Application of Discrete-Event-System Theory to Flexible Manufacturing

S.C. Lauzon, A.K.L. Ma, J.K. Mills, and B. Benhabib

The anticipated extensive use of flexible-manufacturing workcells in the future has encouraged recent research efforts on the development of automatic supervisory-control methodologies. However, despite intensive research on the theoretical aspects of the control of manufacturing workcells, modeled as discrete-event systems (DES), a limited amount of research has been reported on the implementation of DES-based supervisory controllers. In this article, such a generalized implementation methodology, that utilizes recent theoretical advances in conjunction with programmable-logic-controller (PLC) technology, is presented. The two primary advantages of the proposed methodology are: (i) the utilization of limited-size control strategies that can be efficiently generated on-line, and which are conflict- and deadlock-free by construction (via controlled-automata DES theory); and (ii) the use of PLCs, which are currently the most suitable and widely employed industrial process-control technology.

In our proposed methodology, a host personal computer (PC) possesses an on-line capability for the automatic generation of supervisory-control strategies, and their downloading to a PLC as required. The PLC, in turn, is responsible for monitoring the workcell, reacting to events and enforcing device behavior based on the current control strategy residing in its processor. A supervisory controller developed based on this approach was successfully implemented for a manufacturing workcell in our laboratory.

Introduction

Motivation

A generalized implementation approach to the supervisory control of flexible-manufacturing workcells is proposed in this article. The proposed methodology utilizes recent theoretical advances in control theory, in conjunction with programmable logic controller (PLC) technology. The flexible workcell considered has two main characteristics: (i) hardware flexibility—part transfer is via a robotic manipulator within the workcell; and (ii) software flexibility—alternate part routes, existing within the workcell, can be utilized when needed by the workcell controller.

Manufacturing workcells for *discrete* production usually exhibit the characteristics of a discrete-event system (DES). They are event driven, discrete in time and space, usually asynchronous, and typically nondeterministic. In the past, DESs have usually been sufficiently simple that intuitive or *ad-hoc* control solutions have been adequate [1]. However, the increasing complexity of these systems has created a need for formal approaches for their analysis and control. Supervisory controllers for such systems are required to perform the following three main tasks: (i) monitoring the workcell behavior, (ii) controlling and evaluating workcell status according to a supervisory-control law, and (iii) enforcing device behavior. A variety of methodologies have been proposed to address the control of DESs within a formal theoretical framework. These methodologies, usually classified according to their modeling technique, include Petri nets [2,3], real-time temporal logic [4,5], and controlled-automata [1,6].

Controlled-automata theory provides similar modeling features as Petri nets do; however, it differs in that the design of supervisory controllers is based on logic and formal language theory. The theory ensures that the plant under supervision will behave optimally (deadlock-free) and legally (according to given specifications). The supervisory controller thus obtained is correct by construction.

Despite intensive research efforts on the theoretical modeling and development of DES-based supervisory controllers, there has not been significant research reported on the application of such controllers. In [7,8,9,10], implementations of DES-based controllers for manufacturing workcells were presented. In both [7] and [8], while the employed modeling techniques were different (Petri-net versus controlled-automata), a computer in which the supervisory-control strategies resided was directly wired to the devices to be controlled. In [9,10], however, the workcell devices were controlled using a PLC. The control strategy in [9] was developed based on controlled-automata, as in [6], manually translated into a ladder-logic code, and subsequently programmed manually into the PLC. In another PLC-based implementation, [10], a rule-based method is presented to derive a ladder-logic program from a high-level system model. The proposed method uses Petri-net modeling as an intermediate step in moving from a high-level description of a control-strategy to the Boolean format of the corresponding ladder-logic description.

Our proposed methodology herein is based on combining a PLC and a PC into an integrated system that can communicate effectively with local controllers. The integrated controller, thus, draws on both technologies' strengths, while offsetting some of their shortcomings. PLCs are re-programmable controllers with built-in A/D and D/A converters and allow convenient connection to workcell devices. They are also readily adaptable for the application of DES-based control strategies. Yet their programming is considered to be time-consuming and error-prone. Con-

The authors are with the Computer Integrated Manufacturing Laboratory, Department of Mechanical Engineering, University of Toronto, 5 King's College Road, Toronto, Ontario, Canada, M5S 1A4, email beno@me.utoronto.ca. *An earlier version of this article was presented at the 1995 IEEE International Conference on Robotics and Automation in Nagoya, Japan.*

 0272-1708/96/$05.00©1996IEEE

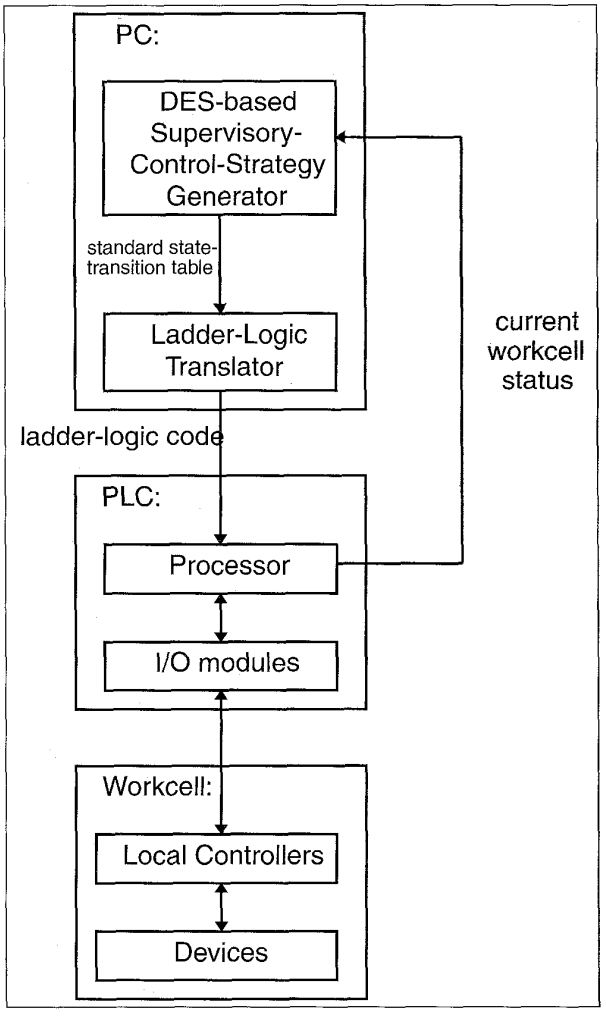


Fig. 1. Proposed general methodology.

versely, PCs are more readily programmable but are more difficult to connect to devices for I/O purposes.

Overview of Proposed Methodology

The specific approach for the implementation of a supervisory-control system, proposed in this article, is shown in Fig. 1. Within the host PC, a DES-based control strategy, in a format specified by a standard state-transition table, is received by an automatic ladder-logic translator. It is thereafter translated into a ladder-logic code and downloaded to a PLC processor that supervises the manufacturing workcell. This process is implemented as a closed-loop automatic system, where different control strategies would be downloaded from the PC to the PLC in an on-line manner, in response to the occurrence of events in the workcell.

A strategy downloaded to the PLC processor is only a subset of a comprehensive control strategy. A complete DES control strategy to describe all possible events in the workcell would be very large, and as a consequence need too much storage space on a PLC. Furthermore, it would also be subject to an exponential explosion of required states, and thus might not be obtainable. According to our proposed approach, when events in the workcell deviate from an initially downloaded nominal strategy, it would then be possible to re-synthesize and re-download a new strategy to the PLC.

A key function in our proposed methodology is the ability to automatically obtain ladder-logic programs. Knowledge-based generators that take as input a description, in English prose, of the (continuous) process and create task codes for a PLC have been reported [11,12]. However, in the context of our proposed methodology, it is not a *generator* but rather a *translator* that has been developed. In this case, the role of the translator is to facilitate the process of going from a DES strategy to a usable PLC source code.

In the following sections, DES modeling, based on the work presented in [1,13], will be reviewed first. The proposed supervisory-control implementation approach will be detailed thereafter. The experimental work carried out in our laboratory will conclude this article.

Controlled-Automata

Flexible-manufacturing workcells can be modeled as controlled-automata, and corresponding supervisory-control strategies can be developed using the fundamental tools of DES theory presented in [1]. Since automata theory is founded upon formal language theory, we first define below the modeling issues, followed by synthesis of control strategies.

Modeling

A DES is typically represented as a set of states that are linked by transitions. They represent controllable or uncontrollable events in a manufacturing workcell. The design of a supervisory controller involves the formulation of the workcell DES models and their synthesis into a control strategy, where the DES itself is thus controlled as a generator of a formal language [1].

In automata theory, the set of state transitions, called events, can be considered as the *alphabet* of a language, and the sequence of events as *words* within the language. An automaton can be viewed simply as a device that generates a language by the manipulation of the alphabet according to a specific set of rules.

Proceeding to the description of an automaton, its basic structure is given by

$$G = \{ \Sigma, Q, q_0, \delta(\sigma, q), Q_m \} , \qquad (1)$$

where Σ is a non-empty finite set of distinct symbols that contains the alphabet. Q is the set of states in the system. The initial state is given by q_0. The transition structure, $\delta(\sigma, q)$ is a function δ: $Q \times \Sigma \rightarrow Q$ that maps, one-to-one, the current state and event to the next state. A transition is the change from one state to another in response to the occurrence of a physical event. The input to the function is the current workcell state q and the event, as labeled by σ, where $\sigma \in \Sigma$. The output of the function is the next workcell state, due to the occurrence of the event σ. Finally, Q_m is a set of marker states. A marker state may be considered as a state that represents a "completed task". However, there is no implication that a marker is a terminal state.

The two main advantages of designing supervisory controllers based on [1] are as follows: a supervisory-control strategy

developed using the tools within the theory is correct by construction (i.e., the generated language is non-conflicting and non-blocking), thus guaranteeing that workcell behavior will not violate manufacturing-process specifications; and the supervisory-control strategy is guaranteed to be the most permissive (i.e., all possible events are allowed) within the given specifications.

Control Strategy Synthesis

To illustrate the synthesis of control strategies, a controlled-automata model for a two-machine workcell synthesized from the languages of two plant models is considered. The basic controlled-automata plant model is shown in Fig. 2. The events α ("begin an operation") and μ ("repair the machine") are controllable events, while β ("complete an operation") and λ ("machine breakdown") are uncontrollable. The *nominal* (i.e., preferred) part route in the workcell is to have a part operated on first by Machine 1, then by Machine 2.

If the model for Machine 1, as in Figure 2, is labeled *plant_1*, and the model for Machine 2 is labeled *plant_2*, the combined system model is constructed by applying the *shuffle product*[1], thus creating the automaton *plant*. This operation is represented by

$$plant = \text{sync}\ (plant\_1, plant\_2). \tag{2}$$

To enforce the desired behavior of the system, that is, to have a part operated on by Machine 1, then by Machine 2, the specification that constrains the plant's behavior must also be modeled and combined with the aggregate system model *plant*. The specification, *spec_1*, can be modeled as in Fig. 3, which states that a β-event generated by Machine 1 must occur before an α-event is generated by Machine 2. Generally, when there are more than one specification, they are combined by the application of the *meet*[2] operation.

The specification *spec_1* serves two purposes in this case. The first purpose is to establish the correct order of operation. This is accomplished by specifying that only β_1 and α_2 can cause a change in state, where β_1 occurs before α_2. The second purpose is to establish the buffer size between the first and second machines. The buffer size is limited to one by not including events α_1 and β_1 in the second self-looping[3] event ("b"), as it is then impossible for another α_1 event to occur. The first self-looping event ("a") simply permits all events to occur, with the exception of α_2.

To synthesize the system model, *supervisor*, for the supervisory controller, the *supcon*[4] operation is performed on the results of the shuffle product and meet operations,

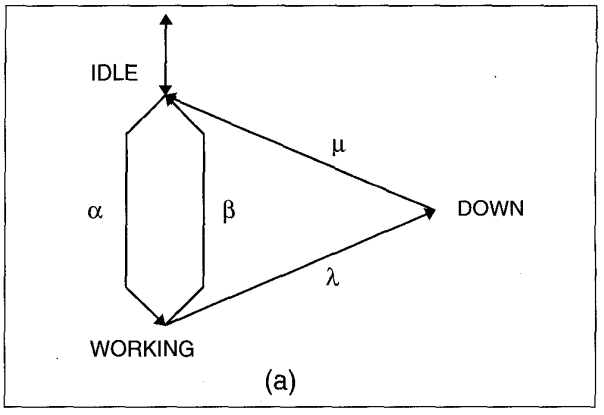


Fig. 2. A basic controlled-automata plant model.

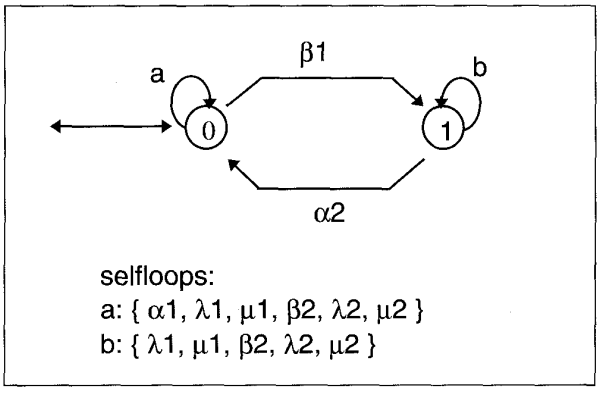


Fig. 3. Specification for the nominal part route.

$$supervisor = \text{supcon}\ (spec\_1, plant). \tag{3}$$

The state-transition diagram, generated from the supervisor, for a nominal route of a part in the workcell is shown in Fig. 4. According to the nominal part route, a part enters the workcell for operation by Machine 1 (e.g., event α_1, from State 0 to State 1). Subsequently, it is expected that the Machine 1 will either complete its operation (e.g., event β_1, from State 1 to State 2) or break down (e.g., event λ_1, from State 1 to State 3). In the former case, Machine 2 starts its operation (e.g., event α_2, from State 2 to State 4). As the second machining device starts its operation, the nominal strategy allows the Machine 1 to begin operating on a new part (e.g., event α_1, from State 4 to State 5). Once the second machining device has completed its task, it is assumed that a transport device, unmodeled herein, removes the part from the workcell.

On-Line Synthesis

Manufacturing workcells have the primary characteristic of being nondeterministic, where a large number of possible (controllable and uncontrollable) events may occur at a given state. This characteristic of nondeterminism complicates the implementation of an effective supervisory controller, since even moderately complex systems (e.g., four to six machines which

[1]If the plant models do not share identical events, the shuffle product is equivalently referred to as the synchronous product (defined as *combined_model* = sync (*model_1, model_2*)) [1].

[2]The meet operation is simply defined as the intersection of two languages [1].

[3]A self-looping event is an event that may occur at a state, but it does not cause a change to another state.

[4]The supcon operation determines the supremal-controllable language, that is, the legal language for the supervisory control [1].

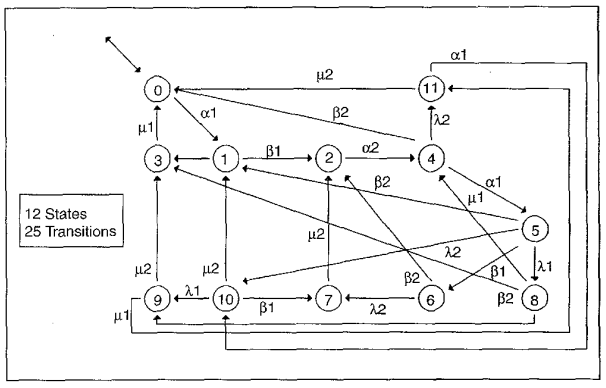


Fig. 4. State-transition diagram of a control strategy for a nominal part route.

process four to six different parts) may require very large DES strategies.

The problem of large strategies may be addressed by applying modular synthesis, aggregation, decentralization, and hierarchical techniques. However, these techniques have limited use, since they draw on the special characteristics of the many control objectives [14]. To address this problem, a hybrid supervisor with an alternate-routing mechanism can be utilized to interact with the *pure* DES controller when the system behavior deviates from nominal specifications.

The Hybrid Supervisory Controller (HSC), proposed in [13], is such a system. It comprises three main modules (Fig. 5): (i) a DES Supervisor, (ii) an Alternate-Strategy Driver (ASD), and (iii) a Diagnostic System. The DES Supervisor is responsible for nominal part routing. The ASD is a heuristic system that generates new part routes when an event deviates from the nominal strategy within the DES controller. The Diagnostic System's main task is to interpret sensory feedback from the workcell, and notify the DES Supervisor and the ASD of its interpretations [15].

Proposed Implementation Methodology

An essential element to the proposed supervisory-controller implementation methodology, as depicted in Fig. 1, is the ability to re-synthesize control strategies on-line for limited sub-sets of events in the manufacturing workcell. At any given time, we consider only a limited set of possible events for the development of a control strategy, such that it can be efficiently and automatically generated.

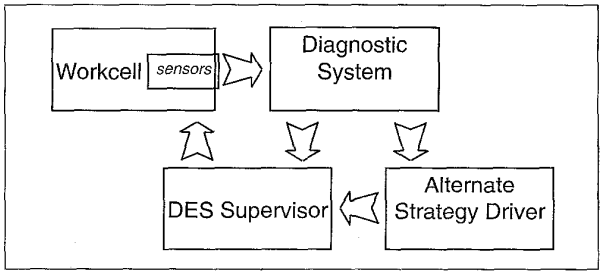


Fig. 5. An overview of the HSC.

Methodology

The principal automatic procedure for the proposed approach comprises: (i) obtaining a control strategy in the form of a standard state-transition table, (ii) translating the control strategy, via the automatic ladder-logic translator, into a ladder-logic program, and (iii) downloading the code to the PLC processor.

For the first step, it is assumed that the control strategy development capability would be residing in a PC. An essential element is the ability to automatically generate control strategies in an on-line manner. This objective can be achieved if only subsets of a comprehensive control strategy are synthesized and executed one at a time. These *limited-size* control strategies could still comprise multiple states and transitions This solution proposal is in contrast to the consideration of the comprehensive control strategy, which could contain a very large number of states and events not practically achievable, even if carried out in a computationally efficient manner.

As limited-size strategies are synthesized, their output are compiled into standard state-transition tables. The specifications of the standard state-transition table should be generic, namely, independent of the DES-modeling technique used to obtain the control strategy. The proposed format for a standard table entry is simply (q, σ, q'), where q is the exit (current) state, σ is the transition, and q' is the entrance (next) state. It can be noted that controlled-automata based techniques would normally output a control strategy in exactly this format. As briefly described below, Petri-net modeling techniques could as well output strategies in this format.

In Petri-net theory, both the availability of a machine to perform a required operation and the availability of parts are considered for the allowance of transitions between states. However, the responsibility for verifying the availability of parts can lie with the local controllers, freeing the supervisory controller to consider only the availability of machines. In other words, when the supervisor requests an operation to begin, it would be the local controller's responsibility to ensure the availability of the required resource before the operation can actually begin. Hence, the states in the standard state-transition table would only consist of states that deal with machine availability, as is the case in controlled automata.

The second step of the proposed approach is the input of the control strategy to a ladder-logic translator. The automatic translator receives as input the standard state-transition table, and the list of the PLC's input/output (I/O) reference addresses for each event type. The output of the translator is a text file containing the PLC source code.

The state-transition table input file contains the *number* of possible events for the control strategy at hand, as well as the possible events themselves (where each individual event that may occur in the workcell is labeled by a unique number). Our translator was designed such that only the I/O reference address files for the events in the state-transition table at hand would need to be read during the translation. Each possible event has its own I/O reference address file, with the event label incorporated into the file name (e.g., *ref11* for an event labeled as *11*). This renders the translator independent of specified workcell events and PLC wiring, since the information is not embedded into the hard-code part of the software.

It should be noted that a translator would have to be hardware dependent, since currently each PLC manufacturer provides its

own PLC programming language. However, all PLC programming languages are based on basic ladder logic. Therefore, the translator described in this article can be adapted to other PLCs with minimal effort. (An example of the translation operation for a DES-strategy statement is given in the appendix.)

In regard to PLC-to-PC communications, the primary concern is to monitor the data tables in the PLC processor and to be able to react to events as required. For example, if a machine failure sets a certain I/O bit in the PLC's I/O data table, a monitoring program residing in the PC can read the value of this bit (1 or 0) and react to the fault by reporting the fault for the synthesis and translation of a new control strategy.

Implementation Issues

For smaller systems consisting of only a few machines, one may generate alternate strategies that consider different sets of events and translate to ladder-logic programs in an *off-line* manner. These would be subsequently downloaded to a PLC in an on-line manner, based on events occurring in the workcell. An on-line capability of generating alternate strategies, on the other hand, requires the use of a *real-time* controller. The supervisory controller must have the capability of generating the control strategies as required, based on feedback information on the occurrence of events in the workcell. This information would then be written into a standard state-transition table and sent to the automatic translator, which would translate all the possible events into a ladder-logic code, thus closing the loop for the on-line operation of the controller. Although deadlock or conflicts may arise while operating in this mode, the supervisory controller can successfully resolve such problems by the use of tools within DES theory (i.e., the application of the supcon operation), or by a heuristic means [13], as utilized by the controller in our laboratory. In the latter case, a heuristic algorithm is utilized, which detects potentially deadlocked parts and then re-routes the parts out of the deadlock.

An important aspect that has to be addressed in the creation of the automatic ladder-logic translator is to ensure that the ladder-logic code written for each event type at each workcell state renders each event type uniquely distinguishable from an identical event type that may occur at another workcell state. (In other words, an α-type event that occurs when the workcell is in state X, and that triggers a certain output reaction in the PLC program, has to be distinct from the same α-type event that could occur within the workcell at State Y, and that would trigger a different output reaction.) To render each possible event unique, the workcell state has to be included as an input condition in the ladder-logic code, thus enabling the correct output. This implies that the PLC keeps track of the workcell states directly, and not of the machine states.

In the implementation phase, one must also consider that controllable events do not occur spontaneously, as assumed previously in [1,6], but rather only as *responses* to requests sent by the supervisory controller. To accomplish this, the basic DES plant model must be modified such that controllable events are considered to comprise two distinct phases [8]: a request by the supervisor and the response from the plant, as shown in Fig. 6. The supervisor considers a change of state only when a response is received from the plant. Thus, when a translator is developed, another aspect addressed must be a capability to look ahead to the next state. This is in order to determine whether an event

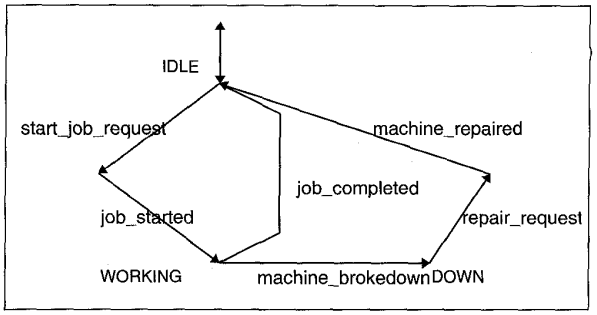


Fig. 6. The extended plant model [8].

request will be required in the ladder-logic code. Since, at each state, there is more than one possible transitional event, this capability is in essence a mechanism for choosing one of the next possible controllable events.

Experimental Results

A DES-based supervisory controller for a robotic workcell was implemented in our laboratory. During the experimental run-time, different limited-size control strategies were withdrawn from a database, translated, and downloaded to the PLC in an on-line manner in response to a machine-failure simulation. The following is a description of the hardware and the software utilized, and the test results obtained.

Experimental Set-Up

The experimental hardware consisted of an 80486 host PC, an Allen-Bradley PLC-5/11 Programmable Controller, two machining devices, an industrial GMF S-100 robot, and a pallet conveyor, as shown in Fig. 7. One of the machining devices was an industrial CNC milling machine, while the second one was a simple *switch-box* designed and built to emulate the behavior of a machining device. The CNC milling machine and the robot were linked to the PLC via a dedicated 80486 PC that acted as their local controller. The physical communication links were achieved via Allen-Bradley's Data Highway Plus (DH+) network, and a 1784-KT card that resided in each of the PCs. The conveyor and the other machining center were linked directly to the PLC processor.

The software for PLC-to-PC communications utilized Allen-Bradley's C-library function calls, linkable with commercial C compilers, from their Interchange software. The function calls allow the user to change processor modes, to download ladder-logic programs, and to read and write to the I/O data tables in the processor's memory.

Experimental Procedure

To demonstrate the feasibility of our proposed methodology, the experimental testbed was considered as a two-machine-workcell DES model. For simplicity purposes, the robot and the conveyor were considered as transport devices and they were not part of the DES model. The nominal state-transition diagram for a two-machine workcell (developed with the controlled-automata tools from [1]) is as shown in Fig. 4. The α-type events represent the start of a job, the β-type events represent the completion of a job, the λ-type events represent the breakdown

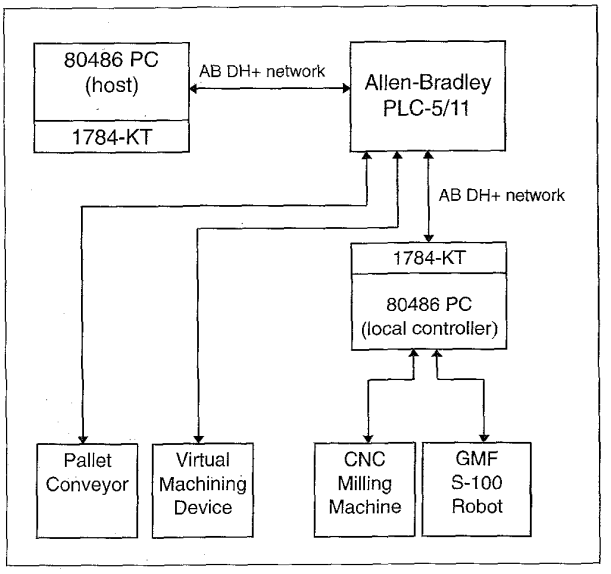


Fig. 7. Workcell setup.

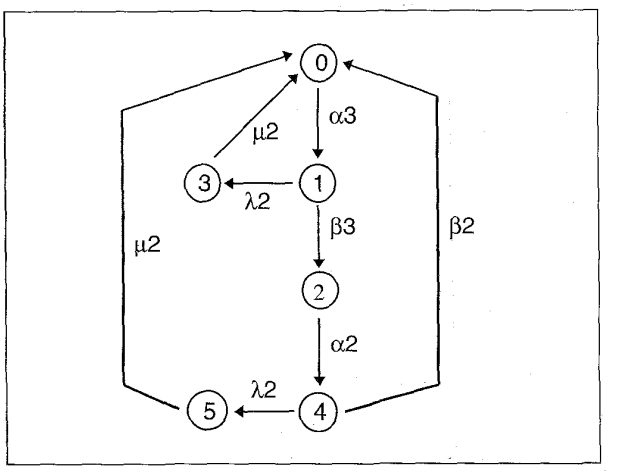


Fig. 8. State-transition diagram of a control strategy for an alternate part route.

of a machine, and the μ-type events represent the repair of a machine.

According to the nominal strategy, a part first enters the workcell via a pallet conveyor. When a limit switch senses that a pallet has arrived at its expected destination, it notifies the PLC, which in turn enables the CNC milling machine, via a *request* signal, to start its operation. The robot controller intercepts this request and activates the robot to transfer the part from the conveyor to the CNC milling machine. Upon the completion of this operation, the robot's controller notifies the PLC processor, which in turn allows the entry of another part into the workcell. In parallel, the CNC milling machine also notifies the PLC that it has started its operation. Subsequently, it is expected that the CNC milling machine will either complete its operation, and send the appropriate signal to the PLC, or break down, sending a different signal to the PLC.

In the former case, the PLC enables the second (virtual) machining device to begin its operation. The robot's controller once more intercepts this signal and transfers the part from the CNC milling machine to the second machining device. As the second machining device starts its operation, the nominal strategy allows the CNC milling machine to begin operating on a new part. Once the second machining device has completed its task, it is assumed that another transport device removes the part from the workcell.

In the latter case, namely when the CNC milling machine fails, the workcell is stopped, and an alternate control strategy, shown in Fig. 8, is downloaded to the PLC from the host PC. The alternate strategy requires a part to go through two consecutive operations on the second machining device. The operation that replaces the operation on the CNC milling machine is denoted by the subscript "3". Once the CNC milling machine is repaired, however, the nominal strategy is re-downloaded to the PLC's processor.

The failure of the second machining device, on the other hand, is considered "fatal" to the operation of the workcell, since it is assumed that the CNC milling machine is incapable of replacing it as an alternate device. In this case, the host PC instructs the PLC to await the repair of the CNC milling machine while maintaining the cell in a "frozen" status.

The above supervisory-control strategies were vigorously tested. The experiments successfully verified the validity of our proposed methodology.

Conclusion

In this article, we have developed a generalized methodology for the implementation of a supervisory-control system that has the capability to generate control strategies in an on-line and automatic manner. The strategies are sub-sets of a comprehensive DES strategy and they are downloaded to the PLC according to events occurring in the workcell. The experimental results for a two-machine workcell has demonstrated that the proposed general methodology using PLC technology is a realistic and practical means for implementing DES supervisory-control strategies.

For the on-line generation of control strategies in real-time, a hybrid controller such as the one presented in [13] should be considered. This specific controller has the capability to generate limited-size control strategies in a step-by-step manner in response to the occurrence of each event in the workcell. Operating in this mode, the possibility of deadlock arises, but the controller has the capability of dealing with direct deadlocks.

Acknowledgments

We gratefully acknowledge the support of the Natural Sciences and Engineering Research Council of Canada, the Allen-Bradley Canada Company, and the Rockwell International Canadian Trust.

Appendix: Automatic Translator Example

The automatic ladder-logic translator developed herein receives a control-strategy statement in the form of a triplet (exit state, transition event, entrance state), and translates it to a ladder-logic source code. The inputs to the translator are (i) a file that contains the DES strategy triplets and (ii) a file which

contains a priori known I/O reference addresses for each event. The output is the ladder-logic code for an Allen-Bradley PLC-5/11 Programmable Controller.

As an example, the translation of the first two events in the state-transition table of the alternate control strategy, as shown in Fig. 8, is presented.

Table 1 is the complete state-transition table for the alternate route. It has eight possible states as indicated by the first line in the table. The events are numbered 31, 30, 32, 33, 21, 20, 22, 23 and correspond to the events α_3, β_3, λ_3, μ_3, α_2, β_2, λ_2, and μ_2. As previously mentioned, the α and μ events are controllable, while the events β and λ are uncontrollable.

Table 1. State-Transition Table of the Alternate Control Strategy

8		
0	31	1
1	30	2
1	32	3
2	21	4
3	33	0
4	20	0
4	22	5
5	23	0

Using Table 1 and the I/O reference-addresses files, the following translated ladder-logic code is obtained, via automatic translation, for the first two events α_3 and β_3:

```
SOR XIC O:002/00 XIO O:002/06 OTL O:001/03 EOR
SOR XIC O:002/00 XIC I:001/00 XIO O:002/06 OTU O:001/03 EOR
SOR XIC O:002/00 XIC I:001/00 XIO O:002/06 OTL O:003/10 EOR
SOR XIC O:002/00 XIC I:001/00 XIO O:002/06 OTL O:002/01 EOR
SOR XIC O:002/00 XIC I:001/00 XIO O:002/06 OTU O:002/00 EOR
SOR XIC O:002/01 XIC I:001/01 XIO O:002/06 OTL O:003/11 EOR
SOR XIC O:002/01 XIC I:001/01 XIO O:002/06 OTL O:002/02 EOR
SOR XIC O:002/01 XIC I:001/01 XIO O:002/06 OTU O:002/01 EOR
```

In the first rung, SOR is the "start of rung" instruction, XIC is the "examine bit on" instruction, XIO is the "examine bit off" instruction, OTL is the "latch output," and EOR is the "end of rung" of rung instruction. The I/O reference addresses are represented by the alpha-numeric sequences beginning with "O" for an output and "I" for an input. The first set of three digits before the slash indicate the data address at the word level, while the last two digits after the slash indicate the data address at the bit level. The first five rungs deal with the first possible event α_3 (a controllable event) and the last three rungs deal with the β_3 event.

The first rung requests for an operation to begin, where the event-request bit at address O:001/03 is enabled. The input conditions of the rung are given by the workcell state (O:002/00) and a general event-request-disabling bit (O:002/06). (This latter input condition is included to ensure that the workcell status does not change while the next set of possible events is being processed. It is a precautionary measure to avoid injury to persons or damage to equipment when a machine failure occurs.) The second rung resets the event-request bit at (O:001/03) using the OTU ("unlatch output") instruction, once a response has been received from the machine for the event-confirmation input bit (I:001/00). The third rung sets the bit at (O:003/10) to register the occurrence of the event. The fourth rung sets the next workcell state bit (O:002/01) "on," and the fifth rung resets the current workcell state (O:002/00) "off." The rungs dealing with β_3 event are similar to those for the α_3 event, except that no rung for an event-request is necessary since β_3 is an uncontrollable event. The ladder-logic code for the other events are also similar to $\alpha 3$ and β_3, depending whether they are controllable or uncontrollable events.

References

[1] P. Ramadge and W.M. Wonham, "The Control of Discrete Event Systems," *Proceedings of the IEEE*, 1989, vol. 77, no. 1, pp. 81-98.

[2] F. DiCesare, G. Harhalakis, J.M. Proth, M. Silva, and F.B. Vernadat, *Practice of Petri Nets in Manufacturing*, Chapman & Hall, New York, 1993.

[3] H. Van Brussel, Y. Peng, and P. Valckenaers, "Modeling Flexible Manufacturing Systems Based on Petri Nets," *CIRP Annals on Manufacturing Technology*, 1993, vol. 42, no. 1, pp. 479-484.

[4] F. Lin, "Analysis and Synthesis of Discrete Event Systems Using Temporal Logic," *Journal of Control-Theory and Advanced Technology*, 1993, vol. 9, no. 1, pp. 341-350.

[5] J.S. Ostroff and W.M. Wonham, "A Framework for Real-Time Discrete-Event Control," *IEEE, Transactions on Automatic Control*, 1990, vol. 35, no. 4, pp. 386-397.

[6] B.A. Brandin, W.M. Wonham, and B. Benhabib, "Discrete Event System Supervisory Control Applied to the Management of Manufacturing Workcells," *Computer-Aided Production Engineering*, V.C. Venkatesh and J.A. McGeough, eds., Elsevier, 1991, pp. 527-536.

[7] M. Zhou and F. DiCesare, *Petri Net Synthesis for Discrete Event Control of Manufacturing Systems*, Kluwer Academic Publishers, Boston, 1993.

[8] S. Balemi, "Discrete Event Systems Control of a Rapid Multiprocessor," *7th IFAC Symposium on Information Control Problems in Manufacturing Technology*, Toronto, Canada, 1992, pp. 53-58.

[9] J.F. Arinez, B. Benhabib, K.C. Smith, and B.A. Brandin, "Design of a PLC-Based Supervisory-Control System for a Manufacturing Workcell," The Canadian High Technology Show and Conference, Toronto, 1993.

[10] M.A. Jafari and T.O. Boucher, " A Rule-Based System for Generating a Ladder Logic Control Program from a High-Level Systems Model," *Journal of Intelligent Manufacturing*, 1994, vol. 5, pp. 103-120.

[11] S. Bhatnagar and R.J. Linn, "Automatic Programmable Logic Controller Program Generator with Standard Interface," *ASME Manufacturing Review*, 1990, vol. 3, no. 2, pp. 98-105.

[12] R. Devanathan, "Computer Aided Design of Relay Ladder Logic via State Transition Diagram," *16th Annual Conference of the IEEE Industrial Electronics Society*, 1990, pp. 527-532.

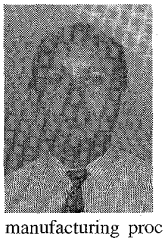
[13] R.A. Williams, B. Benhabib, and K.C. Smith, "A Hybrid Supervisory Control System for Flexible Manufacturing Workcells," *IEEE, International Conference on Robotics and Automation*, San Diego, 1994, pp. 2551-2556.

[14] B.A. Brandin, W.M. Wonham and B. Benhabib, "Manufacturing Cell Supervisory Control: A Modular Timed Discrete-Event System Approach," *IEEE International Conference on Robotics and Automation*, Atlanta, 1993, pp. 846-851.

[15] R.A. Williams, B. Benhabib, and K.C. Smith, "Model-Based Diagnostics for the Supervisory Control of Manufacturing Systems," *AAAI Sympo-*

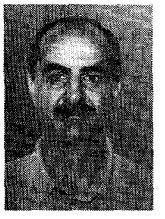
sium on Detecting and Resolving Errors in Manufacturing Systems, Stanford, 1994, pp. 134-139.

Stephane C. Lauzon obtained his B.A.Sc. in mechanical engineering in 1993 from the University of Ottawa and his M.A. Sc., also in mechanical engineering, in 1995 from the University of Toronto. His contributions to this article originated from his M.A. Sc. thesis in the area of DES-Theory-based supervisory control of manufacturing systems. He is currently a systems engineer with Honeywell's Industrial Automation and Control Group. Lauzon is a member of the ASME and the IEEE.



James K. Mills is currently associate professor of mechanical engineering at the University of Toronto. From 1988-1993, he was assistant professor in the same department. Mills has authored or co-authored a number of research papers on various aspects of robot dynamics, stability, and control. His current research interests include the development of multi-robot assembly systems for use in automotive assembly plants, as well as a various applications of control theory to robots and manufacturing processes. He was the recipient in 1990 of the Canadian Society for Mechanical Engineering I.W. Smith Award for Creative Engineering. Mills is a Member of IEEE and the Professional Engineers of Ontario.

Anthony K.L. Ma obtained his B.A. Sc. and M.Eng. in mechanical engineering in 1991 and 1995, respectively, from the University of Toronto. His contributions to this article originated from his M.Eng. project in the area of automatic supervisory control of manufacturing systems.



Beno Benhabib obtained his Ph.D. in mechanical engineering in 1985 from the University of Toronto, where he is currently an associate professor in the Department of Mechanical Engineering. His research interests are in the general area of computer-integrated manufacturing. His work has been published widely on various aspects of robot-motion planning, machine vision, robotic sensors, and supervisor-control of manufacturing systems. He has consulted to various Ontario manufacturers in the areas of CAD/CAM, robotics, and automated quality control. He is a senior member of the SME as well as a member of ASME, IEEE, and AAAI. He is a registered Professional Engineer.

Sampled Data

Computer Users' Ode to Dr. Seuss

Here's an easy game to play.
Here's an easy thing to say.

If a packet hits a pocket on a socket on a port,
And the bus is interrupted as a very last resort,
And the address of the memory makes your floppy disk abort,
Then the socket packet pocket has an error to report!

If your cursor finds a menu item followed by a dash,
And the double-clicking icon puts your window in the trash,
And your data is corrupted cause the index doesn't hash,
Then your situation's hopeless, and your system's gonna crash.

You can't say this? What a shame, sir!
We'll find you another game, sir.

If the label on the cable on the table at your house,
Says the network is connected to the button on your mouse,
But your packets want to tunnel on another protocol,
That's repeatedly rejected by the printer down the hall,
And your screen is all distorted by the side effects of gauss,
So you icons in the window are as wavy as a souse,
Then you may as well reboot and go out with a bang,
'Cause as sure as I'm a poet, the sucker's gonna hang!

When the copy of your floppy's getting sloppy on the disk,
And the microcode instructions cause unnecessary risc,
Then you have to flash your memory and you'll want to RAM your ROM
Quickly turn off your computer and be sure to tell your mom!

—Anonymous, from the Editor's email