Synthesis of control implementation for discrete manufacturing systems

J. Zaytoon, Véronique Carré-Ménétrier

► To cite this version:

J. Zaytoon, Véronique Carré-Ménétrier. Synthesis of control implementation for discrete manufacturing systems. International Journal of Production Research, 2001, 39 (2), pp.329-345. 10.1080/00207540010002388 . hal-02111218

HAL Id: hal-02111218

<https://hal.science/hal-02111218>

Submitted on 28 Oct 2019

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Synthesis of control implementation for discrete manufacturing systems

J. Zaytoon & V. Carre-Meneatrier

To cite this article: J. Zaytoon & V. Carre-Meneatrier (2001) Synthesis of control implementation for discrete manufacturing systems, International Journal of Production Research, 39:2, 329-345, DOI: [10.1080/00207540010002388](https://doi.org/10.1080/00207540010002388)

To link to this article: <http://dx.doi.org/10.1080/00207540010002388>

Published online: 14 Nov 2010.

Submit your article to this journal [↗](#)

Article views: 21

View related articles [↗](#)

Citing articles: 10 View citing articles [↗](#)

Synthesis of control implementation for discrete manufacturing systems

J. ZAYTOON†* and V. CARRÉ-MÉNÉTRIÉR†

The paper presents the concepts and steps required to synthesize a correct control implementation for discrete manufacturing systems, starting from Grafcet specifications. A formal framework implementing the synthesis steps is also presented and illustrated with an example of a drilling system.

1. Introduction

1.1. Background

In the past, discrete manufacturing systems have been sufficiently simple so that intuitive or *ad hoc* control solutions have been adequate. However, the increasing complexity of these systems and the necessity of meeting the more demanding quality and dependability constraints have created a need for formal approaches for the synthesis of efficient and correct control implementation. The supervisory control theory has therefore been introduced (Ramadge and Wonham 1989) to provide algorithms for the synthesis of supervisory controllers from their specifications. Despite its theoretical appeal, there are very few control logic synthesis applications based on this theory. This is due to the fact that the proposed logical model assumes a plant that generates events spontaneously and the only control mechanism available to the supervisor is the ability to prevent the occurrence of some events called controllable events. In contrast, real time systems usually react to commands as inputs with responses as outputs.

Despite the many extensions of the supervisory-control theory and intensive research efforts on the theoretical modelling, only a few applications of supervisory controllers were reported (Brandin *et al.* 1991, Arinez *et al.* 1993, Balemi *et al.* 1993, Lauzon *et al.* 1997). In Balemi *et al.* (1993), controlled automata are employed and a computer in which the supervisory-control strategies resided was directly wired to the devices to be controlled. The control strategy in Brandin *et al.* (1991) and in Arinez *et al.* (1993) was developed based on controlled-automata manually translated into a ladder logic code, and subsequently programmed manually into the PLC. In Lauzon *et al.* (1997), an implementation method that utilizes results of the supervisory control theory in conjunction with PLC technology is used for on-line generation of limited-size control strategies. A host PC generates the on-line control strategies and downloads them to a PLC that supervises a manufacturing workcell, reacting to events and enforcing device behaviour based on the current strategy. The controlled automate used in these approaches provide a general framework for establishing fundamental properties of discrete-event controllers problems. However, they do not represent convenient or intuitive models for practical systems

Revision received July 2000.

†LAM, Faculté des Sciences, B.P. 1039, F-51687 Reims Cedex 2, France.

*To whom correspondence should be addressed. e-mail: janan.zaytoon @univ-reims.fr

because of the large number of states they have to introduce to represent several interacting subsystems, and because of the lack of structure.

These limitations have motivated the use of Petri nets to provide compact descriptions in the context of supervisory control (Giua 1996), because the structure of the net may be maintained small in size even if the marking grows. Furthermore, Petri nets can be efficiently used in modular synthesis. Some of the existing synthesis techniques (Holloway and Krogh 1990, Li and Wonham 1993) involve the design of maximally permissible control policy for enforcing linear constraints on the reachability set of the Petri net. In Holloway and Krogh (1990), control places and a feedback law are introduced to control the marking of safe marked graphs. This work has received a lot of attention in the literature and has also been extended to classes of Petri nets other than marked graphs. Li and Wonham (1993) used incidence matrix analysis to compute the control law that enforces linear constraints for a class of Petri nets. This approach is likely to be unfeasible in practical applications because the controller has to solve several integer-programming problems on-line. The authors have therefore studied other classes of nets for which the controller may be represented as a Petri net. Other approaches, such as the one proposed in Yamalidou *et al.* (1996), give up the requirement that the control policy be maximally permissible and propose a more restrictive policy provided it can easily be computed. On the implementation level, Petri nets are employed by Zhou and Dicesare (1993) to obtain supervisory control strategy resided on a computer and directly wired to the devices to be controlled. Other approaches are based on the transformation of some variants of Petri nets into ladder logic programs, which can be run on a PLC (Jafari and Boucher 1994, Zhou and Twiss 1998).

Most of these existing Petri net-based approaches are concerned with the supervisory control level, which refer to maintaining overall control of a number of machines or subsystems including PLC to prevent conflicts and enable cooperative or shared tasks. They do not address the control of the operating cycle of individual machines. This paper addresses this local control level by using Grafcet, which is derived from Petri nets and widely spread in industrial applications for the specification of PLC controllers.

1.2. Overview of synthesis steps

This paper provides a gradual and informal presentation of the concepts and steps that should be considered to synthesize a correct control implementation for discrete manufacturing systems, starting from high-level control specifications given in Grafcet. The term 'correct' refers here to a deterministic, reactive and deadlock-free control implementation, which also satisfies a number of safety requirements. A reactive system is characterized by being event-driven, continuously having to react to external stimuli (Harel 1987). Determinism implies that for each input scenario, the system produces one and only one output scenario.

The SADT diagram of figure 1 presents the steps required to move from a given Grafcet specification to the corresponding controller implementation. The first step consists of extracting an automaton of stable situations (ASS) that represents a deterministic and reactive execution motor for Grafcet. In the second step, the largest permissible behaviour of this execution motor with respect to a number of given safety constraints is extracted. The resulting 'safe, deterministic and reactive controller' may exhibit a number of deadlocks in real execution, when it is connected to the plant. To identify and treat these deadlocks, the third step relies on an abstract

Figure 1. Basic steps given in terms of SADT.

modelling of the behaviour of the plant. The analysis of the identified deadlocks allows the designer to correct the Grafcet, relax the safety constraints, or refine the abstract model of the plant. Then, the fourth step consists in implementing this correct controller by taking into account the constraints relative to the target control-execution architecture.

The paper is organized as follows. After a recall on the Grafcet model in Section 2, Sections 3-6 are each dedicated to one of the four mentioned steps. A simple example is used to provide a didactic illustration of the concepts presented in these sections. An integrated framework implementing these steps will be presented in Section 7 together with an example of a drilling system in Section 8.

2. Grafcet

Grafcet or sequential function charts is an international standard used for the specification of logic controllers in manufacturing systems (IEC 1991, David and Alla 1992). This model consists in describing parallel and synchronized sequences of elementary operations (figure 2) applied to the plant with due consideration to plant's response. The basic concepts of Grafcet are clear and simple. The step, drawn as a square, represents a partial state of the controller to which actions can be associated. A step can be active or idle; associated actions are performed when the step is active and remain asleep when it is idle. A situation is given by the set of active steps. The transition, represented as a bar, links one (or several) previous step(s) to one (or several) following step(s). A logical expression, called receptivity, is associated to each transition. This expression manipulates Boolean variables, representing controller inputs or the activation state of individual steps, and events corresponding to rising edges ($\uparrow$) or falling edges ($\downarrow$) of Boolean inputs.

The use of Grafcet as a control model in the frame of the supervisory control theory has been proposed in Charbonnier *et al.* (1999). Despite its successful application to an industrial system, this approach is limited to a subclass of Grafcet in which only impulse actions and single-event receptivities can be used. The proposed

Figure 2. Different configurations of Grafcet.

formal analysis is based on the restrictive hypothesis that a Grafset action and its downstream plant reaction can be considered as an atomic event, whereas the plant in a real system may induce different interleaving sequences between control actions and their consequent reactions. Another approach has therefore been proposed by the authors (Zaytoon *et al.* 1999) to take into account all the features of Grafset and to obtain a correct control implementation. This approach, which is briefly reviewed in Section 7, is based on the concepts and the steps presented in Sections 3–6.

3. Generation of the automaton of stable situations of Grafset

To achieve both reactivity and determinism, it is desirable to generate an execution motor (an automaton) for Grafset that evolves instantaneously from one stable situation to another, upon the occurrence of input events. A stable situation is a state in which the Grafset cannot evolve without acquiring a new input. To ensure that deterministic semantics are conferred to Grafset, actions are only performed during stable situations; actions related to the intermediate unstable situations remain asleep (Frachet and Colombari 1993, UTE 1993, Lhoste *et al.* 1997).

The generation of the automaton of stable situations (ASS) of Grafset (Roussel 1994, Zaytoon and Carré-Ménétrier 1999) are illustrated using the simple Grafset shown in figure 3a. Despite its simplicity, this example illustrates many of the convenient features of Grafset, including parallel sequences, simultaneous transitions, search for stability, logical expressions involving input variables, events, and/or Boolean-step variables.

Each state of the ASS (figure 3b) represents a stable situation of Grafset, and it is therefore defined in terms of a set of simultaneously active steps, as well as the actions to be performed during the situation. A transition of ASS, which represents an evolution from one stable situation of Grafset to another, is labelled with an event that can be associated with a logical expression. Each transition is identified with the corresponding transition(s) of Grafset. In the following text, the terms ‘state’ (of ASS) and ‘situation’ (of Grafset) will be used interchangeably.

The initial state of ASS corresponds to the initial situation of Grafset, in which only step 0 is active and action ACT6 is performed. In this state, transition 1 is taken when $\uparrow f$ occurs to activate the situation $\{1\}$, which performs the actions ACT1, ACT2, ACT3 and ACT4. Then, when $\uparrow a$ occurs, and provided that d is *true*, transition 2 activates steps 10 and 20. In this situation, where no actions are performed, only transition 3 can occur (because transition 6 is conditioned with the activation of step 11) to activate $\{11, 20\}$. The next evolution depends on the values of the inputs a , b and e . If a is *true* when $\uparrow b$ occurs, transition 6 activates the situation $\{11, 21\}$. On the other hand, if b becomes *true* before a , the next evolution will be conditioned by e : if e is *false* when $\uparrow a$ occurs, transition 4 is taken and the resulting situation $\{12, 20\}$ is stable, because transitions 5 and 6 are not enabled. Otherwise, a *true* value for e when $\uparrow a$ occurs implies that $\{12, 20\}$ is unstable, because transition 5 can be taken immediately. Therefore, action ACT5 will not be performed in this case and a second evolution step will take place instantaneously to attain the stable situation $\{13, 20\}$, which causes ACT6 to be performed. This two-step evolution is represented on ASS of figure 3b by the transition labelled with ‘(4 then 5)’. The resulting situation $\{13, 20\}$, which can also be reached when $\uparrow e$ occurs in situation $\{12, 20\}$, is a deadlock because it has no output transition; deadlock treatment is discussed in Section 5. All the evolutions starting from situation $\{11, 21\}$ will take a few transitions to return to the initial state.

Figure 3. Extraction of the automaton ASS and the largest permissible behaviour.

4. Extraction of the largest permissible behaviour

The second step is to extract the largest permissible behaviour of Grafcet with respect to a number of given safety constraints. A dedicated intersection procedure is required to generate the corresponding safe, deterministic and reactive controller (SDRC).

The constraints to be satisfied are generally expressed in terms of prohibition of Grafcet actions and/or reinforcement of certain control executions. These constraints are specified using an automaton, S , that behaves as a reactive acceptor, accepting Grafcet inputs and outputs, and consequently evolving; each state of S is accompanied by a set of actions to be prohibited in the corresponding situation of Grafcet. This automaton should be deterministic; that is, the logical expressions associated with the output transitions of each state will never be *true* at the same time. Complex constraints can be obtained by synchronous composition of automata corresponding to simple constraints. The automaton S depicted in figure 3c, for example, indicates that the action ACT4 of Grafcet is to be prohibited in state q_2 , following the occurrence of $\uparrow f$ and then ACT1.

The intersection procedure goes through ASS and S to construct the automaton, SDRC, each state of which is given by the couple (state of ASS, state of S) together with a set of actions to perform, and those to prohibit, during the corresponding situation. A transition is added to the SDRC whenever it can be taken in S or in ASS, or in both. The intersection will be illustrated using the automaton ASS resulting from Grafcet (figure 3b) and the automaton, S , of figure 3c; the first evolutions of the resulting SDRC are depicted in figure 3d.

The initial state $\{(0), q_0\}$ of the SDRC corresponds to the initial situation of ASS and S , where ACT6 is performed and no action is prohibited by S . In this situation, the only possible transition is related to the rising edge of f , which results in a parallel evolution of ASS and S to states (1) and q_1 respectively. This evolution renders q_1 unstable, since its output transition is conditioned with the action ACT1, which is also performed during the current situation of ASS (state 1). Hence, the automaton, S , will immediately take this transition to activate state q_2 that is, in turn, unstable because its output transition is also conditioned with the activated action ACT2. This transition will therefore occur immediately to attain the stable state q_3 . As a result, a transition labelled with $\uparrow f$ will be created in SDRC to take this automaton directly from state $\{(0), q_0\}$ to $\{(1), q_3\}$. The latter state is accompanied by the actions to be performed in situation (1) of ASS (ACT1, ACT2, ACT3, ACT4), and action ACT3 to be prohibited by q_3 in S . Hence, ACT3 will be prohibited during controller execution, even though it should have been performed by Grafset in this situation. Three transitions are created from state $\{(1), q_3\}$ of the SDRC. The first occurs, when $\uparrow a.d$ becomes true, to activate $\{(10, 20), q_4\}$. This transition corresponds to the execution of transition (2) in ASS, followed immediately by the transition from q_3 to q_4 in S , because ACT1 is asleep in the corresponding situation of ASS (10, 20). The second transition, which corresponds to transition $\uparrow b$ in S when a is false, activates the state $\{(1), q_5\}$. The third output transition of $\{(1), q_3\}$ is also triggered by $\uparrow b$, but is related to the case where a is true. In this case, q_5 will be activated in S , and then will be immediately deactivated because its output transition will be immediately taken to activate q_0 . Hence, the resulting state in SDRC will be $\{(1), q_0\}$. Note that transition /ACT1 of S cannot be taken in $\{(1), q_3\}$, because ACT1 is performed in situation $\{1\}$ of ASS. The other transitions of SDRC are created in a similar way.

An intersection algorithm implementing the concepts informally presented above should result in a deterministic automaton SDRC because ASS and S are also deterministic. Reactivity can be guaranteed if the logical expressions (associated with the transitions of the SDRC) indicate the event provoking the transition from one stable situation of the SDRC to another. An extension of the technique adapted to generate ASS (see Section 3) can be used to achieve reactivity.

5. Identification and treatment of deadlocks

Certain deadlocks identified in the SDRC are not reachable because the reactions of the plant imply that the executions of the SDRC leading to the corresponding situations will never occur in reality. For example, the highlighted state, $\{(13, 20), q_1\}$, shown in figure 3d corresponds to a deadlock because it has no output transitions. However, if the plant behaviour in situation $\{(11, 20), q_0\}$ implies that a becomes true before a rising edge of b occurs, then the two transitions leading to $\{(12, 20), q_0\}$ and to $\{(13, 20), q_0\}$ cannot be taken, and hence the deadlock $\{(13, 20), q_1\}$ in the model will never be reachable in reality.

On the other hand, even though an SDRC is deadlock-free, the corresponding control execution can lead to a deadlock if it brings the plant to a state where the logical expression of the enabled transitions can never become *true*. For example, if the plant reaction, $\uparrow f$, can only occur consequent to the action ACT6, then the output transition of $\{(11, 23), q_0\}$ that is labelled with $\uparrow f$ will never be enabled, because ACT6 has not been performed by SDRC since the last occurrence of $\uparrow f$ (that is, ACT6 is not performed throughout the consecutive situations $\{(1), q_3\}$,

$\{(10,20),q_4\}$, $\{(11,20),q_0\}$, $\{(11,21),q_0\}$, and then $\{(11,23),q_0\}$). In this case, the situation $\{(11,23),q_0\}$, which appears to be normal in SDRC may correspond to a deadlock during real execution if the plant behaviour implies that b cannot become true before the occurrence of $\uparrow a$. Indeed, none of the output transitions of $\{(11,23),q_0\}$ can be enabled in this case. An intuitive solution to avoid this deadlock consists of prohibiting the occurrence of $\uparrow d$ in the upstream state $\{(11,21),q_0\}$ by performing an appropriate control action. For example, if the reaction $\uparrow d$ is known to be related to ACT4, then the corresponding transition can be avoided by inhibiting ACT4 in $\{(11,21),q_0\}$.

The above discussion shows that some knowledge of plant behaviour is required to be able to identify and to treat deadlocks. Indeed, during the specification phase, the control designer tends to implicitly integrate an image of the plant within the developed Grafset, in order to express the existing causality relations between Grafset actions and the consequent plant reactions. The underlying hypothesis is that all the reactions of the plant and their interleaving are known precisely and in advance, for each control action. Unfortunately, this hypothesis is restrictive and unrealistic for the case of complex systems involving partial Grafsets and parallel evolutions related to the different elements of the plant. The theoretical behaviour of the resulting model may, therefore, be different from its behaviour when it is executed to control the plant. This behavioural difference is the main cause of deadlocks, such as the ones discussed above.

It therefore becomes necessary to make the image of the plant explicit, by connecting an appropriate abstract model for each of its elements. However, the precise description of the behaviour of the plant is not trivial and the difficulty lies in the choice of the aspects to be modelled and the degree of granularity of the required model (Zaytoon and Carré-Ménétrier 1999). Hence, a modular approach is required to extract a simple model for each of the elements of the plant. Such a model can be derived using automata that accept the control actions, and react by changing the logical values of Grafset inputs. For example, the automata depicted in figure 4 may represent the behaviour of two elements of a hypothetical plant related to the example given in figure 3. The first automaton shows that the plant reaction, $\uparrow f$, can only occur following action ACT6, whereas the second reflects the causality relationship between ACT3 and ACT4 on the one hand, and d and c on the other.

The intersection of the automata of the different elements of the plant with the SDRC results in a global automaton reflecting the real execution of the system. Deadlocks that will be identified in this automaton will correspond to real deadlocks that may occur during system execution, provided that the plant is correctly modelled. Algorithms, such as those proposed by Ramadge and Wonham (1989), can provide the required corrections to avoid the real-execution deadlocks: for example, inhibiting ACT4 in $\{(11,21),q_0\}$ to avoid the reaction $\uparrow d$ and, by consequence, to

Figure 4. Automata representing a possible model for two plant elements.

avoid the state $\{(11,23),q_0\}$ of the SDRC. However, the success of such an automatic synthesis approach is conditioned by the availability of accurate models of the plant, which is difficult to achieve. A more pragmatic and realistic approach is therefore to highlight the identified deadlocks of the model resulting from the intersection, use a trace of the ‘deadlocking’ evolutions, and use the corrections proposed by an automatic synthesis approach. The designer can verify the usefulness of these corrections, or propose ones that are more appropriate. The feedback provided by such an approach would also assist the designer to identify errors in the model of the plant or to relax the imposed safety constraints if necessary. The synthesis approach can then be iterated on the basis of the refined models to generate the required correct controller, COR.

6. Implementation of the correct controller

The final step consists of implementing the correct controller by taking into account the constraints related to the target-execution architecture. A mono-processor architecture is proposed here, whereas issues related to the implementation of Grafcet on a distributed architecture are addressed in Kouthon *et al.* (1996). The proposed implementation architecture executes the correct controller and correspondingly updates the state of Grafcet, so as to visualise the execution of the control model and to highlight the corrections induced by the synthesis framework. This implementation (figure 5) is based on the use of a state/transition/action table, an input/output driver, a sequencer, and a module for update and animation of Grafcet. Each entry of the table corresponds to a state of the correct controller, COR. It includes the following elements related to the state: the state index, a list of active Grafcet steps, the actions to perform and to prohibit; and the output transitions given in terms of the firing event, the associated logical expression, and the destination state. The input/output driver transmits the actions of the current situation to the plant and receives the plant inputs. An interrupt-driven architecture is used to guarantee reactivity; when a plant-originated event occurs, the sequencer goes through the events corresponding to the current state of the table to determine the interrupt source. The next state associated with this event becomes the new current state, and the actions to be performed are sent to the input/output driver. The update module provides a synthetic image of control execution, and allows the designer to easily recognize the control tasks to be maintained, restricted or inhibited by the synthesis approach.

7. A synthesis framework

A global synthesis framework has been proposed by the authors to formalize the steps presented above. This framework (figure 6) is used to generate the required

Figure 5. Architecture of control implementation.

Figure 6. A framework for the synthesis of a correct control implementation for Grafset.

correct controller implementation (automaton COR) for a given Grafset, subject to a number of safety constraints related to the modelled plant. It is based on the use of a supervisor and a controller; the supervisor enforces the given safety constraints, whereas the controller (given by Grafset) directs the system toward the desired goal, to accomplish a specific set of tasks. An intuitive presentation of the different steps involved in this framework is given below; for the formal algorithms, see Zaytoon *et al.* (1999). A CASE tool (Carré-Ménétrier *et al.* 1999) has also been developed using C++ to implement the proposed framework.

The first step consists of modelling the plant behaviour (as spontaneous event generators; Ramadge and Wonham 1989) and the safety constraints using automata, and specifying the required control behaviour using Grafset. The activation and deactivation of Grafset actions correspond to controllable events, Σ_c , because it is possible to prevent their occurrence by an appropriate conditioning of Grafset actions. Uncontrollable events, Σ_u , are initiated by the plant. These events, which cannot be disabled by control action, are associated with the rising and falling edges of Grafset inputs. This interpretation can be considered as an adaptation of Balemi's scheme (Balemi *et al.* 1993) to the case of Grafset.

The second step is used to obtain the automaton SUP, corresponding to the supremal language of the supervised plant by applying the synthesis algorithm proposed in Kumar (1991). The transition structure of SUP represents the maximum permissible behaviour of the controlled plant with respect to the given constraints. The third step consists of extracting the automata of stable situations, ASS, of Grafset (see Section 3) by applying the algorithm given in Roussel (1994).

In the fourth step, the sequences of events that can be generated both by ASS and by SUP are extracted by intersection of the two corresponding languages; a dedicated synchronization algorithm (Zaytoon *et al.* 1999) goes through ASS and

SUP to construct the automaton SYNC. A transition labelled with a controllable event is included in this automaton when SUP accepts this event, provided that it is not in contradiction with the current output state of ASS. Transitions labelled with uncontrollable events are included in SYNC whenever SUP can generate these events, irrespective of whether or not ASS can accept them. In this way, only the control evolutions that are acceptable by the supervisor are retained in SYNC.

The reduction step is then used to generate the maximum reactive non-blocking execution controller by removing the deadlock situations from SYNC, as well as the evolutions that may lead to these situations. The resulting automaton COR represents the most permissible subset of the behaviour of Grafcet that is deadlock-free, and satisfies the imposed constraints. The states of COR correspond to the stable situations of the controller to be implemented; they contain information about the active Grafcet steps, and the actions to be activated and prohibited in the corresponding situation. The transitions of COR correspond to uncontrollable events driving the controller from one stable situation to another.

The last step consists of executing the generated correct control COR and correspondingly updating the Grafcet, so as to visualise the execution of the control model and to highlight the corrections induced by the synthesis framework. The size of the automaton COR is proportional to $|ASS| \times |SUP| \times 2^{U/2}$, where $|ASS|$ and $|SUP|$ represent the number of states of ASS and SUP respectively, and U is the number of uncontrollable events in the system.

8. Example

The synthesis framework presented above will be illustrated using a process for drilling holes along two rows on a working piece (figure 7). The first row of holes is to be drilled when the piece is in its initial position, whereas the second row corresponds to the position indicated by the sensor, p . Another sensor, $piece$, detects the arrival of a new piece. Control specifications are given by three partial Grafcets (G1, G2, G3), which are used to control the drilling process, the translation of the drill, and the displacement of a piece respectively (figure 8).

Figure 7. Drilling system.

Figure 8. Grafcet for the drilling process.

8.1. Control specifications

8.1.1. Grafcet G1

The arrival of a piece activates step 101 to lower the drill. When the low position (*l*) is attained, the drill is commanded to rise. Two evolutions are then possible when the high position (*u*) is reached, depending on whether the hole has been correctly drilled (*opt* = 1) or not (*opt* = 0). In the first case, synchronization step 103 is activated, whereas transition 3 is taken in the second case to drill the hole again.

Two cases are also possible at step 103:

- (1) activation of step 101 to restart a new drilling cycle whenever the next drilling point is reached ($\uparrow \text{step}$) or if the piece is pushed to the second row ($\uparrow p$); or
- (2) activation of step 104, when the piece is evacuated ($\uparrow f$), to wait for the synchronization of the three Grafcets in order to treat the next piece.

8.1.2. Grafcet G2

When a piece arrives, step 201 waits for the drilling of a hole and then step 202 advances the drill by one step. As soon as the next drilling position is reached, and provided that the final position of the translation movement is not attained (*n* = 0), transition 10 is taken to repeat steps 201 and 202. On the other hand, if *n* is true, transition 11 activates the waiting step (203). When the last hole is drilled, step 204 returns the drill to its initial position, indicated by *d*. Two cases are possible here depending on the state of steps 304–6, which indicate whether the second row of holes has been drilled. In the first case, step 205 is activated to indicate that the drilling of the piece is terminated, and to wait for the global synchronization of the three Grafcets. If the second row has not been drilled, the arrival of a piece at position, *p*, allows transition 13 to be taken, and hence the drilling of the second row to be started.

8.1.3. *Grafcet G3*

When the first row has been drilled, step 302 pushes the piece to position, p , which activates step 303 to wait for the end of the drilling of the second row. Then step 304 waits for the evacuation of the finished piece. In this case ($f = 0$), step 305 ejects the piece, and step 306 is then activated to wait for the synchronization of the global cycle.

8.2. *Plant modelling*

The model corresponding to the free behaviour of the plant is given by three automata (figure 9) representing the evacuation of a piece, the drilling process and the displacement of the drill. The first automaton represents the evacuation of a piece from its initial position, through the position indicated by the sensor, p , until it is ejected from the plant (position f). This evacuation is carried out by the displacement system when the action, *Push*, is performed. The deactivation of this action in state 2 (or 3), entails the activation of state 4 (or 5), in which the displacement system returns to its initial position, and the piece remains immobile. Therefore, the rising edge of f cannot occur in this state, and only the reactivation of *Push* allows the evacuation of the piece to resume. The two other automata can be described in a similar way.

Note that the objective here is not to give the details of the model of the plant, but rather to emphasize the difficulty in developing such a model. The difficulty is caused by the low-level manipulated concepts, such as the use of rudimentary events and the inappropriate semantics of the supervisory control theory with respect to plant/controller interactions. As will be indicated in the conclusion, a more adapted model should be used for the plant.

8.3. *Constraints modelling*

Two constraints are imposed to guarantee safe functioning of the system: drilling should be avoided during the displacement of the drill or the piece, and the drill should rise before any other movement of the drill or the piece.

Six automata are used to model the first safety constraint. The first and second automata depicted in figure 10 indicate that drilling and displacement of a piece cannot be performed simultaneously (Σ is the set of events). In figure 10a, the automaton is used to avoid the simultaneous activation of the actions, *Push* and *Descend*. When these actions are not performed, the system remains in state 0, which has a self-looping transition that implies that all other events can occur in this state. The activation of the action, *Descend*, allows the attainment of state 1, in which all events are authorized, apart from the activation of *Push*. State 0 can be reached again when *Descend* is deactivated. The same explanation applies for the action *Push*, with respect to state 2.

The automaton of figure 10b implies that the actions *Push* and *Rise* cannot be performed at the same time. Four identical automata are also used to avoid a simultaneous activation of the vertical and the horizontal translation of the drill. Figure 10c depicts one of the three automata used to represent the second constraint. This automaton specifies that the drill should rise before the displacement of the pieces. The transition to state 1 occurs when the action *Rise* is activated. In this state, the action *Push* can be performed, and its deactivation entails a transition back to state 0. Two similar automata are used to model the occurrence of the action *Rise*, before the actions *Return* and *Advance*.

Figure 9. Automaton describing the drilling process.

Figure 10. Safety constraints.

8.4. Results

The application of the steps of the synthesis framework (figure 6) generates the correct control implementation automaton COR, which is composed of 1050 states and 2725 transitions. A partial evolution trajectory for this controller is depicted in figure 11, to illustrate one of the corrections induced by the synthesis process. Each of the states of COR is characterized by a set of active Grafcet steps, as well as the set of actions to perform and those to prohibit when the state is active. The actions *Return* and *Push* are performed during state 84. When a rising edge of *p* occurs, state 88 is activated and this state only performs the action *Return*.

Note that the action *Descend* associated with the corresponding situation of Grafcet {101,204,304} is prohibited to avoid drilling during the translation of the drill. This correction will be highlighted during control execution. Hence, the first safety property is satisfied in this case, and the action *Descend* is delayed until the end of the translation of the drill ($\uparrow d$) and the activation of state 96 of the control automation. In a similar way, all the evolutions of the generated control implementation will satisfy the given constraints.

9. Conclusions

The control of discrete manufacturing systems entails the specification of control tasks and the implementation of such tasks by means of a safe, reactive, deterministic and deadlock-free controller. This paper has presented the issues and the concepts related to the synthesis of a correct controller implementation starting from high-level specifications given in Grafcet. It has also emphasized the necessity of extending the traditional Grafcet specification method by introducing an explicit model of the controlled plant, so as to synthesize controllers that have a guaranteed deadlock-free execution. These concepts were formalized and implemented through a synthesis framework that uses supervisory control theory and automata to model the plant. However, and despite the formal backgrounds of these models, they do not represent convenient or intuitive approaches for practical systems.

Current research is directed at improving the applicability of the proposed formal framework by using more suitable models such as high-level Petri nets instead of automata to represent the plant. Existing techniques for the synthesis of supervisory policy for nets representing the plant can be used in this case instead of step 2 of figure 6. This will entail the modification of the intersection procedure to accommodate the semantics of the resulting supervisor. The proposed concept of the alternation between an analysis and a synthesis phase should also be formalized and implemented to provide user-guidance for the analysis and correction of design errors for sizeable applications.

Figure 11. Partial control execution.

References

- ARINEZ, J. F., BENHABIB, B., SMITH, K. C. and BRANDIN, B. A., 1993, Design of a PLC-based supervisory-control system for a manufacturing workcell. *Proceedings of the Canadian High Technology Show and Conference*, Toronto.
- BALEMI, S., HOFFMANN, G. J., GYUGYI, P., WONG-TOI, H. and FRANKLIN, G. F., 1993, Supervisory control of a rapid thermal multiprocessor. *IEEE Transactions on Automatic Control*, **38**, 1040–1059.
- BRANDIN, B. A., WONHAM, W. M. and BENHABIB, B., 1991, Discrete event system supervisory control applied to the management of manufacturing workcells. In V. C. Venkatesh and J. A. Mgeough (eds), *Computer-Aided Production Engineering* (Amsterdam: Elsevier), pp. 5270–536.
- CARRÉ-MÉNÉTRIER, V., NDJAB, C., GELLOU, F. and ZAYTOON, J., 1999, A CASE tool for the synthesis of optimal control implementation of Grafcet. *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, Tokyo, pp. 1.796–801.
- CHARBONNIER, F., ALLA, H. and DAVID, R., 1999, The supervised control of discrete-event dynamic systems. *IEEE Transactions on Control Systems Technology*, **7**, 175–187.
- DAVID, R. and ALLA, H., 1992, *Petri Nets and Grafcet: Tools for Modelling of Discrete-Event Systems* (Englewood Cliffs: Prentice Hall).
- FRACHET, J. P. and COLOMBARI, G., 1993, Elements for a semantics of the time in Grafcet and dynamic systems using non-standard analysis. *Automatic Control Production Systems APII*, **27**, 107–125.
- GIUA, A., 1996, Petri net techniques for supervisory control of discrete event systems. *Proceedings of the First International Workshop on Manufacturing and Petri Nets*, Osaka.
- HAREL, D., 1987, Statecharts, a visual formalism for complex systems. *Journal of Science of Computer Programming*, **8**, 231–274.
- HOLLOWAY, L. E. and KROGH, B. H., 1990, Synthesis of feedback control logic for a class of controlled Petri nets. *IEEE Transactions on Automatic Control*, **35**, 514–523.
- INTERNATIONAL ELECTROTECHNICAL COMMISSION, 1991, *Preparation of Function Charts for Control Systems*, revised version. IEC 848.
- JAFARI, M. A. and BOUCHER, T. O., 1994, A rule-based system for generating a ladder logic control program from a high-level system model. *Journal of Intelligent Manufacturing*, **5**, 103–120.
- KOUTHON, T., DECOTIGNIE, J. D. and KOPPENHOEFER, S., 1996, On distribution of Grafcet software. *Proceedings of the IMACS/IEEE Symposium on Discrete Events and Manufacturing Systems CESA'96*, Lille, pp. 498–506.
- KUMAR, R., 1991, Supervisory synthesis techniques for discrete event dynamical systems. PhD thesis, University of Texas.
- LAUZON, S. C., MA, A. K., MILLS, J. K. and BENHABIB, B., 1997, An implementation methodology for the supervisory control of flexible manufacturing workcells. *Journal of Manufacturing Systems*, **16**, 91–101.
- LHOSTE, P., FAURE, J. M., LESAGE, J. J. and ZAYTOON, J., 1997, Comportement temporel du Grafcet. *European Journal of Automation JESA*, **31**, 695–711.
- LI, Y. and WONHAM, W. M., 1993, Control of vector discrete-event systems I—the base model. *IEEE Transactions On Automatic Control*, **38** (8), 1214–1227.
- RAMADGE, P. J. and WONHAM, W. M., 1989, The control of discrete-event systems. *Proceedings of the IEEE*, **77**, 81–97.
- ROUSSEL, J. M., 1994, Analyse de Grafcets par génération logique de l'automate équivalent. PhD thesis, Ecole Normale Supérieure de Cachan, France.
- UNION TECHNIQUE D'ELECTRICITÉ, 1993, *Function Chart GRAFCET: Extension of Basic Principles*. UTE C03-191, June.
- YAMALIDOU, K., MOODY, J. O., LEMMON, M. D. and ANTSAKLIS, P. J., 1996, Feedback control of Petri Nets based on place invariants. *Automatica*, **32**.
- ZAYTOON, J. and CARRÉ-MÉNÉTRIER, V., 1999, Grafcet et graphes d'état: comportement, raffinement, vérification et validation. *European Journal of Automation JESA*, **33**, 751–782.

- ZAYTOON, J., NDJAB, C. and CARRÉ-MÉNÉTRIER, V., 1999, Grafcet et graphes d'état: synthèse hors ligne de la commande. *European Journal of Automation JESA*, **33**, 783–814.
- ZHOU, M. and DICESARE, F., 1993, Petri Net Synthesis for Discrete Event Control of Manufacturing Systems (Boston: Kluwer).
- ZHOU, M. and TWISS, E., 1998, Design of industrial automated systems via relay ladder logic programming and Petri nets. *IEEE Transactions on Systems Man and Cybernetics—Part C*, **28**, 137–150.