International Journal of Production Research

ISSN: 0020-7543 (Print) 1366-588X (Online) Journal homepage: http://www.tandfonline.com/loi/tprs20

Towards an approach of synthesis, validation and implementation of distributed control for AMS by using events ordering relations

Yassine Qamsane , Abdelouahed Tajer & Alexandre Philippot

To cite this article: Yassine Qamsane , Abdelouahed Tajer & Alexandre Philippot (2017): Towards an approach of synthesis, validation and implementation of distributed control for AMS by using events ordering relations, International Journal of Production Research

To link to this article: http://dx.doi.org/10.1080/00207543.2017.1333648

Published online: 01 Jun 2017.

International Journal of Production Research, 2017
https://doi.org/10.1080/00207543.2017.1333648

Towards an approach of synthesis, validation and implementation of distributed control for AMS by using events ordering relations

Yassine Qamsane[a]*, Abdelouahed Tajer[a] and Alexandre Philippot[b]

[a]*Laboratory of Electrical Engineering and Control Systems, National School of Applied Sciences, Cadi Ayyad University, Marrakech, Morocco;* [b]*Research Centre for Science and Information Technology and Communication, University of Reims Champagne Ardennes, Reims, France*

(*Received 1 December 2016; accepted 12 May 2017*)

We study distributed control synthesis and validation for automated manufacturing systems (AMS) in the framework of supervisory control theory. To reduce the size of the control problem, we view an AMS as comprised of asynchronous subsystems which are coupled through imposed logical Boolean specifications. The principle of the distributed control approach is the decomposition of the global monolithic control action into local coordinated control strategies for the individual subsystems. Owing to its importance in a distributed scheme, the order in which events occur arouses interest. By extending our previous results, we develop a set of rules of events precedence ordering, under which the control strategy via decomposition promises the subsystems synchronisation and coordination. We show how these rules contribute to reduce the size of the controller models used in the verification/validation and implementation steps. The effectiveness of the proposed approach is demonstrated by means of an industrial AMS example.

Keywords: discrete-event systems; automated manufacturing systems; supervisory control synthesis; distributed control; verification & validation; model-checking

1. Introduction

Manufacturing systems are collaborative processes where several production units and humans are involved. Nowadays, the human intervention is increasingly reduced within manufacturing systems, in such a way that some processes are becoming entirely automated. Automated manufacturing systems (AMS) are highly recommended in today's companies thanks to their attractive features, such as efficiency/reliability, reduced stocks, rapid market response and shorter lead times. Furthermore, the fast customer demand shift and legal requirements necessitates more flexibility and re-configurability from the AMS, i.e. AMS should be designed prone to rapid changes in their structures, either on hardware or on software components. Nevertheless, the implementation of flexibility and re-configurability concepts is uneasy to achieve in practice. Generally, the complexity of AMS control logic implementation increases proportionally with the increasing flexibility and re-configurability (Diogo et al. 2012).

Control development in current manufacturing practices is typically based on heuristic methods rather than formal ones. Effectively, for the control of small size systems, a simple, direct, and intuitive interpretation of the specifications into a control task may yield practical solutions. But as systems size becomes larger, intuitive methods become insufficient and formal ones more required. To support the control practitioner in his task, two main formal methods can be exploited: (i) control validation and verification (V&V); and (ii) supervisory control theory (SCT). The V&V consists of letting the control designer develop control models based on the specifications and then automatically analyse a formal representation of these models (Roussel et al. 2004). The models properties to be verified are safety, liveness, boundedness and absence of deadlocks. The validation consists of determining whether the model approves the designer's purpose. Reliable V&V techniques of control programmes are most often based on model-checking techniques (Baier and Katoen 2008). Because the V&V method is based on prior programmes writing, a major weakness of this method is the state-space explosion problem, which arises when dealing with large-scale systems. Another drawback of V&V is that a programmer cannot fully validate a programme when a change is carried out on expected properties. The SCT (Ramadge and Wonham 1987) is a synthesis method which consists of systematically constructing control models, which guarantee by construction the respect of the specified properties. These are therefore taken into account from the beginning of the design, which avoids human errors during the control synthesis, and enables swift reaction to blockings

*Corresponding author. Email: qamsaneyassine@gmail.com and changes in case of redesigns. Because SCT makes use of state models and their composition for the construction of the plant and specification models, the state-space explosion and models interpretation represent major challenges. To solve this impediment, we approach the problem of designing distributed controllers for discrete AMS by combining both SCT with V&V. The objective is to directly deduce control laws from the specifications, without involving the control designer (at least a minimum involvement).

We recently developed an SCT-based synthesis method which enables deducing a consistent formal description of distributed controllers (DC) (Qamsane, Tajer, and Philippot 2016a, 2016b). We model the plant by collections of local modular deterministic automata, and the control specifications by collections of Boolean expressions. The latter have the ability to be applied locally without going through a composition step, which makes them a very suitable alternative to automata for performing synthesis. Though this distributed control synthesis strategy arouses to overcome the computational complexity problem, its use of different modelling formalisms (automata with algebraic Boolean expressions) may lead to deadlocks and/or disrupt the optimality of the global behaviour. To overcome this issue, V&V is solicited to analyse the global behaviour of the system in order to validate the obtained DC. We use a V&V method to detect possible deadlocks or violated constraints and support the designer to make the necessary modifications. Once the designed distributed control satisfies the requested properties, the approach proposes a method to interpret the obtained controller into Grafcet (IEC Standard 60848 2013), which is a standard widely used in the manufacturing domain to specify the functional behaviour of control systems.

In this paper, we stretch our previous work in (Qamsane, Tajer, and Philippot 2016a, 2016b) by branding two contributions. First, as we use a V&V technique for the verification/validation of the synthesised DC, we propose a methodology for transforming the DC models into the syntax of Uppaal software using a procedure based on graph grammars. Uppaal (Behrmann, David, and Larsen 2006) is an integrated environment for modelling, verification (via automatic model-checking) and validation (via graphical simulation) of systems modelled as networks of automata. Second, we present the concept of precedence relation for introducing events ordering relationships within the distributed components of an AMS. This concept allows establishing particular order among the events that occur according to certain specifications, which promises coordination and synchronous execution of the global system. Based on this concept, we established a set of simplification rules, which aid in simplifying the global specifications and make it easier to transform the synthesised controller into simpler models to be used for the verification/validation with Uppaal, and for Grafcet implementation.

The rest of this paper is organised as follows. Section 2 provides preliminary information about SCT; then the basic concepts of our distributed supervisory control approach, previously proposed, are recalled. Section 3 introduces the distributed control verification/validation methodology, which is based on Uppaal software. Section 4 is devoted to introduce the formalism of precedence relations amongst events, which provides a formal basis for the control synthesis, verification/validation and implementation. Section 5 illustrates a part of the experimental AMS used in (Qamsane, Tajer and Philippot 2016b) to verify the usefulness of the new results. In Section 6, a comparison of the presented approach with some existing SCT forcing event and PLC-based approaches is carried out. Finally, Section 7 outlines the contribution of this paper, and draws conclusions and perspectives.

2. Notation and preliminaries

In this section, an overview of the SCT and our previously proposed distributed control synthesis approach follow.

2.1 *Supervisory control theory*

SCT initiated by Ramadge and Wonham (RW) (Ramadge and Wonham 1987) extends some continuous systems control theory concepts to the DES. The RW model separates among the free behaviour of a system, modelling its entire potentially realisable operation (open loop operation), and its desired behaviour (closed loop operation). In SCT, a system is assumed to evolve spontaneously, executes sequences of events that describe its behaviour, and generates a language constructed by the events alphabet. Events are divided into two disjoint sets, controllable and uncontrollable events. The objective is to synthesise a supervisor(s) which disables the occurrence of controllable events, in such a way to force the supervised system to behave according to the desired specifications. A DES is formally represented by the quintuple $G = (Q, \Sigma, \delta, Q_m, q_0)$, where Q is a finite set of *states*, with $q_0 \in Q$, the *initial state* and $Q_m \in Q$, the set of *target states*, Σ is a finite set of events called an *alphabet*, and δ is a *transition function* $\delta: Q \times \Sigma \rightarrow Q$.

Let Σ^* denotes the set of all finite events concatenations in Σ. An element of Σ^* is called a string. The length of a string is given by the number of its events. The empty string denoted ε is the string with no element. If u, v and ω, are strings in Σ^*, u is a prefix of ω if $uv = \omega$. A set that contains all the prefixes of all its elements is said to be *prefix* *closed*. A subset $L \subseteq \Sigma^*$ is called a language over Σ. The language noted $\bar{L}$containing all the prefixes of strings in L defines the prefix-closure of the language L. The language generated by G can be defined as: $L(G) = \{\omega \in \Sigma^* | \delta\ (q_0, \omega)$ is defined$\}$. $L(G)$ is also prefix-closed. The *marked language*, denoted $L_m(G) \subseteq L(G)$ is the language consisting of all strings which reach marked states. Formally, $L_m(G)$ is given by: $L_m(G) = \{\omega \in L(G) | \delta(q_0, \omega) \in Q_m\}$. A DES is said to be non-blocking if $L(G) = \overline{L_m(G)}$.

In some DES applications, several independent processes are considered simultaneously. To combine two DES (A and B) into one single more complex DES, i.e. $C = A||B$, a procedure called synchronous product (parallel composition) is used. In the resulting automaton, common events occur synchronously, while the other events occur asynchronously (Wonham 2012). The inverse operation is called natural projection; it consists of projecting a monolithic DES model into suitable subsystems. For $\Sigma' \subseteq \Sigma$, the natural projection P: $\Sigma^* \to \Sigma'^*$ is defined by:

$$\begin{aligned} P(\varepsilon) &= \varepsilon. \\ P(\sigma) &= \begin{cases} \varepsilon, & \text{if } \sigma \notin \sum', \\ \sigma, & \text{if } \sigma \in \sum'; \end{cases} \\ P(\omega\sigma) &= P(\omega)P(\sigma), \omega \in \Sigma^*, \sigma \in \Sigma. \end{aligned}$$

The sets of controllable and uncontrollable events are denoted by Σ_c and Σ_{uc}, respectively. The supervisor can disable only controllable events and has no effect on uncontrollable ones. The existence of a supervisor is guaranteed by the controllability condition. The latter is given by $\bar{K} \cdot \Sigma_{uc} \cap L(G) \subseteq \bar{K}$; where $L(G)$ is the potentially possible behaviour and K is the desired behaviour. This condition denotes that K is controllable, if for any sequence of events ω that starts from a sequence which is already a prefix of K ($\omega \in K$), the occurrence of an uncontrollable event doesn't lead the sequence out of the desired behaviour K.

2.2 *Concepts of the distributed controller design*

Distributed control approaches assume that a system is composed of several interconnected subsystems. The subsystems share information among each other in order to reach a global goal. The idea is to design loosely coupled local controllers (LCs) for the subsystems, and for the global couplings, those LCs are required to share enough information among each other to cooperatively execute the control actions (see e.g. Hu et al. 2015).

This paper deals with distributed AMS which execute events in several distant sites. In order to reduce the computational complexity, we divide the control problem according to two levels: a low-level where each local site is partitioned into communicating sub-components; and a high-level where these local sites are also required to communicate with each other through communication channels. Loosely coupled controllers are designed for each local site components. These collaborate with each other and with the other neighbouring sites' components through global logical specifications, such that the global behaviour is equivalent to that of the original global system. For instance, let us consider the case of the AMS of Figure 1, which is composed of three manufacturing sites working on a same product. Each site is composed of a set of components. The problem is that a component (resp., a site) observes only the events of its locus

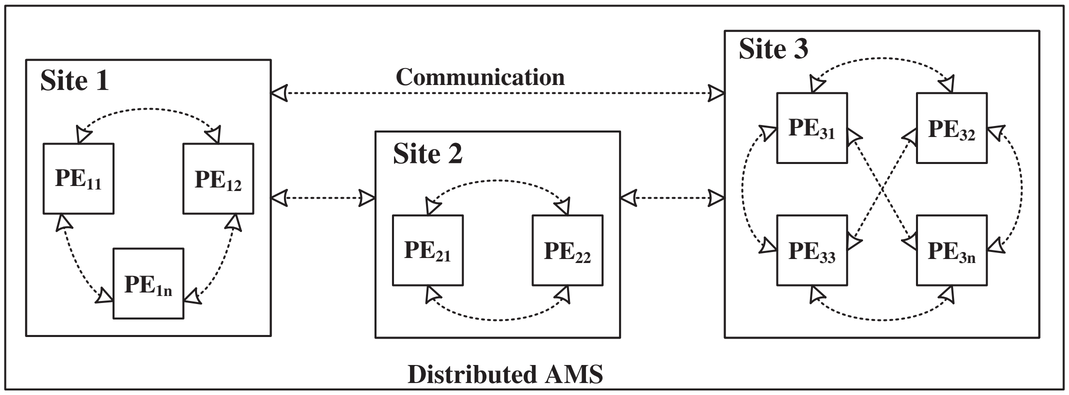


Figure 1. Communicating components in a distributed AMS.

and may have to perform actions that depend on the up-to-date global state of the site (resp., the overall system). To solve this problem, we allow the components (resp., the local sites) to exchange information among each other through global constraints formalised as Boolean expressions.

Within our distributed control architecture (Qamsane, Tajer, and Philippot 2016b), low-level modular plant models are obtained according to the plant mechanical characteristics (sensors and actuators). The obtained local automata models are called plant elements (PEs). First, local control synthesis is carried out by applying local safety and liveness constraints, expressed as logical Boolean equations, to their corresponding local PEs. Second, the application of the global constraints, also defined as Boolean expressions, to the LCs provides DCs, which allow a cooperative interaction among the modular PEs. The main basic concepts of the synthesis approach are reminded in this section.

2.2.1 *Plant modelling*

A practical plant modelling method has been developed in our previous work (Philippot 2006). The method is based on a low-level study of the functional chain of a process where the plant is composed of actions and acquisitions chains. Actions chain is composed of pre-actuators, actuators and effectors. Pre-actuators manage the power originating from the control part and transmit it to actuators which convert it into actions on the effectors. The acquisition chain allows through sensors to acquire and transmit effector information to the control part. PEs modelling consists of representing this functional chain through these elements. A PE is an event-driven model obtained by parallel composition of a sensors model with an actuator model, which is obtained from a pre-actuator one. In a PE model, it is assumed that the set of controllable events corresponds either to the activation orders '$\uparrow Z$' or the deactivation orders '$\downarrow Z$' of the controller, and the set of uncontrollable events is associated with the rising edges '$\uparrow E$' or with the falling edges '$\downarrow E$' of its inputs, i.e. $\Sigma_c = \uparrow Z \cup \downarrow Z$ and $\Sigma_{uc} = \uparrow E \cup \downarrow E$. The resulting PE model is an automaton $G^{(PEi)} = (Q^{(PEi)}, \Sigma^{(PEi)}, \delta^{(PEi)}, q_0{}^{(PEi)})$, where $Q^{(PEi)}$ is the set of states, $\Sigma^{(PEi)}$ is the alphabet of events, $\delta^{(PEi)}$: $Q^{(PEi)} \times \Sigma^{(PEi)} \rightarrow Q^{(PEi)}$ is a transition function and $q_0{}^{(PEi)}$ is the initial state.

2.2.2 *Local constraints modelling*

Local constraints to model are of two types: safety constraints (what the system must not do), and liveness constraints (what the system must do). Integrating these local constraints consists of inhibiting actions and/or sequencing the execution of commands to be sent to each individual PE. In DES, constraints are originally expressed with automata which don't immune design and interpretation problems. To surpass these difficulties, we propose to use Boolean equations instead of automata. The work presented in (Roussel et al., 2004) also uses a Boolean algebraic modelling to develop a formal synthesis method. It allows to describe behaviours integrating temporal aspects, by manipulating binary signals over time and Boolean variables. Our work uses only classical Boolean algebra without temporal quantifiers. We use the Boolean operators (And, Or, Not) to express the consequences of events occurrences ($f((\uparrow\downarrow)e_i, (\uparrow\downarrow)z_j)$), over the activation/deactivation of the control outputs $(\uparrow\downarrow)z_k$. Accordingly, local safety and liveness constraints are represented by equations whose results must be equal to 0 (not to do) and equal to 1 (to do), respectively. These considerations are reflected in the following algebraic expression:

$$f\big((\uparrow\downarrow)e_i,\ (\uparrow\downarrow)z_j\big)\ \textbf{And}\ (\dot{\uparrow\downarrow})z_k = 0\ (=1).$$

2.2.3 *Synthesis of the local controllers*

Basically, the synthesis of a supervisor(s) consists of defining a model which forces the supervised system to behave according to the specifications by disabling some controllable events occurrences. Local controller synthesis consists of considering the logical Boolean expressions to the PEs automata. The principle is to prohibit certain controllable events from occurring to prevent a PE from reaching the states not meeting the specifications. For each individual PE, a produced LC model is an automaton $G^{(LCi)} \subseteq G^{(PEi)}$, $G^{(LCi)} = (Q^{(LCi)}, \Sigma^{(LCi)}, \delta^{(LCi)}, q_0{}^{(LCi)})$.

2.2.4 *Global constraints modelling*

An AMS is composed of collections of relatively small sized, local, interacting, asynchronous and event-driven subsystems. Its execution is an interactive synchronisation of these local subsystems on shared actions (Hu et al. 2015). Global constraints are specified over these subsystems in order to reach the global goal. In order to overcome the complexity related to automata modelling, which often uses a composition step, we adopt a logical Boolean modelling formalism. The formalisation of the global constraints starts by establishing the system signals encompassed within the informal specifications. It consists of defining which subsystem actuator is authorised (activated) or inhibited (deactivated), and under which circumstances. Each informal global constraint is interpreted according to the following basic Boolean expression:

If (*Condition*) **then** (*Action*).

A *condition* over the activation/deactivation of an actuator action could be composed in a combinatorial way with the Boolean operations: And, Or, Not, respectively, noted as $\wedge$, $\vee$, $\neg$; the symbols $\uparrow$, $\downarrow$, respectively, referring to the rising and falling edges associated with events; and the symbol $\rightarrow$, denoting the precedence relation (this relation shall be explained in Section 4). An *action* defines the authorisation (resp., inhibition) of a control order to be sent to activate (resp., deactivate) an actuator. The activation and deactivation of a control order correspond to its enforcement to 1 and to 0, respectively. Thus, the activation is interpreted by the function *Ord*(*order*), which indicates the authorised action; and the deactivation is interpreted by the function *Inh*(*order*), which indicates the inhibited action. Accordingly, the formal global constraints are defined by the pair $Spec = (C^{(spec)}, Act^{(spec)})$, where $C^{(spec)}$ is the set of specifications' conditions; and $Act^{(spec)} = \{Ord^{(spec)}, Inh^{(spec)}\}$ is the set of specifications activation/deactivation actions, with $Ord^{(spec)}$ the set of authorised control actions, and $Inh^{(spec)}$ the set of inhibited ones.

If several global constraints allow the authorisation (resp, inhibition) of a same order, then all of these constraints' conditions are associated by conjunction.

Example 1:

Let us consider the simple pneumatic system shown in Figure 2, which consists of two juxtaposed single-acting cylinders and a push button. We study the system in a modular fashion, then, we consider each cylinder separately. Let us also consider the global specifications defined for the system to reach a global goal as follows: (1) Mutual exclusion between the two cylinders operations. (2) Cylinder 1 extends only if the cylinder 2 is retreated; (3) Cylinder 1 extends if the push button is pressed.

The first constraint reflects that the extension operation of the cylinder 1 (resp., cylinder 2) is authorised only if the cylinder 2 (resp., cylinder 1) is not moving. It could be represented by the following two Boolean expressions:

(a) **If** $\neg$*Extend2* **Then** *Ord*(*Extend1*),
(b) **If** $\neg$*Extend1* **Then** *Ord*(*Extend2*).

The second global constraint reflects that the cylinder 1 is authorised to extend only if the cylinder 2 is retreated (indicated by the sensor c_{21}). It could be represented by the Boolean expression:

(c) **If** c_{21} **Then** *Ord*(*Extend1*).

The third constraint states that the authorisation of the extension operation of the cylinder 1 is also conditioned by pressing the push button, which could be modelled by:

(d) **If** *pb* **Then** *Ord*(*Extend1*).

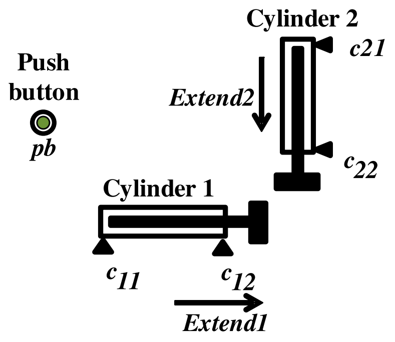


Figure 2. A simple illustrative pneumatic system.

We mention that the authorisation of the order *Extend1* is conditioned within the three global constraint expressions (a), (c) and (d), then, all these constraints' conditions are allied by conjunction to provide a single global constraint as follows:

If ($pb \wedge c_{21} \wedge \neg Extend2$) **Then** $Ord(Extend1)$.

2.2.5 *Synthesis of the distributed controllers*

The synthesis of the DCs consists of integrating the global constraints to the designed LCs. The main idea of the global control synthesis algorithm (Qamsane, Tajer, and Philippot 2016a, 2016b) is to firstly provide an abstraction of the LCs where the states linked with controllable events ($z_i \in \Sigma_c$) are merged into macro-states connected with uncontrollable ones ($e_j \in \Sigma_{uc}$). The principle is to mask the controllable evolutions of a LC using the natural projection operation, then join them into macro-states as follows: a controllable event that is associated with a rising edge is an authorised order, then it belongs to the set $Ord^{(DC)}$; a controllable event that is associated with a falling edge is an inhibited order, then it belongs to the set $Inh^{(DC)}$. The obtained model is called an aggregated local controller (ALC). Secondly, the global control synthesis algorithm considers the global constraints to the ALC. This step receives as input, an ALC and a set of global constraints. It systematically checks all the constraints for each ALC state. In case an authorised (resp., inhibited) order of an ALC state is identical to that authorised (resp., inhibited) within a global constraint, then the constraint's condition ($C^{(spec)}$) should be associated with this state to condition the authorisation (resp., the inhibition) of the corresponding order. The final output of the algorithm is the resulting DC.

2.2.6 *Verification of the deadlock freeness and maximally optimal liveness*

As there is always a conflict between distributed control and the maximal permissiveness, we use techniques of formal verification in order to confirm whether deadlocks among DCs are possible and guarantee the maximally optimal liveness. The verification/validation methodology will be explained in Section 3.

2.2.7 *Grafcet implementation of the distributed controllers*

A straightforward method for the interpretation of the DCs into Grafcet formalism, in order to facilitate their PLC-implementation, was previously proposed in (Qamsane, Tajer and Philippot 2016b). Therein, the reader could find interpretation rules defined to ensure a correct transformation of the DCs into Grafcet.

Section 4 presents some events conditions that could simplify the interpreted Grafcet structures by introducing partial ordering among events.

3. From automatic synthesis to validation of the distributed control

We consider an AMS 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 PEs evolve, their local operation doesn't change. Thus, a set of immutable local constraints (defined by a system expert) is retained for each PE (unless a change of event labels). The application of these constraints to the PEs models allows obtaining immutable LCs. To reach the global goal, global constraints are specified over the subsystems, and then integrated to the LCs. However, the use of automata with Boolean expressions for performing distributed synthesis may spark a conflict between the distributed control and the maximal permissiveness. To raise this challenge, the proposed approach solicits V&V to check that the distributed controller complies with the formal requirements. Among the various V&V techniques, model-checking (Baier and Katoen 2008) has proved to be remarkably accurate in the verification of developed designs. It consists of partially calculating the set of reachable state-transitions of a model, then evaluating if a formal constraint expressed in temporal logic holds for that set. In order to apply model-checking to the DCs, these have to be transformed into a standard state-transition model.

Figure 3 summarises the control design with the V&V procedure. The verification of the consistency and refinement (if necessary) of the DCs is essential to validate them against the control specifications. Typically, the validation of a description D_1 against a description D_2 consists of checking that D_1 satisfies the properties specified in D_2, where D_2 is an informal or semi-formal description. It is a check 'is the specified system the one that the user wants', where D_2 is the informal set of user expectations (Lano 2012). If the analysis of the verification discloses that the specifications are

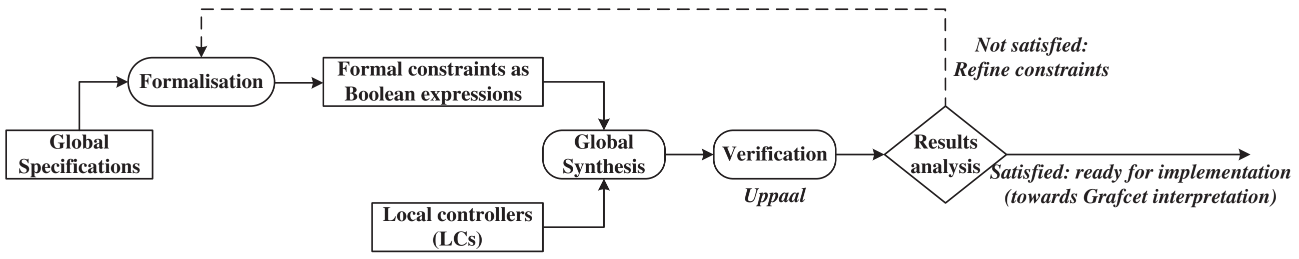


Figure 3. Control synthesis and verification workflow.

met, the controller can be automatically interpreted into Grafcet according to the method proposed in (Qamsane, Tajer, and Philippot 2016b), then programmed into the PLC. A negative verification result may be due to an incorrect formalisation of some global specifications, and then the erroneous expressions leading to the violation have to be refined.

In the following, we first recall the basic structure of a DC, then we present transformation outlines to convert DCs into the input language of Uppaal software afterwards.

3.1 *The DC automata semantics*

We describe a DC as a graphical structuring language for the control of distributed AMS. Syntax and semantic of a DC automaton were formally defined in (Qamsane, Tajer, and Philippot 2016a, 2016b). This syntax introduces a DC automaton as a 6-tuple $G^{(DC)} = (Q^{(DC)}, \Sigma^{(DC)}, \delta^{(DC)}, Act^{(DC)}, C^{(DC)}, q_0{}^{(DC)})$, 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 $q \in Q^{(DC)}$ 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 Boolean expressions that conditions the authorised and/or 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 Boolean 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 $(q, \sigma, q') \in \delta^{(DC)}$, where q is the origin state, σ is an uncontrollable event and q' is the destination state. It is assumed that the DCs automata are deterministic, that is, we cannot have two transitions with the same origin state, the same event and different destination states.

In a DC automaton, states are represented by nodes with their associated actions (authorised and/or inhibited orders) and their conditions, if any. If a condition concerns only the authorised (resp., the inhibited) order, then, the state is divided into two parts by a line. The initial state is indicated by an incoming arrow. Each arrow that links two states q and q', and labelled σ ($\sigma \in \Sigma_{uc}{}^{(DC)}$) represents a transition (q, σ, q'). Empty actions and conditions are represented by '–'.

3.2 *Representation of the DCs within Uppaal*

In order to check the properties of the synthesised distributed control, the obtained DCs automata have to be transformed into a formalism that allows appropriate analysis based on automatic verification software. We use Uppaal (Behrmann, David, and Larsen 2006) which is an integrated environment for modelling and verification/validation of systems modelled as networks of automata. The basic data structure in Uppaal is timed automata, which are an expansion of finite-state machines with clocks, guards and resets. Clocks evaluate time progress and it is allowed to test or to reset a clock value. A guard is a conjunction of clock conditions and/or conditions on the integer variables indicating when a transition is enabled. A synchronisation channel c can be defined for two automata which take a transition at the same time. Sending a message is represented by the discrete action $c!$, while receiving it is represented by $c?$. When taking a transition, two actions are possible: assignment of variables or reset of clocks.

As per our distributed approach, the synchronisation among distributed models is captured by synthesis through events, we use Uppaal (which is also applicable to standard finite-state automata) without time quantifiers. Uppaal comprises a graphical user interface for modelling automata and for an interactive simulation of the behaviour. To make use of Uppaal's features in verifying the DCs, it is necessary to convert their structure into Uppaal syntax.

Each DC is represented by a single automaton within Uppaal by following the conversion method below:

- The structure of a DC ordinary state can be reproduced by a location;
- A DC transition can be reproduced by a transition as follows: if the transition's event is associated with a rising edge, then it is updated to 1; otherwise, it is updated to 0.

- DC macro-states are reproduced as follows: (i) an authorised (resp., inhibited) order within a DC macro-state can be reproduced by a transition within Uppaal where the associated event is updated to 1 (resp., updated to 0). (ii) If an authorised (resp., inhibited) order is conditioned by a conjunction of Boolean variables (a Boolean condition), then this condition is reproduced as a guard indicating when the corresponding transition is enabled. If the condition contains a precedence relation amongst events, it can be reproduced according to the interpretations shown in Section 4, Figure 5. (iii) In a DC macro-state, it is assumed that the inhibition of orders takes priority over the authorisation, i.e. one must disable a command already activated before enabling an authorised one to ensure safety. Then, in a DC macro-state that contains both inhibition and authorisation of orders, transitions with inhibitions come before transitions with authorisations.

Moreover, the models of sensors that are not associated with any actuator should also be represented within Uppaal, because their values could be used in the DCs. These sensors are modelled as follows: at the initial state, the sensor is idle. The occurrence of the sensor activation event leads to the next state, and from this state, the deactivation event leads again to the initial state.

4. The concept of precedence relations amongst events

In distributed computation, subsystems don't share a common global memory, and communicate by exchanging messages through a communication network. Accordingly, events generated within a subsystem are modelled by three types of events: internal, send and receive events. Internal events are events that affect only the subsystem in which they occur and have no effect on the other neighbouring subsystems. Send and receive events refer to the stream of information among the subsystems. They establish causal dependency from the sender subsystem to the receiver one. The causal precedence relation induces a partial ordering of the events in a distributed computation. Within our distributed scheme, the precedence ordering among the events is relevant in modelling specifications of the AMS. To incorporate this novel concept to the approach, we use the relation of one event happens before another known as *Happens-before* relation proposed by Lamport (1978). In the following, we first recall the *Happens-before* relation. Second, we stipulate a set of novel rules of precedence ordering among events, which aid in simplifying the global specifications and allow to obtain simpler controller models. Third, we provide a set of graphical interpretations of these rules, which make it easier to convert the synthesised controllers into the syntax of Uppaal for the verification/validation, and the syntax of Grafcet for the implementation.

4.1 *Happens-before relation proposed by Lamport*

In 1978, Lamport proposed the partial ordering of events representation for reasoning about executions of distributed systems (Lamport 1978). This representation examines the concept of one event happens before another in a distributed system. The *happens-before* relation also known as causal relation, denoted by '$\rightarrow$', was defined as follows.

Definition: The relation '$\rightarrow$' on the set of events of a system is the smallest partial order relation satisfying the following three conditions:

(1) If a and b are events in the same process, and a comes before b, then $a \rightarrow b$.
(2) If a is the sending of a message by one process and b is the receipt of the same message by another process, then $a \rightarrow b$.
(3) If $a \rightarrow b$ and $b \rightarrow c$ then $a \rightarrow c$.

Two events a and b are concurrent if: $\neg (a \rightarrow b \vee b \rightarrow a)$.

The definition states that if an event a occurs before another event b, then a should happen at an earlier time than b, i.e. for any events a and b: if $a \rightarrow b$ then $C(a) < C(b)$, where C represents a system's clock which assigns to any event σ the number $C(\sigma)$.

4.2 *Events precedence ordering rules*

The *happens-before* relation allows us to introduce an order amongst the subsystems' events. Based on this relation, we provide a set of Boolean rules for modelling the dynamic behaviour of concurrent sub-systems of a global AMS. These rules greatly facilitate the interpretation of the control models used in the verification/validation and implementation steps.

The following conditions are satisfied for any events a, b, c, $d \in \mathrm{IB}$:

$a \to a = a$	Idempotent law	(1)
$a \to 1 = a$	Identity laws	(2)
$1 \to a = a$		(3)
$a \to 0 = 0$	Dominance laws	(4)
$0 \to a = 0$		(5)
$a \wedge (b \to c) = (a \wedge b) \to (a \wedge c)$	Distributive law	(6)
$a \to (a \wedge b) = (a \wedge b)$	Absorption laws	(7)
$(a \wedge b) \to a = (a \wedge b)$		(8)
$a \to (b \to c) = (a \to b) \to c$	Associative law	(9)

The following two operations on edges have been easily defined:

$$a \to \neg a = \downarrow a \tag{10}$$
$$\neg a \to a = \uparrow a \tag{11}$$

We mention that the commutative law is not allowed using events precedence relation operation. Nevertheless, the following formal statements have been defined to simplify the interpretation of some events ordering relations:

$$a \vee (b \to c) = (a \vee b) \to (a \vee c) \tag{12}$$
$$\begin{aligned} a \vee (a \to b) &= (a \vee a) \to (a \vee b) \\ &= a \to (a \vee b) \\ &= a \end{aligned} \tag{13}$$
$$\begin{aligned} a \vee (b \to a) &= (a \vee b) \to (a \vee a) \\ &= (a \vee b) \to a \end{aligned} \tag{14}$$
$$(a \to b) \wedge (a \to b) = (a \to b) \tag{15}$$
$$(a \to b) \wedge (c \to a) = c \to a \to b \tag{16}$$
$$(a \to b) \wedge (a \to c) = a \to (b \wedge c) \tag{17}$$
$$(a \to b) \wedge (c \to d) = [a \to (c \to ((d \to b) \vee (b \to d))) \vee (b \to c \to d)] \vee [c \to (a \to ((d \to b) \vee (b \to d))) \vee (d \to a \to b)] \tag{18}$$
$$(a \to b) \vee (a \to b) = (a \to b) \tag{19}$$
$$(a \to b) \vee (c \to a) = (a \to b) \vee (c \to a) \vee (c \to a \to b) \tag{20}$$
$$(a \to b) \vee (a \to c) = a \to (b \vee c) \tag{21}$$
$$(a \to b) \vee (c \to d) = [a \to (b \vee (c \to (b \vee d)))] \vee [c \to (d \vee (a \to (b \vee d)))] \tag{22}$$

The Boolean rules above give a generic structure of possible events precedence relations in a distributed AMS. They allow obtaining simpler models to be used for the verification/validation and Grafcet implementation steps.

For instance, the relation $a \wedge (b \to c)$ could be intuitively represented by the graphs of Figure 4(a) and (c) for verification/validation and for Grafcet implementation, respectively. Nevertheless, by using the distributive simplification rule

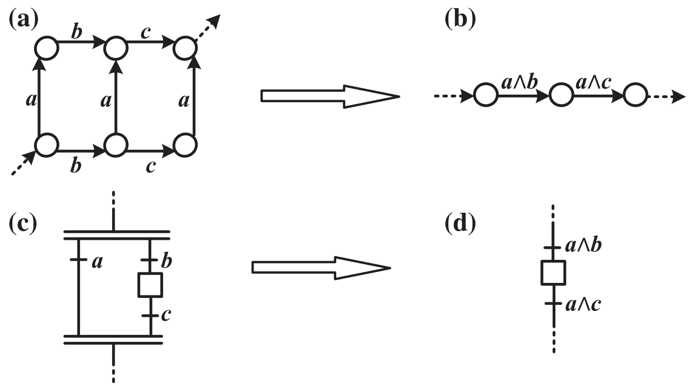


Figure 4. Example of a model simplification using precedence relation rule.

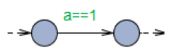
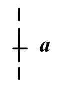
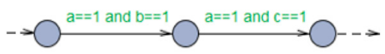
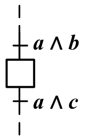
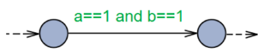
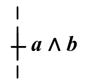
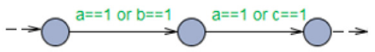
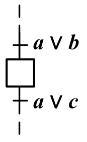
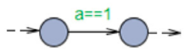
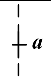
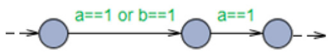
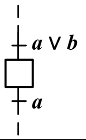
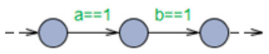
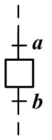
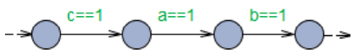
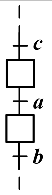
Id	Rule	Interpretation for verification purpose	Interpretation for control purpose
1	$a \to a$		
6	$a \wedge (b \to c)$		
7 & 8	$a \wedge (a \to b)$		
12	$a \vee (b \to c)$		
13	$a \vee (a \to b)$		
14	$a \vee (b \to a)$		
15 & 19	$(a \to b) \wedge (a \to b)$ $(a \to b) \vee (a \to b)$		
16	$(a \to b) \wedge (c \to a)$		

Figure 5. Graphical interpretations of the precedence relations for verification/validation and Grafcet implementation purposes.

(6), this relation could be reproduced by the graphs of Figure 4(b) and (d) for the same purposes. It is easy to perceive that the use of this rule results in less number of states and transitions which streamlines the computation and understanding of the models and also optimise the memory use for implementation.

Important event relations' rules above are graphically interpreted to simplify models for verification/validation and for Grafcet implementation purposes as shown in Figure 5.

5. Illustration: Application to an industrial AMS

5.1 *The industrial AMS*

An early application of the basic concepts of the approach to an industrial AMS, composed of three production stations, was carried out in (Qamsane, Tajer and Philippot 2016b). We focus here on the contribution and impact of the novel results of the approach, then, as an illustrative example, we shall apply these to only one station (the testing station). The global control of the other two stations could be constructed by applying the same principles.

The testing station (Figure 6) is controlled by a PLC Siemens S7-300. It comprises an elevator (double-acting rodless cylinder), an ejector (single-acting cylinder), and an air cushioned slide. The role of the elevator is to lift the

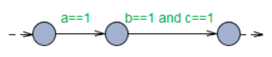
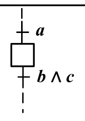
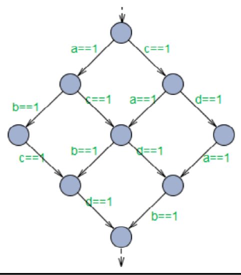
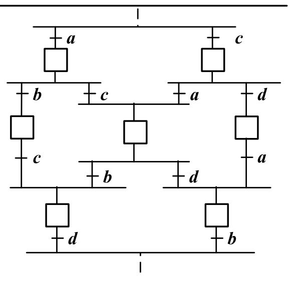
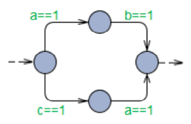
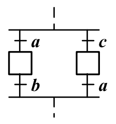
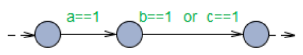
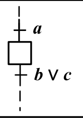
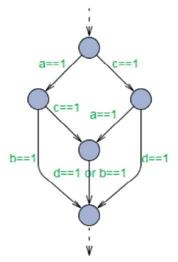
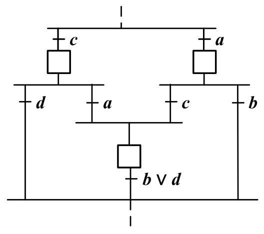
Id	Rule	Interpretation for verification purpose	Interpretation for control purpose
17	$(a \rightarrow b) \wedge (a \rightarrow c)$		
18	$(a \rightarrow b) \wedge (c \rightarrow d)$		
20	$(a \rightarrow b) \vee (c \rightarrow a)$		
21	$(a \rightarrow b) \vee (a \rightarrow c)$		
22	$(a \rightarrow b) \vee (c \rightarrow d)$		

Figure 5. *(Continued)*

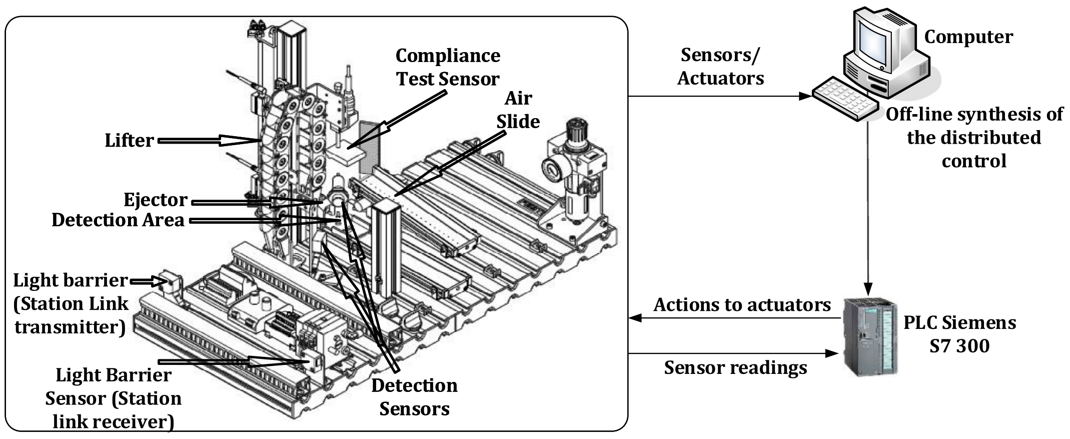


Figure 6. Application to the testing station.

workpieces towards a measuring sensor, which is located at the top of the station, in order to measure their height; the ejector drives the compliant workpieces towards a downstream station (sorting station) through the upper air cushioned slide, and ejects non-compliant ones to the lower slide at the bottom of the station. A set of sensors is used by the

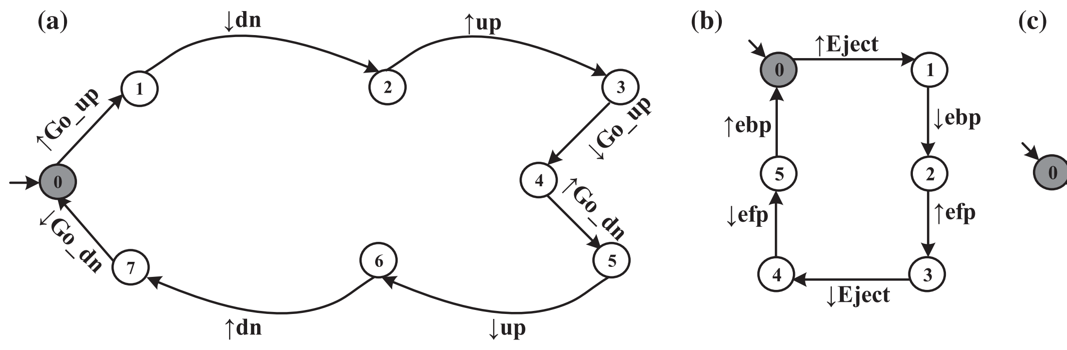


Figure 7. LCs. (a) Elevator, (b) Ejector and (c) Air cushioned slide.

station to detect the arrival of workpieces, their colour and material, working area freeness, and the station status (occupied/vacant).

5.2 *Automata models*

PE automata models are obtained by performing a parallel composition of a detector model, where an initial state is imposed, with an actuator model. A detailed procedure for the practical construction of PE models could be found in (Philippot 2006).

As mentioned before, we view an AMS as a nesting of PEs with immutable individual local operation. Correspondingly, immutable local constraints (defined by a system expert) are retained for each PE. The application of these constraints to the PE models allows obtaining immutable LCs. We don't give here the PE models and their local constraints for the studied AMS due to limited space; the reader can find examples in (Qamsane, Tajer and philippot 2016b). However, the LC automata models of the testing station are given in Figure 7.

5.3 *Definition of the global constraints*

5.3.1 *The informal global constraints*

Let us consider the following informal global specifications, which allow a coordinated operation among the considered PEs within the AMS:

(1) The elevator lifts up the workpieces only if the work area is not occupied;
(2) The elevator should not go up without a workpiece;
(3) Mutual exclusion between the elevator goes up and the ejector operation;
(4) If the tested workpiece is compliant then the elevator goes down after ejecting it towards the air cushioned slide by the ejector; otherwise it goes down to eject it in the down position.
(5) The elevator goes down only if the ejector is retreated;
(6) The ejector ejects the compliant workpieces when the elevator is in the up position, and ejects the non-compliant ones when the elevator reaches the down position.
(7) The air cushioned slide activates when the tested workpiece is compliant.
(8) The air cushioned slide deactivates when the tested workpiece is deposited to the downstream station.

5.3.2 *Formalisation of the global constraints*

The constraint (1) refers to that the elevator's order *Go_up* is authorised only when the swivel drive of the upstream station releases the lifting area. This information is captured by the sensor signal *free*. This constraint is formally described by the following Boolean expression:

(1) **If** *free* **then** *Ord*(*Go_up*)

The constraint (2) states that the authorisation of order *Go_up* is also conditioned by workpiece availability at the entrance of the station, which is captured by the sensor signal *wpa2*. This constraint is formally described by:

(2) **If** *wpa2* **then** *Ord*(*Go_up*)

The constraint (3) states that the elevator goes up only if the ejector is not operating. It can be formally described by:

(3) **If** $\neg$*Eject* **then** *Ord*(*Go_up*)

A single global constraint is constructed for the order *Go_up* by the conjunction of the three constraint's conditions into one condition. This constraint is given by:

(i) **If** ($free \wedge wpa2 \wedge \neg Eject$) **then** *Ord*(*Go_up*)

The order *Go_dn* of the elevator is authorised if the measuring sensor located at the up position of the elevator returns a negative result (captured by $\neg ok$), and the ejector is not operating (captured by $\neg Eject$); otherwise, if the measuring sensor returns a positive result, (captured by *ok*), then the workpiece is ejected to the air cushioned slide. As there is no sensor at the entrance of the air cushioned slide, this information is captured when the ejector returns to its back position after pushing the workpiece. The global constraint authorising the order *Go_dn* is then given by:

(ii) **If** $(\neg ok \wedge \neg Eject) \vee ((ok \wedge Eject) \rightarrow \uparrow ebp)$ **then** *Ord*(*Go_dn*)

Similar formal interpretations of the informal constraints above allow obtaining the global constraints for the PEs as shown in Table 1.

5.4 *Synthesis of the distributed controllers*

In order to synthesise the DCs for the studied AMS, the global constraints of Table 1 are considered to the LCs of Figure 7 according to the synthesis algorithm of (Qamsane, Tajer, and Philippot 2016a, 2016b). A brief illustration of the synthesis steps on the elevator subsystem is shown in Figure 8. The DCs of the suction cup and the ejector shown in Figure 9 are obtained in a similar way.

5.5 *Verification/validation and Grafcet implementation of the DCs*

5.5.1 *Verification/validation*

In order to check the properties of the synthesised distributed control, the obtained DC automata are translated into Uppaal syntax. The translation of the DC automata models into Uppaal is based on the use of the method described in Section 3.2. Additionally, the conditions (within the macro-states) which capture precedence relations amongst events are simplified by using the interpretations of events precedence rules shown in Figure 5.

For instance, the elevator's DC realises the desired operation in the following sense: initially, the elevator is in the down position, which is captured by the sensor '$dn = 1$'. Whenever a workpiece is available at the entrance of the testing station (captured by '$wpa2 = 1$'), the working area is free (captured by '$free = 1$'), and the ejector is not operating (captured by '$Eject = 0$'), the elevator lifts up the workpiece towards the measuring module by the activation of the order '$Go\_up = 1$'. The activation of the latter makes the elevator leave the down position (captured by '$dn = 0$') and reach the up position (captured by '$up = 1$'). Once in the up position, the lifting order is deactivated (captured by '$Go\_up = 0$'). Provided that the condition $(\neg ok \wedge \neg Eject) \vee ((ok \wedge Eject) \rightarrow \uparrow ebp)$ is fulfilled, the activation of the order '$Go\_dn = 1$' is allowed and enables the elevator to go down by leaving the up position ('$up = 0$') and reaching the down position ('$dn = 1$').

Table 1. Formal global constraints of the studied system PEs.

Station	Subsystem	Global constraint
Testing	Elevator	(i) **If** $(free \wedge \neg Eject \wedge wpa2)$ **then** *Ord*(*Go_up*)
		(ii) **If** $(\neg ok \wedge \neg Eject) \vee ((ok \wedge Eject) \rightarrow \uparrow ebp)$ **then** *Ord*(*Go_dn*)
	Ejector	(iii) **If** $(\neg Go\_up \wedge \neg Go\_dn) \wedge ((up \wedge ok) \vee ((up \wedge \neg ok) \rightarrow \uparrow dn))$ **then** *Ord*(*Eject*)
	Air cushioned slide	(iv) **If** $(ok \wedge ssv)$ **then** *Ord*(*Prompt*)
		(v) **If** *wpa3* **then** *Inh*(*Prompt*)

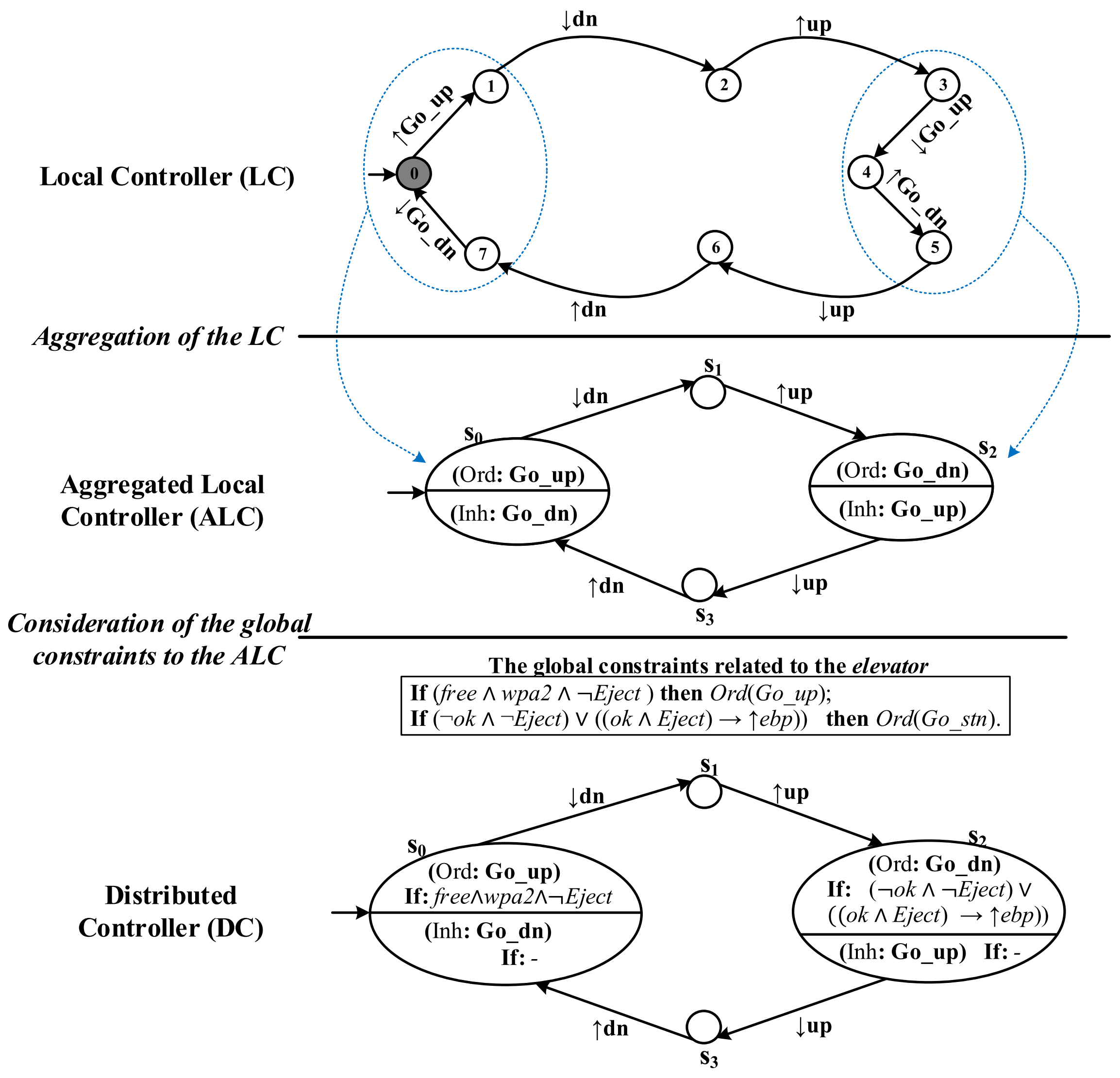

Local Controller (LC)
↓dn
↑up
↑Go_up
↓Go_up
↓Go_dn
↑Go_dn
↑dn
↓up
0
1
2
3
4
5
6
7
Aggregation of the LC
Aggregated Local Controller (ALC)
s0
s1
s2
s3
(Ord: Go_up)
(Inh: Go_dn)
(Ord: Go_dn)
(Inh: Go_up)
Consideration of the global constraints to the ALC
The global constraints related to the elevator
If (free ∧ wpa2 ∧ ¬Eject) then Ord(Go_up);
If (¬ok ∧ ¬Eject) ∨ ((ok ∧ Eject) → ↑ebp)) then Ord(Go_stn).
Distributed Controller (DC)
(Ord: Go_up) If: free∧wpa2∧¬Eject
(Inh: Go_dn) If: -
(Ord: Go_dn) If: (¬ok ∧ ¬Eject) ∨ ((ok ∧ Eject) → ↑ebp))
(Inh: Go_up) If: -

Figure 8. Illustration of the DC synthesis for the elevator subsystem.

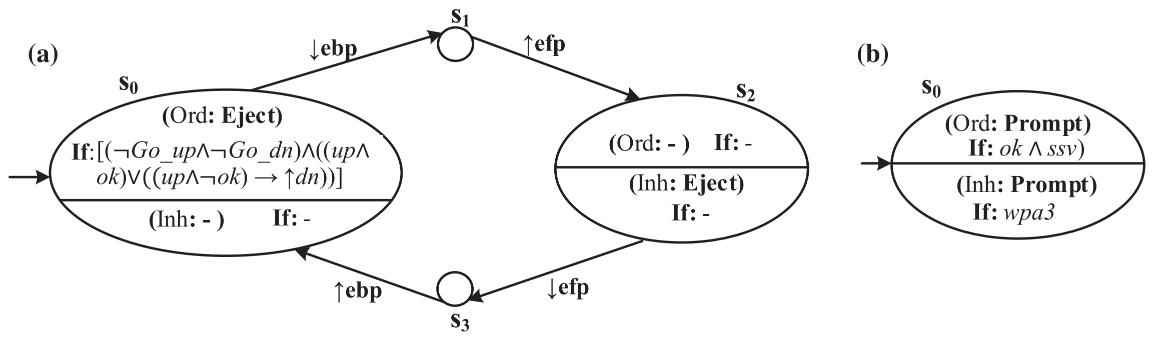


Figure 9. DCs automata models for: (a) Elevator and (b) Air cushioned slide.

The condition $(\neg ok \wedge \neg Eject) \vee ((ok \wedge Eject) \rightarrow \uparrow ebp))$ could be simplified using the basic Boolean operations and the precedence relation rules provided in Section 4.2. Manipulating this formal statement allows to obtain its equivalent as follows:

$$\begin{aligned}(\neg ok \wedge \neg Eject) \vee ((ok \wedge Eject) \rightarrow \uparrow ebp) &= (\neg ok \wedge \neg Eject) \vee (ok \wedge Eject) \rightarrow (\neg ok \wedge \neg Eject) \vee \uparrow ebp \\ &= \neg(ok \oplus Eject) \rightarrow (\neg ok \wedge \neg Eject) \vee \uparrow ebp\end{aligned}$$

Figure 10 shows the automaton describing the elevator DC behaviour within Uppaal.

The remaining models are reproduced into Uppaal syntax as shown in Figure 11.

After reproducing all the system automata into Uppaal, we first verify the absence of deadlock using the property **A [] not deadlock**. Uppaal verifier returns that the property is fulfilled. Second, we verify the system's safety and liveness requirements. For example, the informal global constraint (3) captures a mutual exclusion between the elevator's *Go_up* action and the ejector operation. It can be verified within Uppaal by the property: **E<> not (Elevator.advancing1) and (Ejector.advancing1)**. Uppaal verifier returns that the control specifications are fulfilled. Moreover, by using Uppaal simulator, it is validated that the controller realises the desired production cycle, i.e. the controller is designed correctly with respect to the requirements.

5.5.2 *Grafcet implementation*

An interpretation methodology that aims at transforming the DCs into Grafcet formalism was defined in (Qamsane, Tajer, and Philippot 2016b). This section shows how the event precedence relations proposed in this paper could affect the Grafcet interpretation of the DCs.

For instance, by applying the interpretation methodology to the elevator's DC, we obtain the Grafcet model shown in Figure 12(a). Within this model, the receptivity associated with the transition *t*3 captures a precedence relation, which is a form not defined in the Grafcet standard (IEC Standard 60848 2013). An intuitive interpretation of this form into Grafcet formalism is shown in Figure 12(b). This representation captures a parallel structure. However, by simplifying this form using the precedence relations presented in Section 4, we obtain the Grafcet model shown in Figure 12(c), which has simpler linear structure.

As a result, we mention that the use of the event precedence relations allows to obtain fewer step numbers, which results in an easy understanding of the models and less memory usage for implementation.

Figure 13 shows the partial grafcets corresponding to the remaining DCs of the testing station, which are obtained in a similar way. The three obtained grafcets were tested on the system, and the observed behaviour doesn't violate the desired one.

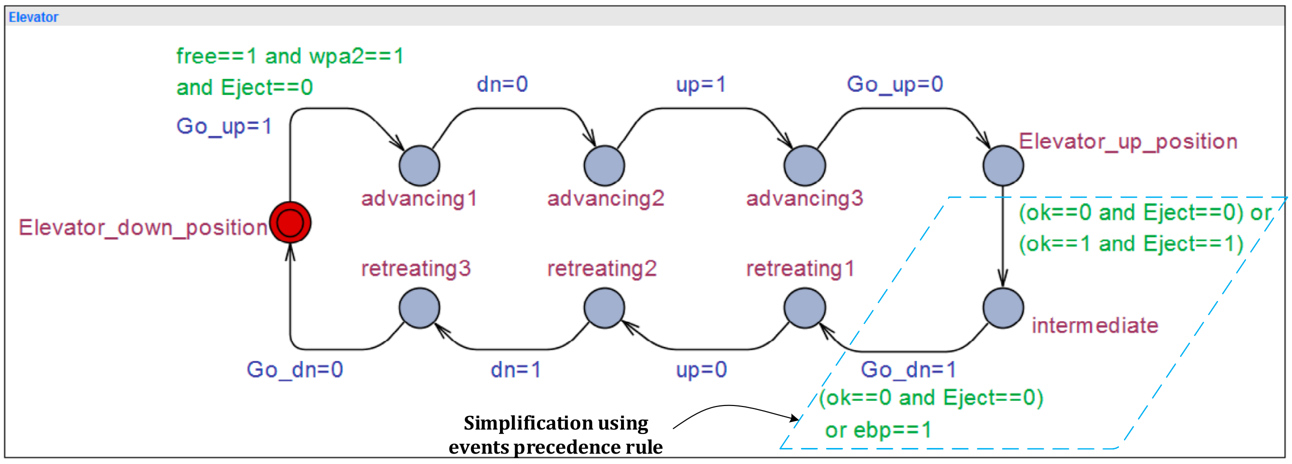


Figure 10. Reproduction of the elevator DC into Uppaal.

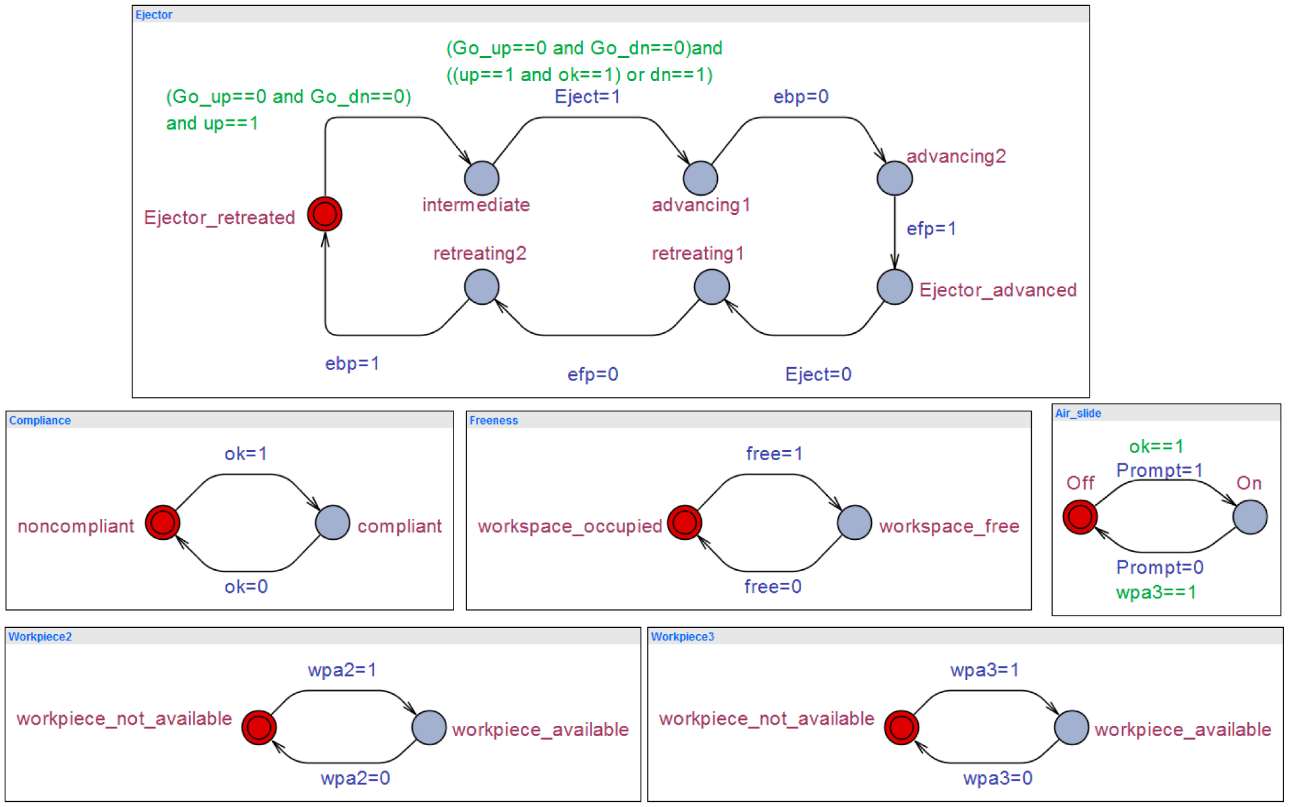


Figure 11. Representation of the DCs and sensor models within Uppaal.

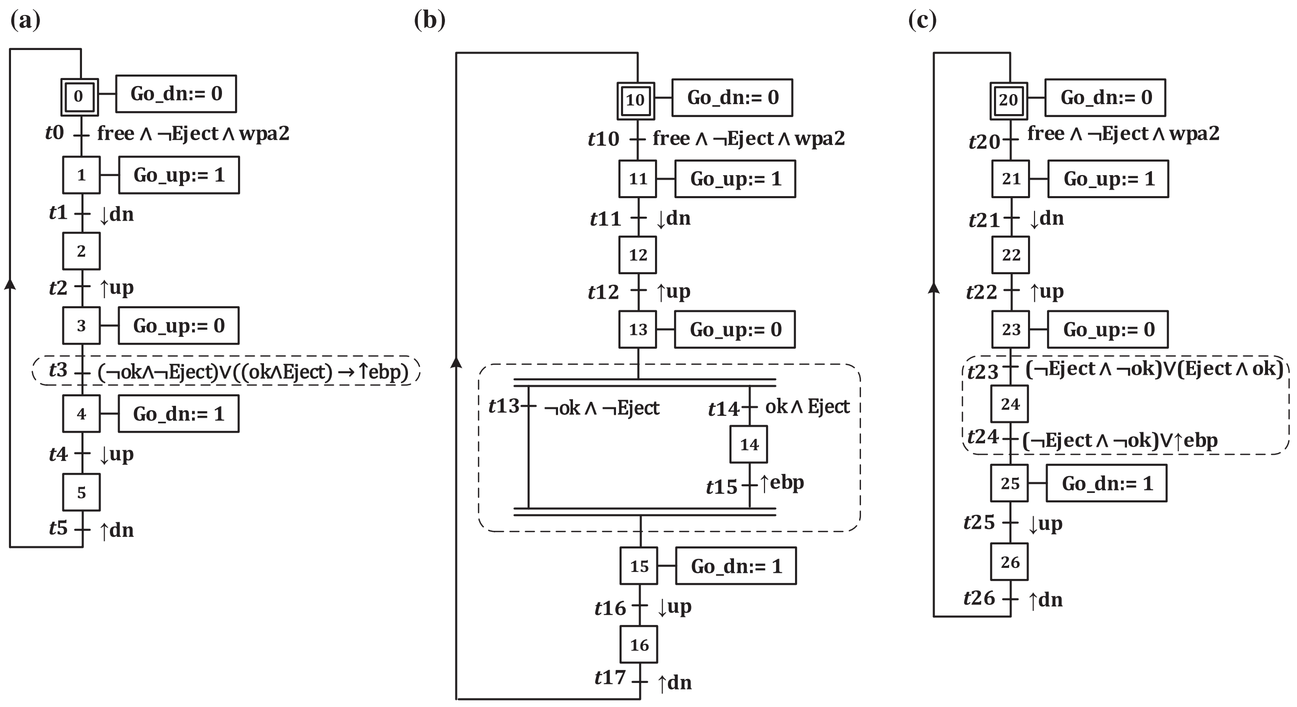


Figure 12. Reduction of the elevator's Grafcet using events precedence relation rules.

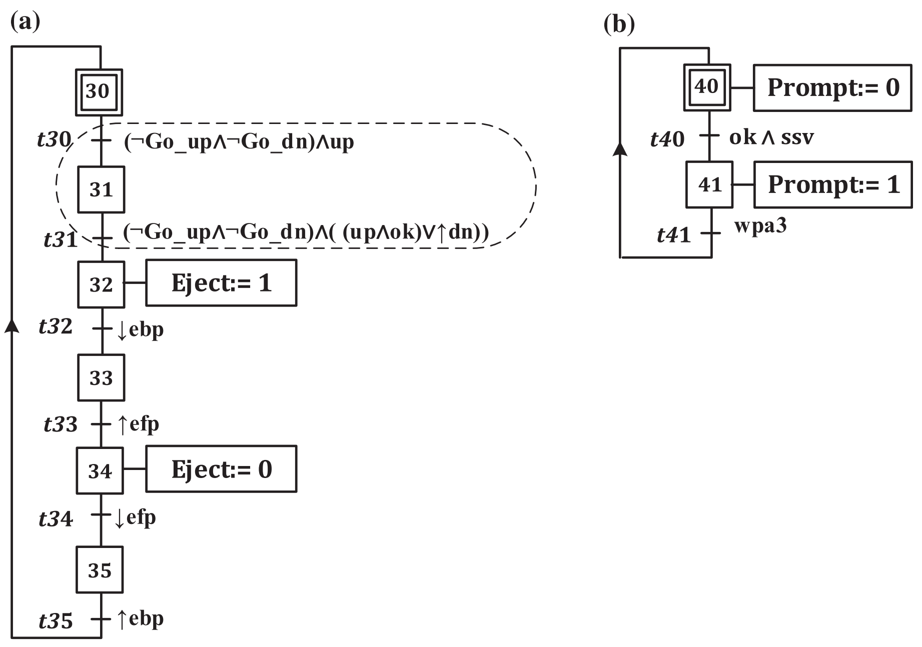


Figure 13. Grafcet models for: (a) Ejector, (b) Air cushioned slide.

6. Discussion

In basic SCT, it is assumed that the plant model spontaneously ‘generates’ events unless banned from doing so. The control mechanism devoted to the supervisor is the ability to enable the occurrence of maximum allowable controllable events at any instant, based on the observation of a sequence of events. Formally, a supervisor, *S*, is a map $S : L(G) \rightarrow 2^{\Sigma}$ such that $\Sigma_{\text{uc}} \subseteq S(\omega)$ for any $\omega \in L(G)$, where $S(\omega)$ represents the set of events that are enabled to occur according to the observation of the string ω. However, for real system applications, processes typically react to commands as inputs and responses as outputs, i.e. system actuators are triggered by commands, while responses return sensor state changes. Then, the plant doesn’t spontaneously generate controllable events, but executes them when commanded by a controller. (Golaszewski and Ramadge 1987) introduce for instance the concept of forced events, where the set of events is partitioned into controllable, uncontrollable and enforceable events. The latter occur if and only if they are enforced by the controller. In this way, the enforcement concept allows to preempt the controllable and uncontrollable events, thus these can be avoided. Ultimately, in (Balemi et al. 1993), it is claimed that to ensure the completeness of the process with regard to the input–output supervisor, a controller that actively enforces control actions is obtained from the synchronous product of the supervisor and the process models. A similar interpretation was introduced in (Brandin 1996) for timed DES. Its principle is that forcible events are able to preempt the tick event of a global clock. Based on these results, several other SCT forcing event approaches have been introduced in the literature. (Cantarelli and Roussel 2008; Huang and Kumar 2008) propose to implement a direct controller that selects at most one controllable event to be enabled at any instant. (Gelen and Uzam 2014; Baldissera, Cury, and Raisch 2016) have developed approaches within the context of forcing events, and applied them to real-world processes. The former proposes a hybrid modular method coupling automata supervisors to processes modelled as Petri nets. Its applicability is demonstrated by the PLC-based real-time control of an experimental manufacturing system; and the latter introduces the state attraction problem associated with the concepts of forcible events and weak preemption. The proposed methodology allows to synthesise controllers for biological processes.

The main difference between the control synthesis method presented in this work and basic SCT lies in the interpretation of the role of the controller. In basic SCT, the maximally permissive supervisor is a passive device that tracks all events produced by the plant and restricts the behaviour of the plant by disabling the controllable events. Our method is mainly based on Balemi’s interpretation (Balemi et al. 1993), where the controllable events $\Sigma_c \subseteq \Sigma$ represent the control outputs (actuators) and the uncontrollable events $\Sigma_{uc} \subseteq \Sigma$ represent the control inputs (sensors). It also assumes that the rising ‘↑’ and falling ‘↓’ edges associated with events are the changes of their values from 0 to 1 and from 1 to 0, respectively. In contrast to basic SCT, the event generation within our approach is initiated not only by the plant, but by both the plant and the controller. Commands are produced by the controller and responses by the plant. Similar to the forcing event approaches above, the DCs developed within our method also directly enforce a control input to the plant at any instant to fulfil the specification.

According to (Balemi et al. 1993), to ensure the completeness of the process with regard to the input–output controller, it is essential to compute the synchronous product of the controller with the plant. This method leads to a large number of states and transitions in the obtained controller. However, because our approach uses logical Boolean expressions for the specifications modelling instead of automata, it turns out to avoid the computational complexity. The reader can find a comparison between the total number of states in the plant, and the controllers for an AMS example, when using the centralised and our proposed distributed approach in (Qamsane, Tajer, and Philippot 2016b).

Additionally, because most industrial systems make use of PLC for various automation tasks, we believe additional investigations on PLC implementation are of interest. Typically, the SCT is supported by automata and formal languages theory (Hopcroft, Motwani, and Ullman 2006), which are based on an asynchronous hypothesis. However, a PLC can involve simultaneous changes of the input or output vectors. This synchronous behaviour is not easy to take into account in SCT approaches. To raise this issue, some breakthrough SCT extensions have been introduced in the literature to adapt the SCT to the PLC programming paradigm (see, e.g. Hellgren, Fabian, and Lennartson 2001; Basile, Chiacchio, and Gerbasio 2013). These works propose several techniques to design controllers which are implementable in PLC. Nevertheless, the use of these techniques in the real-world industry is not so evident, because real control applications involve large-scale system models, which lead to combinatorial explosion. Other work extensions propose novel patterns to deal with the PLC programming issue. The work presented in (Roussel and Lesage 2014) uses a formal framework based on Boolean functions for the design of dependable logic control systems. In this work, the modelling, algorithms and results are only based on Boolean equations. However the functional and safety parts are not separated, which leads to carry an entire update of the controller in case of a change of specifications. In (Riera et al. 2014), a safety filter approach based on Boolean equations is proposed. This method separates the safety part independently from the functional part. Moreover, the developed formalism and algorithm allow to take the synchronous occurrences of events into account.

The future work should extend the current result to find different techniques for moving from the synchronous event-based DES-world, to the asynchronous, signal-based PLC-world.

7. Conclusion

In this paper, we investigated the synthesis, verification and validation of distributed control for AMS. The synthesis of the distributed control is carried out under consideration of enforceable events in addition to basic SCT. The specifications are given as a set of Boolean rules that must be satisfied by the plant subsystems. This is the key to avoid the computational complexity, which is due to the parallel composition of automata models. This work also made an expansion of formal Boolean manipulations by proposing the concept of precedence relations amongst events, which enhances this design method. Additionally, an analysis procedure to verify the non-blockingness and optimality of the designed control is developed. It makes use of Uppaal tool box for the verification via automatic model-checking and validation via graphical simulation.

All steps of the proposed control strategy have been successfully applied to a real AMS low-level control, where the role of the DCs is to organise low-level interaction among the control devices, such as cylinders, motors, etc. Because the controller obtained in this study is a distributed type, the state-space would get significantly reduced, when this method is used for large-scale systems. This is due to the modularity, which provides small and compact structure of the controller. Therefore, for PLC-based implementation purposes, the utilisation of this approach results in less memory usage.

The main weak point of the proposed approach is the necessity of using formal manipulations that are not familiar to the control practitioners. To overcome this difficulty, in our current work (Qamsane et al. 2017), we are developing a tool support for the method. This tool will embed a library of PEs automata, local Boolean constraints for constructing the LCs, and formal operations/relations to easily specify the global constraints. The tool will allow industrial acceptance of the proposed control synthesis method. Further prospect shall also concern the development of an automatic method to refine global constraints in case of negative verification results. Another avenue of research is the introduction of a modal temporal logic framework to the approach presented in this paper. The two major objectives of such framework are (a) to consider the issue of moving from the synchronous event-based DES-world, to the asynchronous, signal-based PLC-world, and (b) to treat quantitative control problems in order to evaluate the system performance.

Disclosure statement

No potential conflict of interest was reported by the authors.

ORCID

Yassine Qamsane http://orcid.org/0000-0003-4036-2586

References

Baier, C., and J. P. Katoen. 2008. *Principles of Model Checking*. Boston, MA: MIT Press.

Baldissera, F. L., J. E. R. Cury, and J. Raisch. 2016. “A Supervisory Control Theory Approach to Control Gene Regulatory Networks.” *IEEE Transactions on Automatic Control* 61 (1): 18–33.

Balemi, S., G. J. Hoffmann, P. Gyugyi, H. Wong-Toi, and G. F. Franklin. 1993. “Supervisory Control of a Rapid Thermal Multiprocessor.” *IEEE Transactions on Automatic Control* 38 (7): 1040–1059.

Basile, F., P. Chiacchio, and D. Gerbasio. 2013. “On the Implementation of Industrial Automation Systems based on PLC.” *IEEE Transactions on Automation Science and Engineering* 10 (4): 990–1003.

Behrmann, G., A. David, and K. G. Larsen. 2006. “A Tutorial on Uppaal.” In *Formal Methods for the Design of Real-time Systems*, 200–236. Berlin: Springer-Verlag.

Brandin, B. A. 1996. “The Real-time Supervisory Control of an Experimental Manufacturing Cell.” *IEEE Transactions on Robotics and Automation* 12 (1): 1–14.

Cantarelli, M., and J. M. Roussel. 2008. “Reactive Control System Design using the Supervisory Control Theory: Evaluation of Possibilities and Limits.” In *9th International Workshop On Discrete Event Systems*, 200–205. Goteborg.

Diogo, R. A., E. A. P. Santos, A. D. Vieira, E. D. F. Vieira, and M. A. Busetti. 2012. “A Computational Control Implementation Environment for Automated Manufacturing Systems.” *International Journal of Production Research* 50 (22): 6272–6287.

Gelen, G., and M. Uzam. 2014. “The Synthesis and PLC Implementation of Hybrid Modular Supervisors for Real Time Control of an Experimental Manufacturing System.” *Journal of Manufacturing Systems* 33: 313–326.

Golaszewski, C. H., and P. J. Ramadge. 1987. “Control of Discrete-event Processes with Forced Events.” In *Proceedings of the 28th Conference on Decision and Control*, 247–251. Los Angeles.

Hellgren, A., M. Fabian, and B. Lennartson. 2001. “Modular Implementation of Discrete Event Systems as Sequential Function Charts Applied to an Assembly Cell.” In *Proceedings of the 2001 IEEE Conference on Control Applications (CCA ‘01)*, 453−258. Mexico City.

Hopcroft, J., R. Motwani, and J. Ullman. 2006. *Introduction to Automata Theory, Languages, and Computation*. 3rd ed. Boston, MA: Addison-Wesley Longman .

Hu, H., R. Su, M. C. Zhou, and Y. Liu. 2015. “Polynomially Complex Synthesis of Distributed Supervisors for Large-scale AMSs Using Petri Nets.” *IEEE Transactions on Control Systems Technology* 24 (5): 1–13.

Huang, J., and R. Kumar. 2008. “Directed Control of Discrete Event Systems for Safety and Nonblocking.” *IEEE Transactions on Automation Science and Engineering* 5 (4): 620–29.

IEC Standard 60848. 2013. “Grafcet Specification Language for Sequential Function Charts.” *International Electrotechnical Commission*. Geneva.

Lamport, L. 1978. “Time, Clocks, and The Ordering of Events in a Distributed System.” *Communications of the ACM* 21 (7): 558–565.

Lano, K. 2012. *The B Language and Method: A Guide to Practical Formal Development*. London: Springer-Verlag London.

Philippot, A. 2006. “Contribution Au Diagnostic Décentralisé Des Systèmes À Événements Discrets: Application Aux Systèmes Manufacturiers.” PhD thesis, University of Reims-Champagne Ardenne (in French).

Qamsane, Y., A. Tajer, and A. Philippot. 2016a. “Distributed Supervisory Control Synthesis For Discrete Manufacturing Systems.” *IFAC-PapersOnLine* 49(12), 396–401.

Qamsane, Y., A. Tajer, and A. Philippot. 2016b. “A Synthesis Approach to Distributed Supervisory Control Design for Manufacturing Systems with Grafcet Implementation.” *International Journal of Production Research* 55 (15): 4283–4303.

Qamsane, Y., M. Elhamlaoui, A. Tajer, and A. Philippot. 2017. “A Tool Support to Distributed Control Synthesis and Grafcet Implementation for Discrete Event Manufacturing Systems.” In *The 20th World Congress of the International Federation of Automatic Control.* Toulouse.

Ramadge, P. J., and W. M. Wonham. 1987. “Supervisory Control of a Class of Discrete Event Processes.” *SIAM Journal on Control and Optimization* 25 (1): 206–230.

Riera, B., R. Coupat, A. Philippot, D. Annebique, and F. Gellot. 2014. “Control Design Pattern Based on Safety Logical Constraints for Manufacturing Systems: Application to a Palletizer.” *IFAC Proceedings Volumes* 47 (2): 388–393.

Roussel, J. M., and J. J. Lesage. 2014. “Design of Logic Controllers Thanks to Symbolic Computation of Simultaneously Asserted Boolean Equations.” *Mathematical Problems in Engineering* 2014: 1–15.

Roussel, J. M., J. M. Faure, J. J. Lesage, and A. Medina. 2004. “Algebraic Approach for Dependable Logic Control Systems Design.” *International Journal of Production Research* 42 (14): 2859–2876.

Wonham, W. M. 2012. *Supervisory Control of Discrete-event Systems*. Systems Control Group, Department of Electrical and Computer Engineering, University of Toronto. http://www.control.toronto.edu/DES.