Proceedings of the
44th IEEE Conference on Decision and Control, and
the European Control Conference 2005
Seville, Spain, December 12-15, 2005

A modular control design method for a flexible manufacturing cell including error handling

Seungjoo Lee and Dawn M. Tilbury

Abstract— **This paper introduces a modular design method for a flexible manufacturing cell. First, we provide a definition for a flexible manufacturing cell, and then we propose a design method for its cell controller. We divide the controller into two parts: the resource allocation control and the operation control. Based on this structure, we develop operation blocks integrated with error handling and recovery, and prove some properties about their behaviors. Finally, we introduce a case study and apply the proposed method to this example.**

I. Introduction

A large-volume manufacturing system is normally divided into multiple stages each with manageable size. Each stage is often called a 'cell' in industry and is independently controlled by a cell controller. This cell controller coordinates several process machines and material handlers to bring raw materials closer to the final product through sequences of operations. At the same time, this controller should detect and handle errors.

However, the traditional development time of such a cell controller is too slow to cope with the quick changeover for new products because the traditional development procedure relies heavily on the experience of experts and is very time intensive [6].

To reduce the development time and to remove feasible design mistakes in advance, some integrated software tools [10] have been introduced to design the cell controller. These software tools can generate the executable control code by combining the given specification with designers' knowledge. The correctness of the generated control code can be verified by 3D graphic simulation. However, this type of verification is not complete, and well-disciplined experts are required to generate the control code.

On the other hand, most research efforts in academia have focused only on analysis of control behaviors (mathematical verification) [3,4,9,12] such as sequence deadlock problems. Moreover, error handling and recovery for flexible manufacturing cell controllers have not been investigated rigorously.

One effort to investigate a modular design method with error handling was limited to dedicated systems (e.g., transfer lines) with a fixed product path, and cannot be directly applied to more flexible systems [8]. In this paper, we generalize that method and propose a new cell controller design method applicable to flexible systems. In this new design method, we divide the control functions into two parts: resource allocation control and operation control. To allow communication among two control parts and other system components, we define the required interfaces and develop operation blocks.

This paper is organized as follows. In Section II, we define the flexible manufacturing cell. In Section III, we propose its control framework and discuss the issues with current industrial practice. In Section IV, we develop the modular design method, including operation blocks, and show how this new design method addresses the main problems found in industrial control design. In Section V, we present a case study where the method is applied. Finally, we conclude and discuss future work.

II. Definition of a Flexible Manufacturing Cell

Consider an automated manufacturing system consisting of several cells connected by inter-cell material handlers that move parts from one cell to another. The system level controller controls the inter-cell material handlers and starts sequences of operations by communicating with the cell controllers. We assume that each cell controller is independent of the other cell controllers. A cell controller interacts only with the system level controller when a part enters or exits the cell through an entry or an exit buffer. Under this assumption, we define a Flexible Manufacturing Cell (FMC) as follows:

Definition 1: [FMC] A flexible manufacturing cell (FMC) consists of an independent cell controller and several resources (R). This cell can move and process multiple different parts simultaneously. Resources in the FMC can be grouped into three categories: process machines (M_1, M_2, $\cdots$, M_n), material handlers (H_1, H_2, $\cdots$, H_m), and internal single capacity buffers (B_1, B_2, $\cdots$, B_ℓ).

In the FMC, operations can be performed sequentially or concurrently to change the part closer to the final form. Operations in the FMC are defined as follows:

Definition 2: [Operations] An operation (O_i) performed in the FMC is any one of a process operation (O_i^M), a material handling operation (O_i^H), or a communication operation (O_i^C).

- A process operation (O_i^M) is performed by a process machine (M) to change the part closer to the final form; it requires a process machine resource.
- A material handling operation (O_i^H) moves a part from one location to another. In addition to the material handling resource which performs the operation, two more resources are required: the location that the part leaves ($R^L(O_i^H)$) and the location that the part arrives

This work was supported by NSF under grant EEC95-29125

S. Lee is with the Department of Mechanical Engineering, University of Michigan, Ann Arbor, MI 48109-2125, USA seungjoo@umich.edu

D. M. Tilbury is with the Department of Mechanical Engineering, University of Michigan, Ann Arbor, MI 48109-2125, USA tilbury@umich.com

0-7803-9568-9/05/$20.00 ©2005 IEEE

($R^A(O_i^H)$). These two resources can be process machines (M), internal buffers (B), or entry/exit buffers (B^{entry}/B^{exit}). These two resources can be the same ($R^L(O_i^H) = R^A(O_i^H)$) when a material handling operation reorients a part and then puts it back.
- A communication operation (O_i^C) communicates with the system level controller (SLC); it does not require any resources.

A process or a material handling operation (O^M or O^H) executes the corresponding job program in a process machine (M) or a material handling (H) controller. This job program is written in the language of each piece of equipment (e.g., robot language or NC code). The operation start event signals the job program to start, and the operation finish event is generated when the job program completes sucessfully.

When a part enters the cell, the cell controller needs to know which process operations are required in which order, and what buffers the part will be stored in between process operations. Recognizing that each buffer resource can only hold one part, but that there may be several buffers located near each other, we define a buffer group (B^G) as a set of internal buffers, such as a partitioned shelf or box that can store multiple parts.

Definition 3: [A part flow path] A part flow path (P_i) is a sequentially-ordered set consisting of process operations (O^M), buffer groups (B^G), and entry/exit buffers (B^{entry}/B^{exit}) that indicates how a part is processed in a cell and when a part is stored in a buffer. This part flow path starts with a specific entry buffer (B_j^{entry}), passes through process operations or buffer groups, and ends with a specific exit buffer (B_k^{exit}). In the middle of a path, only process operations or buffer groups are allowed.

If two parts have the same part flow path, we say that they belong to the same part group (P^G).

III. Control Framework and Problem Statements

A. Control Framework

To make the complexity of the FMC controller manageable, we divide its important functions into two parts: the resource allocation control and the operation control. The resource allocation control allocates resources to requesting operations, and then generates operation start events. To accomplish these tasks, the resource allocation control consists of three different categories of rules: part acceptance rules, operation starting rules, and conflict resolution rules. These rules are sequentially executed by considering the current resource states, safety interlock conditions, and sequences of operations.

On the other hand, the operation control interacts with the equipment controllers to execute operations, keeps track of operational control states, and handles errors to resume the production. Once an operation start event is generated by the resource allocation control, the corresponding operation starts the job program residing in an equipment controller. In addition, the operation control manages operational states in both auto and manual modes. Therefore, even during error handling in manual mode, the operational states that are changed by manual recovery steps can be traced. In this paper, we discuss the operation control in detail.

B. Problems with Current Industrial Practice

Cell controllers developed in industrial practice have many potential problems which are difficult to identify without running the system. Among such control problems, five important ones are listed as follows: part flow deadlocks, repeated execution of on-going operations, non-recoverable states, discrepancies between the control states and the current plant states, and non-deterministic behaviors. These undesirable control problems may result in errors, which require unproductive downtime to debug the control code. Therefore, the cell controller developed by the proposed modular design method should avoid such control problems. These problems are described in more detail below.

A part flow deadlock is a permanent blocking of a set of operations, each of which competes for resources. This part flow deadlock can occur when a resource is improperly allocated to one of several simultaneously requesting operations. If no proper recovery action is taken, the system will be blocked permanently. For complex routings, this part flow deadlock cannot easily be detected by intuition.

A repeated execution of an on-going operation can occur when a current working resource is preemptively allocated to another operation by the cell controller.

A non-recoverable state is a state from which the control cannot return to the initial state. Once the control reaches this state, the controller cannot be initialized except by rebooting.

Discrepancies between the control states and the current plant states can occur when the system is manually recovered from an error, but its controller fails to trace the current state correctly [11]. If such a discrepancy occurs; the next behavior of the controller may be unpredictable.

Non-deterministic behaviors can occur when two or more conflicting transition conditions are simultaneously satisfied. In this case, it is unpredictable which transition is fired.

To address these problems, we propose a formal modular control design method.

IV. Proposed Solution

A. Interactions between controllers

Before discussing the control design method, we define the interactions between the two parts of the cell controller (resource allocation control and operation control) and the system level controller. The interactions necessary to process a part in a FMC are defined as follows:

Definition 4: [Interactions among the controllers] Necessary interactions between the operation control and the resource allocation control are as follows:

- The resource allocation control chooses the next operation while avoiding deadlocks and conflicts, and generates an operation start event. When the operation control receives this start event, the corresponding operation in the operation control starts the job program.

- The resource allocation control monitors the operation control states to update the resource states for resource allocation-purposes (Busy, Idle, or Error).

Necessary interactions between the operation control and the system level controller are as follows:

- Communication operations in the operation control generate events to inform the system level controller of the necessary operational states: O_1^C for a part loaded into the cell, O_2^C for exit buffer request, and O_3^C for the part exited from the cell.

Finally, interactions between the resource allocation control and the system level controller are as follows:

- The resource allocation control can generate a part reject event when the part cannot be processed in the cell. When the system level controller receives this reject event, the system level controller handles this rejected part by either re-routing or scrapping it.
- The system level controller generates three events sequentially for each part: a part arrival event (S_1^C), the exit buffer availability event (S_2^C), and a part leave event (S_3^C).
- The process information a part arriving in the cell (e.g., a part flow path) is sent together with the part arrival event (S_1^C) by the system level controller.

For a normal automatic sequence, these interactions are illustrated by a UML activity diagram in Fig. 1. In the normal automatic sequence, the events generated by the system level controller and the operation control are ordered as follows: $S_1^C \cdots O_1^C \cdots O_2^C \cdots\ S_2^C \cdots O_3^C\, S_3^C$.

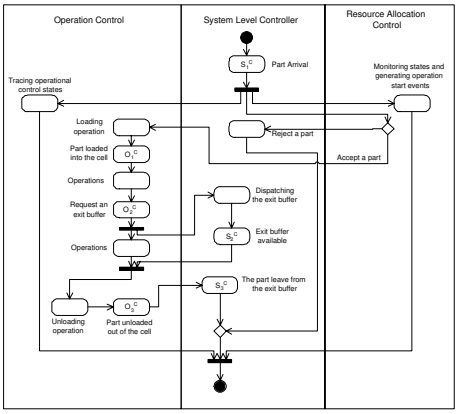

Fig. 1. An activity diagram of UML (Unified Modeling Languages) [2] divided by three *swimlanes*: the operation control, the resource allocation control, and the system level controller. This activity diagram shows a work flow that is comprised of important activities during a normal automatic sequence for a part. The communication operations (O_1^C, O_2^C, O_3^C) generate events to inform the system level controller of the states in the FMC. The resource allocation control monitors operational states from the operation control and generates operation start events. The system level controller generates three events: a part arrival event (S_1^C) with the process information of a part, the exit buffer availability event (S_2^C), and a part leave event (S_3^C). The resource allocation control can generate a part reject event to inform the system level controller that this part cannot be processed in this cell

B. Operation Blocks

In addition to interacting with the system level controller, the flexible manufacturing cell controller must interact with equipment controllers and human operators. During manual error handling, the FMC must keep track of its control state. To illustrate such interactions, an activity diagram is prepared for a process operation as shown in Fig. 2. This diagram shows the interactions between a process operation defined in the operation control and a process machine equipment controller. When an operation start event signal is generated, the current state is changed from the 'IDLE' state to the 'RUN' state. While the corresponding job program is running in the equipment controller, the current state in the operation control remains in the 'RUN' state. If an error occurs during this operation, the current state will be changed to the 'ERROR' state. In the 'ERROR' state, the cause of error is fixed by the operator through manual manipulation. If the job program is finished without any error, the current state will move to the 'DONE' state. After a part is taken out of the process machine, this machine is available for other operations, and the current state is changed from 'DONE' to 'IDLE'. In addition, the 'WAIT' state (reachable only from the 'ERROR' state) represents waiting for a material handler to remove the part from the machine by performing the part scrap operation. Until the part is removed from the machine, this machine cannot be used for other operations.

In the 'ERROR' state, manual error recovery steps are taken after the cause of error is fixed. In this paper, we restrict the number of manual error recovery steps to three: Reset, Scrap, and Retry. The operator pushes a button on the HMI to select the appropriate event ('reset', 'scrap', or 'retry') so that the control state is synchronized with the physical state.

The 'reset' event moves the control state from 'ERROR' to 'IDLE', indicating that the machine is now ready for other operations. The 'scrap' event requests that the part should be scrapped by a material handler. This event moves the control state from 'ERROR' to 'WAIT', and the control state remains in 'WAIT' until the part is removed from the machine by a part scrap operation. The 'retry' event causes the job program to be restarted (although if the true cause of the error has not been fixed, it may stop again). In addition, the 'error' event is generated when either the equipment controller or the cell controller detects an error.

The control states and events in the interaction are influenced by the control mode of the equipment devices. We assume two control modes: auto mode and manual mode. In the auto mode, a normal sequence is executed, while in the manual mode, manual error recovery steps can be taken.

Based on the interactions in Fig. 2 and the control mode, the basic process operation block is developed as in Fig. 3 and Table I. The process operation block is written in SFC (Sequential Function Chart), one of the IEC61131-3 PLC languages [5], and has four steps and eight transitions. Table I shows the allowable events and conditions for each control mode. The material handling operation block and the communication operation block are defined similarly.

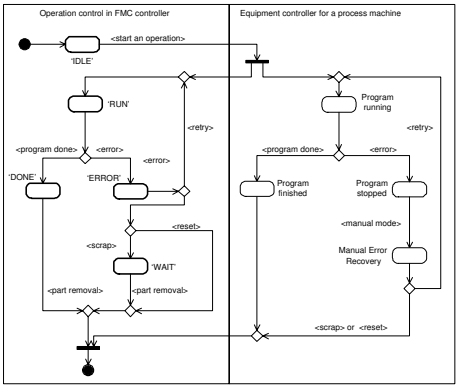

Fig. 2. An activity diagram for a process operation divided by two $swimlanes$: the operation control in the FMC controller and the equipment controller. This diagram shows the interactions between the operation control in FMC controller and a machine equipment controller during a process operation. A process operation can have four states: 'RUN', 'DONE', 'ERROR', and 'WAIT'. Three error recovery steps are allowed and their corresponding events ('reset', 'scrap', and 'retry') are generated by operators. The 'reset' event moves the control state from 'ERROR' to 'IDLE'. The 'scrap' event moves the control state from 'ERROR' to 'WAIT', and the control state remains in 'WAIT' until the part is removed from the machine by a part scrap operation. The 'retry' event causes the job program to be restarted.

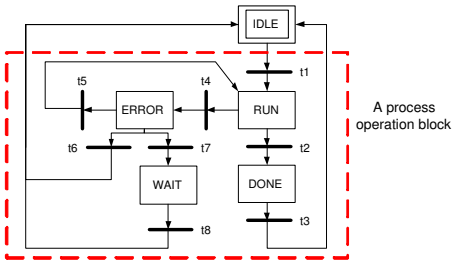

Fig. 3. An 'IDLE' step and a process operation block written in SFC.

Because there are multiple transitions coming out of some places, conflicting transitions can exist in an operation block. When two or more firing conditions of conflicting transitions are satisfied simultaneously, it is not clear which one should be fired. To avoid such non-deterministic behaviors in the operation blocks, we define event priorities as follows:

Assumption 1 (Event priority): The error event has priority over the operation finish event in the 'RUN' step. The manual error recovery events are priority ordered as: 'reset' $>$ 'retry' $>$ 'scrap'. Therefore, in every control execution cycle, only one transition can be fired even if two or more transition conditions are satisfied simultaneously.

Safety interlock conditions protect parts, resources, and/or human operators, but sometimes they are too restrictive and may generate unnecessary blocking situations [11]. In this paper, we assume that no safety interlock conditions cause blocking.

Assumption 2 (Safety Interlock Conditions): We assume that the safety interlock conditions protecting parts, resources, and human operators from damage are proper and do not cause unnecessary blocking situations.

When operational states in a basic operation block need to be further refined, the following restrictions should be imposed to preserve the control properties, as explained in section IV-C.

TABLE I

TRANSITIONS AND STEPS OF A PROCESS OPERATION BLOCK (SIn: SAFETY INTERLOCK n)

Trans.	Auto Mode	Manual Mode
$t1$	operation start $\uparrow$	
$t2$	operation finish $\uparrow$	operation finish $\uparrow$
$t3$	part removal $\uparrow \wedge SI1$	part removal $\uparrow \wedge SI1$
$t4$	error $\uparrow \vee$ timer out $\uparrow$	error $\uparrow$
$t5$		$retry \uparrow \wedge$ SI2
$t6$		$reset \uparrow \wedge$ SI1
$t7$		$scrap \uparrow \wedge$ SI3
$t8$	part removal $\uparrow \wedge$ SI1	part removal $\uparrow \wedge$ SI1

Steps	Description	Actions
$IDLE$	Idle State	'IDLE' state on
RUN	Program Run	Send the program number 'RUN' state on
$DONE$	Program Finished	'DONE' state on
$ERROR$	Error State	'ERROR' state on
$WAIT$	An error recovery operation	'WAIT' state on

Restriction 1 (Restrictions on the refinement): An operation block can be refined as long as the following restrictions are satisfied.

1) Every transition should have a single input step and a single output step (finite state machine).
2) When a refined operation block is connected to the 'IDLE' state, there should exist at least one path from every step in the block to the 'IDLE' state.

Well-developed refined operation blocks can be stored in the library of control design tools and be reused for other applications. Using basic or refined blocks and part flow paths, we can automatically generate the operation control by applying a simple algorithm to instantiate and group related operation blocks. Finally, by combining the operation control with the resource allocation control, we can complete the development of a cell controller for FMC.

C. Proof of control behaviors

In this section, we investigate the behaviors of the cell controller developed by the proposed method. In the operation control, the operation blocks were described by SFC (the sequential function chart). However, since SFC is not a formal language, it does not allow direct mathematical analysis. For analysis purposes, the operation blocks in SFC need to be converted into a formal language such as Petri nets [1]. SFC was developed from Petri nets and is very similar in terms of its appearance and dynamic evolution rules. The mathematical properties of Petri nets can be used to verify and analyze the behaviors of discrete event dynamic systems, and are used here to verify the behaviors of the proposed FMC controller.

For Petri net models of controllers, three important behaviors are typically considered: liveness, safeness, and reversibility. The physical meaning of these control behaviors can be phrased as follows [1]:

- Liveness means the absence of deadlocks. This guarantees that all transitions can eventually be fired.
- Safeness guarantees that there is no attempt to request the execution of any on-going operation.
- Reversibility characterizes the recoverability to the initial state from any state of the system.

The operation control part of the FMC controller is built from operation groups. Each material handler or process machine resource in the cell has an associated operation group that includes the operation blocks for all of the operations that the resource can perform. An initial step ('IDLE') has an outgoing transition to the run state of each operation block; transitions from the operation blocks back to IDLE are as shown in Fig. 3.

To study the behavior of the operation controller, we convert the SFC operation groups to Petri net models. Although the explicit transition conditions cannot be included in the Petri net, they will be considered later. The following proposition shows that each operation group is a strongly connected finite state machine as defined in [1].

Proposition 4.1: The converted Petri net model of each operation group is structurally a strongly connected finite state machine.

Proof: Each operation block converted into a Petri net is a finite state machine since each transition has one input place and one output place (by definition of finite state machine). In addition, from any place in the converted model, there exists a transition firing sequence returning to the initial place ('IDLE'). Therefore, the converted model for each operation group is strongly connected. ⋄

Since a Petri net model for an operation group is a finite state machine and starts with a single token in the initial state ('IDLE'), only one place can hold a token at a time in the group. In addition, each operation group satisfies the following behavioral properties.

Proposition 4.2: Each group is live, safe, and reversible if the resource allocation control behaves correctly as follows:

- Do not attempt the execution of an on-going operation.
- Do not allow the preemption of a resource (A resource in use remains in use.).
- Avoid part flow deadlocks in sequences of operations.

Proof: Suppose the resource allocation control behaves as mentioned. The safeness of each group of operation blocks is guaranteed by the first two behaviors and Proposition 4.1. Any operation in the sequence can start eventually since the resource allocation control doesn't cause any part flow deadlock in sequences of operations. Once an operation starts, this operation is exclusively controlled by operation blocks in the operation control. By assumptions 1 and 2, all the events in an operation block are asynchronously triggered, and no safety interlock conditions cause unnecessary blocking situations. From these two assumptions and proposition 4.1, all transitions can be eventually fired, therefore each group is live. By proposition 4.1, the liveness property is equivalent to the reversibility property [8]. ⋄

Based on these two propositions, the following theorem can be derived for the control behaviors of the FMC controller.

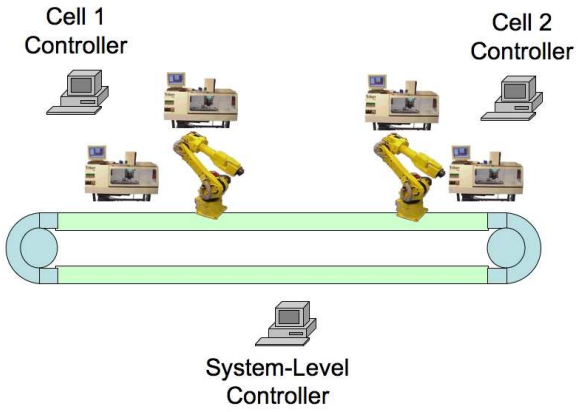

Fig. 4. The Reconfigurable Factory Testbed at the University of Michigan includes two flexible manufacturing cells and a conveyor for inter-cell material handling [7].

Theorem 4.3: The flexible manufacturing cell (FMC) controller is live, safe, and reversible if and only if the resource allocation control behaves as in Proposition 4.2.

Proof: (only if) Suppose the flexible manufacturing cell controller is live, safe, and reversible. It is clear that any violation of the behaviors in Proposition 4.2 leads to the contradiction of this supposition. (if) Each group of operation blocks is live, safe, and reversible by Proposition 4.2. Since each group is safe, the FMC controller is safe. In auto mode, interactions with other groups can occur only through the resource allocation control. While an operation in a group is being performed in auto mode, this group is independent of others until the operation is finished. In manual mode, the operation block may interact with other groups only through safety interlock conditions. By assumption 2, there is no blocking situation caused by safety interlock conditions. Therefore, the FMC controller is live. Also, there's at least one transition firing sequence by which each group can return to its initial place ('IDLE') because each group is reversible and live. Therefore, the FMC controller is safe, live, and reversible. ⋄

V. Case Study

To validate the proposed control design method, a flexible manufacturing cell controller was developed for one of cells in the reconfigurable manufacturing test-bed in the University of Michigan, see Figure 4. Each cell is comprised of two 3-axis CNC horizontal milling machines and a 6-axis robot with a gripper. The cell has a PC-based Siemens PLC and is networked over Profibus. The cell controller not only communicates with an in-house system level controller to exchange events and part information, but also communicates with a Web HMI over OPC standard. Since the automatic program generation software has not been fully developed yet, some control design steps were manually taken.

In an example production scenario, two part flow paths were specified to produce two different part groups: one part flow path was $B^{entry}O_1^M B^{exit}$, and the other was

$B^{entry}O_2^M B^{exit}$. Two part scrap operations were specified by O_5^H and O_6^H to remove damaged parts from the machines to a trash can. Both CNC machines (M_1 and M_2) could perform both operations O_1^M and O_2^M.

The resource allocation control was somewhat trivial in this case because neither the production scenario nor the cell were complex. A newly arrived part was rejected when both machines were out of service or when the robot was out of service. If a part arrived at the entry buffer while both machines were working, this newly arrived part was also rejected because there was no location holding the part temporarily. When both CNC machines were idle, the least-recently-used machine was chosen.

Three operation groups were created for the two CNC machines and the robot. The group of operation blocks for the robot is shown in Fig. 5. This robot could perform six different operations: two loading operations (O_1^H, O_3^H), two unloading operations (O_2^H, O_4^H), and two part scrapping operations (O_5^H, O_6^H). For each operation, a refined operation block was developed and used. In addition, two communication operations (O_1^C and O_3^C) were inserted after loading or unloading operations by following the interactions explained in Section IV-A.

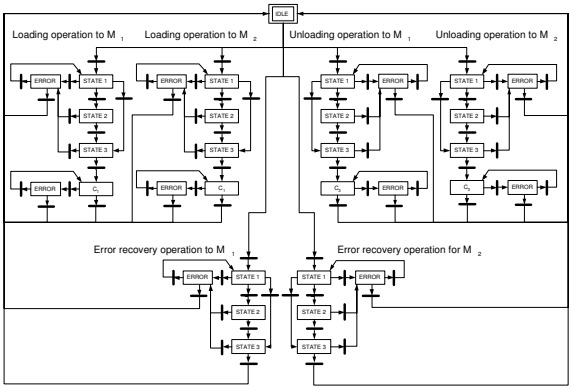

Fig. 5. The robot (H_1) operation group. This robot can perform six different operations for loading, unloading, and error recovery. The communication operation O_1^C immediately follows a loading operation (O_1^H or O_3^H), and the communication operation O_3^C immediately follows an unloading operation (O_2^H or O_4^H). Two part scrapping operations (O_5^H and O_6^H) removed damaged parts from the CNC machines to a trash can.

The operation groups for two CNC machine resources were similarly made. Each operation group consists of two process (machining) operation blocks (O_1^M and O_2^M), with a communication block (O_2^C) after each process operation block.

VI. Conclusion and future work

The main contribution of this paper was to propose a new modular control design method for a flexible manufacturing cell, and to guarantee the correctness of the control under some assumptions. In this new design method, modular operation blocks integrated with error handling were developed. We proved that the resulting FMC controller is live, safe, and reversible if and only if the resource allocation control does not have part flow deadlocks, repeated execution of on-going operations, or preemption. The proposed design method was applied to develop the FMC controller of a test cell.

As our future work, we will extend this design method to cover more complex resource allocation control and to include the integrated HMI control and $RFID$ for part tracking. In addition, we will develop a design and verification method for the safety interlock conditions. Finally, we will propose an automated error recovery framework interacting with human operators.

References

[1] A. A. Desrochers and R. Y. Al-Jaar. *Applications of Petri Nets in Manufacturing Systems : Modeling, Control, and Performance Analysis*. IEEE Press, 1995.

[2] M. Fowler and K. Scott. *UML Distrilled*. Addison-Wesley, 2000.

[3] M. D. Jeng and F. Dicesare. Synthesis using resource control nets for modeling shared-resource systems. *IEEE Transactions on Robotics and Automation*, 11(3):317–327, 1995.

[4] F. L. Lewis, A. Gurel, S. Bogdan, A. Doganalp, and O. Pastravanu. Analysis of deadlock and circular waits using matrix model for flexible manufacturing systems. *Automatica*, 34(9):1083–1100, 1998.

[5] R. W. Lewis. *Programming Industrial Control System using IEC1131-3*. IEE Publication, 1995.

[6] M. R. Lucas and D. M. Tilbury. A study of current logic design practices in the automotive manufacturing industry. *International Journal of Human-Computer Studies*, 59(5):725–753, 2003.

[7] J. Moyne, J. Korsakas, and D. M. Tilbury. Reconfigurable factory testbed (RFT): A distributed testbed for reconfigurable manufacturing systems. In *Proceedings of the Japan–U.S.A. Symposium on Flexible Automation*, Denver, CO, July 2004. American Society of Mechanical Engineers (ASME).

[8] E. Park, D. M. Tilbury, and P. P. Khargonekar. A modeling and analysis methodology for modular logic controller of machining systems using Petri net formalism. *IEEE Transactions on Systems, Man, and Cybernetics*, 31(2):168–188, 2001.

[9] J. Park and S. A. Reveliotis. Deadlock avoidance in sequential resource allocation systems with multiple resource acquisition and flexible routings. *IEEE Transactions on Automatic Control*, 46(10):1572–1583, 2001.

[10] J. Richardsson. A survey of tools and methods for design of automated production plants. In *Proceedings of the 33rd International Symposium on Robotics*, 2002.

[11] J. Richardsson, K. Danielsson, and M. Fabian. Design of control programs for efficient handling of errors in flexible manufacturing cells. In *Proceedings of the 2004 IEEE International Conference on Robotics and Automation*, pages 2273 – 2278, 2004.

[12] M. C. Zhou, F. Dicesare, and A. A. Desrochers. A hybrid methodology for synthesis of Petri net models for manufacturing systems. *IEEE Transactions on Robotics and Automation*, 8(3):350–361, 1992.