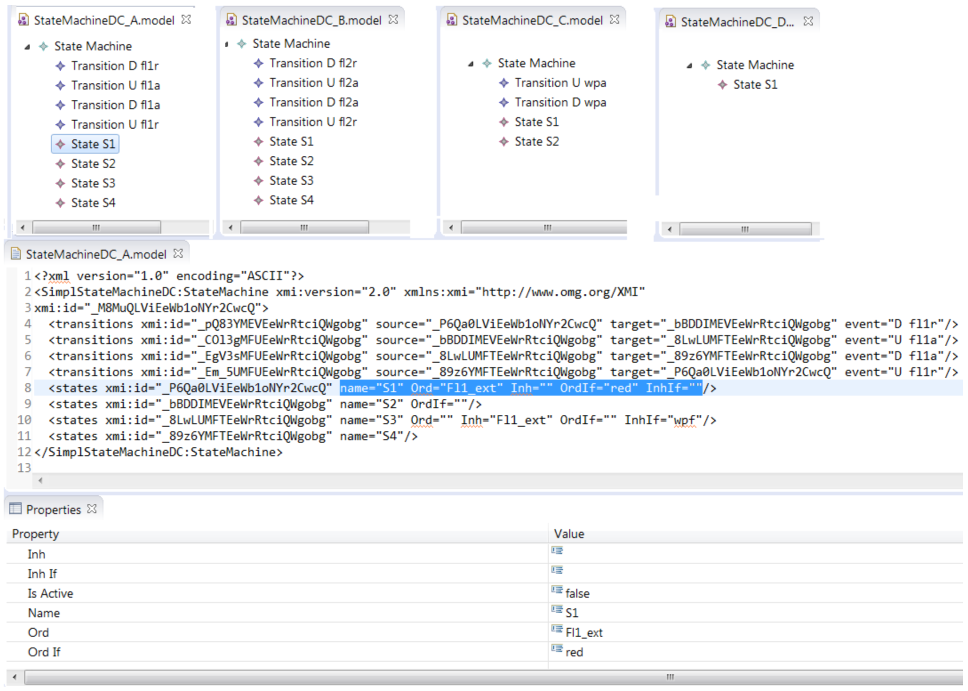
[10] M. Didonet Del Fabro and P. Valduriez, "Towards the efficient development of model transformations using model weaving and matching transformations," *Software and Systems Modeling*, vol. 8, no. 3, pp. 305–324, 2009.

[11] Z. Drey, C. Faucher, F. Fleurey, V. Mahé, and D. Vojtisek, "Kermeta language," *Reference Manual*, 2009.

[12] A. Philippot, M. Sayed-Mouchaweh, and V. Carré-Ménétrier, "Modelling of a discrete manufacturing system by parts of plant," *IFAC Proceedings Volumes*, vol. 42, no. 4, pp. 343–348, 2009.

[13] Eclipse, "2011. java emitter templates," http://www.eclipse.org/modeling/m2t/?project=jet, 2011.

APPENDIX : INPUT/OUTPUT OF THE TOOL


StateMachineDC_A.model
- State Machine
  - Transition D fl1r
  - Transition U fl1a
  - Transition D fl1a
  - Transition U fl1r
  - State S1
  - State S2
  - State S3
  - State S4

StateMachineDC_B.model
- State Machine
  - Transition D fl2r
  - Transition U fl2a
  - Transition D fl2a
  - Transition U fl2r
  - State S1
  - State S2
  - State S3
  - State S4

StateMachineDC_C.model
- State Machine
  - Transition U wpa
  - Transition D wpa
  - State S1
  - State S2

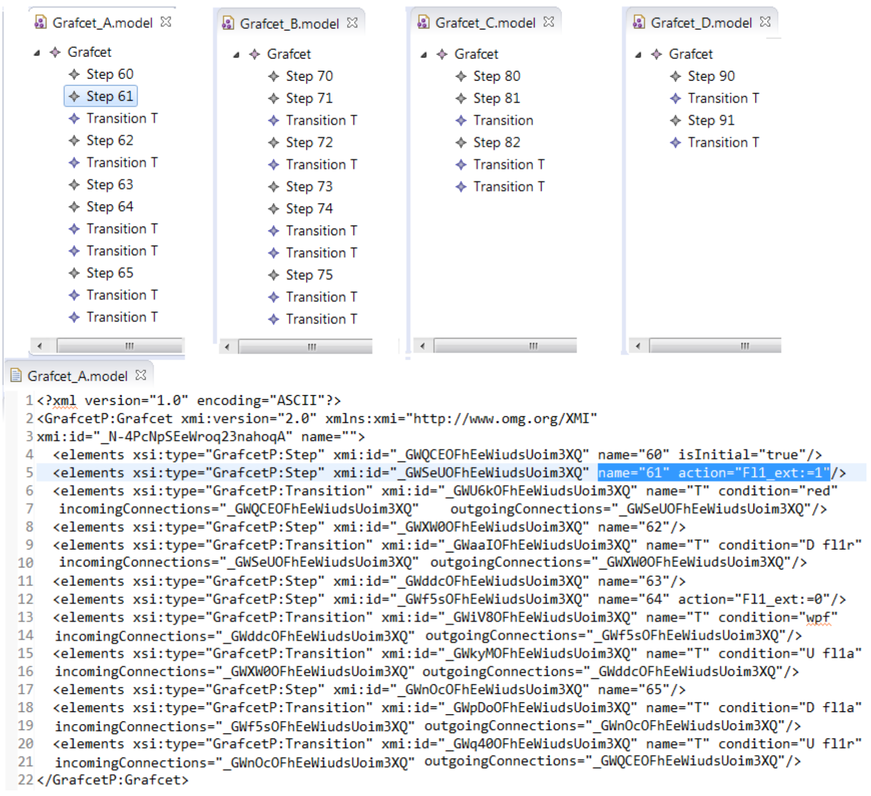
StateMachineDC_D...
- State Machine
  - State S1

StateMachineDC_A.model
```
<?xml version="1.0" encoding="ASCII"?>
<SimplStateMachineDC:StateMachine xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
xmi:id="_M8MuQLViEeWb1oNYr2CwcQ">
  <transitions xmi:id="_pQ83YMEVEeWrRtciQWgobg" source="_P6Qa0LViEeWb1oNYr2CwcQ" target="_bBDDIMEVEeWrRtciQWgobg" event="D fl1r"/>
  <transitions xmi:id="_COl3gMFUEeWrRtciQWgobg" source="_bBDDIMEVEeWrRtciQWgobg" target="_8LwLUMFTEeWrRtciQWgobg" event="U fl1a"/>
  <transitions xmi:id="_EgV3sMFUEeWrRtciQWgobg" source="_8LwLUMFTEeWrRtciQWgobg" target="_89z6YMFTEeWrRtciQWgobg" event="D fl1a"/>
  <transitions xmi:id="_Em_5UMFUEeWrRtciQWgobg" source="_89z6YMFTEeWrRtciQWgobg" target="_P6Qa0LViEeWb1oNYr2CwcQ" event="U fl1r"/>
  <states xmi:id="_P6Qa0LViEeWb1oNYr2CwcQ" name="S1" Ord="Fl1_ext" Inh="" OrdIf="red" InhIf=""/>
  <states xmi:id="_bBDDIMEVEeWrRtciQWgobg" name="S2" OrdIf=""/>
  <states xmi:id="_8LwLUMFTEeWrRtciQWgobg" name="S3" Ord="" Inh="Fl1_ext" OrdIf="" InhIf="wpf"/>
  <states xmi:id="_89z6YMFTEeWrRtciQWgobg" name="S4"/>
</SimplStateMachineDC:StateMachine>
```

Properties

Property	Value
Inh	
Inh If	
Is Active	false
Name	S1
Ord	Fl1_ext
Ord If	red

Fig. 13. DCs input models


Grafcet_A.model
- Grafcet
  - Step 60
  - Step 61
  - Transition T
  - Step 62
  - Transition T
  - Step 63
  - Step 64
  - Transition T
  - Transition T
  - Step 65
  - Transition T
  - Transition T

Grafcet_B.model
- Grafcet
  - Step 70
  - Step 71
  - Transition T
  - Step 72
  - Transition T
  - Step 73
  - Step 74
  - Transition T
  - Transition T
  - Step 75
  - Transition T
  - Transition T

Grafcet_C.model
- Grafcet
  - Step 80
  - Step 81
  - Transition
  - Step 82
  - Transition T
  - Transition T

Grafcet_D.model
- Grafcet
  - Step 90
  - Transition T
  - Step 91
  - Transition T

Grafcet_A.model
```
<?xml version="1.0" encoding="ASCII"?>
<GrafcetP:Grafcet xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
xmi:id="_N-4PcNpSEeWroq23nahoqA" name="">
  <elements xsi:type="GrafcetP:Step" xmi:id="_GWQCEOFhEeWiudsUoim3XQ" name="60" isInitial="true"/>
  <elements xsi:type="GrafcetP:Step" xmi:id="_GWSeUOFhEeWiudsUoim3XQ" name="61" action="Fl1_ext:=1"/>
  <elements xsi:type="GrafcetP:Transition" xmi:id="_GWU6kOFhEeWiudsUoim3XQ" name="T" condition="red"
   incomingConnections="_GWQCEOFhEeWiudsUoim3XQ"    outgoingConnections="_GWSeUOFhEeWiudsUoim3XQ"/>
  <elements xsi:type="GrafcetP:Step" xmi:id="_GWXW0OFhEeWiudsUoim3XQ" name="62"/>
  <elements xsi:type="GrafcetP:Transition" xmi:id="_GWaaIOFhEeWiudsUoim3XQ" name="T" condition="D fl1r"
   incomingConnections="_GWSeUOFhEeWiudsUoim3XQ" outgoingConnections="_GWXW0OFhEeWiudsUoim3XQ"/>
  <elements xsi:type="GrafcetP:Step" xmi:id="_GWddcOFhEeWiudsUoim3XQ" name="63"/>
  <elements xsi:type="GrafcetP:Step" xmi:id="_GWf5sOFhEeWiudsUoim3XQ" name="64" action="Fl1_ext:=0"/>
  <elements xsi:type="GrafcetP:Transition" xmi:id="_GWiV8OFhEeWiudsUoim3XQ" name="T" condition="wpf"
  incomingConnections="_GWddcOFhEeWiudsUoim3XQ" outgoingConnections="_GWf5sOFhEeWiudsUoim3XQ"/>
  <elements xsi:type="GrafcetP:Transition" xmi:id="_GWkyMOFhEeWiudsUoim3XQ" name="T" condition="U fl1a"
  incomingConnections="_GWXW0OFhEeWiudsUoim3XQ" outgoingConnections="_GWddcOFhEeWiudsUoim3XQ"/>
  <elements xsi:type="GrafcetP:Step" xmi:id="_GWnOcOFhEeWiudsUoim3XQ" name="65"/>
  <elements xsi:type="GrafcetP:Transition" xmi:id="_GWpDoOFhEeWiudsUoim3XQ" name="T" condition="D fl1a"
  incomingConnections="_GWf5sOFhEeWiudsUoim3XQ" outgoingConnections="_GWnOcOFhEeWiudsUoim3XQ"/>
  <elements xsi:type="GrafcetP:Transition" xmi:id="_GWq40OFhEeWiudsUoim3XQ" name="T" condition="U fl1r"
  incomingConnections="_GWnOcOFhEeWiudsUoim3XQ" outgoingConnections="_GWQCEOFhEeWiudsUoim3XQ"/>
</GrafcetP:Grafcet>
```

Fig. 14. Grafcet models produced by the tool support