Journal of the Korean
OR/ MS Society
Vol. 20, No. 2, August 1995

Design and Specification of a Low-Level Control Software for an FMC Using Supervisory Control Theory

Sangkyun Kim* · Jonghun Park** · Namkyu Park*** · Jin Woo Park*

Abstract

Supervisory control is an approach based on formal language. It is used to model and control discrete event systems in which each discrete event process is represented as an automaton. A supervisor is a generator that switches control patterns in such a way that a given discrete event process behaves in obedience to various constraints. A flexible manufacturing cell (FMC) is one of discrete event systems. Functions necessary for the operation of an FMC are characterized by operational components and informational compomens. The operational components can be modeled using the finite state machines and the informational components can be modeled using the abstract formalism which describes supporting operations of the cell controller.

In this paper, we addressed functions required for FMC control specification, software engineering aspects on FMC control based on supervisory control, a concept of event queue for resolving synchronization problem, and complexity reduction. Based on the mathematical model of an FMC, we synthesized the controller by integrating a supervisor for FMC with control specification that specifies event-driven operation of the cell controller. The proposed control scheme is stable mathematically so that the system always behaves on a controlled way even under the existence of uncontrollable events. Furthermore, using an event queue concept, we can solve a synchronization problem caused by the violation of instantaneity assumption of supervisory control theory in real life situation. And also, we can propotype a control software rapidly due to the modularity of the proposed control scheme.

* Dept. of Industrial Engineering, Seoul National University
** Engineering Research Center for Advanced Control and Instrumentation
*** School of Industrial Engineering, Purdue University

1. Introduction

A flexible manufacturing system(FMS) is defined as an integrated manufacturing system composed of several flexible numerically controlled(NC) machines and automated material handling systems such as automated guided vehicles(AGV) and automatic storage/retrieval systems(AS/RS) under computer control. In general, the flexibility of FMSs can lead to the benefits such as increased productivity, reduced inventory, reduced production cost and quality improvement. But these virtues come to reality only through the fullest exploitation of the potential flexibility residing in FMS. Here comes the importance of the computer control systems. Control aspects of FMS can be subdivided into two parts : one for system operation optimization and the other for control software generation. While there are numerous literature concerning operation optimization, relatively few research efforts exist in relation to software development process. Representative research efforts aiming to tackle such software engineering aspect of the control are the conceptual framework proposed by Naylor et al. [13], object-oriented representation of a cell controller for rapid prototyping of control software [10] and the automatic generation of control software based on the formal model of the system [12]. In those researches, manufacturing systems are modeled using formal specification method such as object hierarchy or state transition diagrams to ease the implementation and maintenance of the control software.

In our research we intend to tackle the problem of designing and specifying control software for FMC. In what follows we consider the FMC control problem as an event dispatching problem based on the state transition model of a cell coupled with a cell supervisor. To be rigorous with the proposed control scheme, the cell should be represented with sufficient formality in the sense that such concepts as synchronization and mutual exclusion are formally treated. We selected the finite state automata as the prime modeling tool because of its simplicity and comprehensiveness. Moreover there exist well developed theories on supervisor synthesis for such systems [16-19].

Today the main trend in structuring automated manufacturing systems is to expand gradually from an FMC level to a full-fledged FMS level [1]. This was made possible due to the advancement of the computer communication technology and standardization efforts by the industry. In particular, for small to medium enterprises pursuing CIM, this gradual build up is the most suitable approach from both economical and technological points of view.

This research is motivated by the expectation that the gradualism would persist : i. e. we first concentrate on rapid design of a control software for a single cell and then implement it to other cells with a minimum effort as new cells are added. Assuming that the behavior of the target FMC is deterministic, our research objectives can be summarized as follows : 1) Formal representation of the FMC and its control requirements using a finite automata model, 2) Synthesis of a supervisor satisfying control requirements and guaranteeing some control invariance(e. g. deadlock free), and 3) Proposing a control software architecture based on the concept of cell supervisor for real-time event dispatching in FMC.

In the following section 2, background of this research is outlined. In section 3, the supervisory control approach to developing control software for FMC is given. A software architecture for the proposed control scheme is presented in section 4-1 and an example problem is illustrated in section 4-5. And the concluding remarks and directions for further researches are given in section 5.

2. Background of the research

2-1. Relevant previous researches

Kim [7,8] claims that automata-theoretic models be the most appropriate models for the mathematical representation of automated manufacturing systems due to the state transition property of such systems. Mettala et al. [12] proposed a CASE tool, CIMGEN, for automatic generation of control software which uses the concept of system state space graph.

One important variant of the automata-based models is a Petri-net model. Its language complexity as well as algebraic complexity is higher than that of finite state machines [5]. The lack of satisfactory verification method for data structures and liveness, and the difficulty of analyzing reachability graphs in the case of infinite states, however, limit the applicability of Petri-net models [15].

In the meantime, control community developed systematic theories for the control of discrete event systems such as manufacturing systems, communication systems and database systems, by characterizing the asynchronous, nondeterministic and discrete natures of such systems. Main objective of the theories in control community is to determine the qualitative structural features of the control problems in the discrete event systems [16]. Informally, these research directions are called supervisory control. The name, " supervisor", comes from the name of the automaton that provides an appropriate control pattern according to the state transition of the target system. One of such theories under the framework of Ramadge and Wonham [16] constitutes the theoretical background of this research.

Maimon and Tadmor [11] applied supervisory control to low-level control of FMS. They represented control requirements using forbidden string and temporal logic, and converted them to automata, thus enlarging the expressive power of their model. And then, they reduced the system model and the supervisor at each step of the controller synthesis to overcome the complexity problem. Unfortunately Maimon et al.'s research lacks many extensions to the basic framework of Ramadge et al. For example, modular synthesis [19], decentralized control [9], partial-information .system control [3], real-time control using temporal logic [14] and real-time control using clock-automaton [2] can provide a wealth of modeling tools and controller design methods for manufacturing engineers. Our research includes a control software framework based on a supervisor and its implementation-related issues.

2-2. Supervisory control theory

In this section, the main theme of the supervisory control scheme presented by Ramadge et al. [16] will be described.

Supervisory control is one of the approaches to model and control discrete event systems. It is based on a formal language theory. The theory is elegant and is independent of the models used for applications. In most applications, each discrete event process is assumed to be modeled by an automaton or a state machine, and its behavior is also assumed to be completely described by the language generated by the automaton. All system requirements or specifications are also assumed to be specified as languages. Therefore, a design problem for supervisory controllers of discrete event systems can be stated as follows : find an automaton which is a supervisory controller such that the combined automaton for the controlled system generates the specified language. A supervisory controller controls a discrete event system by enabling or disabling controllable events. Many interesting results have been reported on controllability, observability and modular synthesis [20].

In their original paper, the automaton describing each discrete event process is called a generator and a generator that accepts external control is called a controlled discrete event process(CDEP). A supervisor is defined as a generator that switches control patterns in such a way that a given CDEP behaves in obedience to various constraints. When a CDEP and a supervisor are coupled together, the system is called a supervised discrete event process (SDEP) and the language generated by the SDEP represents the possible behavior of the controlled system. Therefore, the objective of the supervisory control problem is the synthesis of a supervisor that, when coupled with a CDEP, generates a desirable language. In fact, this synthesis is accomplished through the search of supremal controllable sublanguage of the SDEP [16].

3. Specification of the FMC control software

In this research, we intend to construct an FMC control software based on formal state transition model of a plant. For this intention, we analyzed the functional requirements of the FMC control. We identified, from these requirements, suitable functions for modeling using state transition net. The other functions are integrated into the control software around the state transition model. We will apply the techniques of Ramadge and Wonham when we extract the state transition model of the cell controller from the system state transition model.

3. 1 Functional requirements of the FMC control

A manufacturing cell concept was first introduced in Norway [1]. Under this concept, the manufacturing cell is composed of several CNC machines arranged in a circle around a single robot. Today, the definition of FMC varies widely among industries and companies. For example, the machining cell of ASRI(Automation and Systems Research Institute) FMS located at SNU(Seoul National University) in Korea consists of one CNC-lathe and one CNC-machining center connected via one AGV. But, viewing in terms of machine tools connectivity through automatic carriers, conveyors, or robots without AGVs, there are many FMCs in literature with a tightly coupled configuration as well.

In general, functions necessary for the operation of an FMC can be categorized into three sub-functions. First, ***the cell operation control function*** monitors and regulates the various kinds of discrete events occurring in the cell to accomplish the design objectives of the cell. This can be thought of as the "physical" part of the cell controller. But, for these operational control to be exercised properly, necessary ***informational support*** should be provided, which constitutes the "informational" part of the cell controller. This is the second function. Moreover, modern automated manufacturing control systems should be equipped with a ***facility for communicating with other parts of the system.*** This is the third function of the FMC controller. We classified three sub-functions into two separate modeling components of the cell controller : the first(operation control) function as an "***operational***" control component of the cell control model, and the second and the third(data management and communication control) functions as an *"informational"* component of the cell control model. We will describe the operational part of the model using the finite state machine and the informational part of the model using the abstract formalism which specifies supporting operations of the cell controller according to the behavior of the operational part. Our modeling scheme is illustrated in the Figure 1.

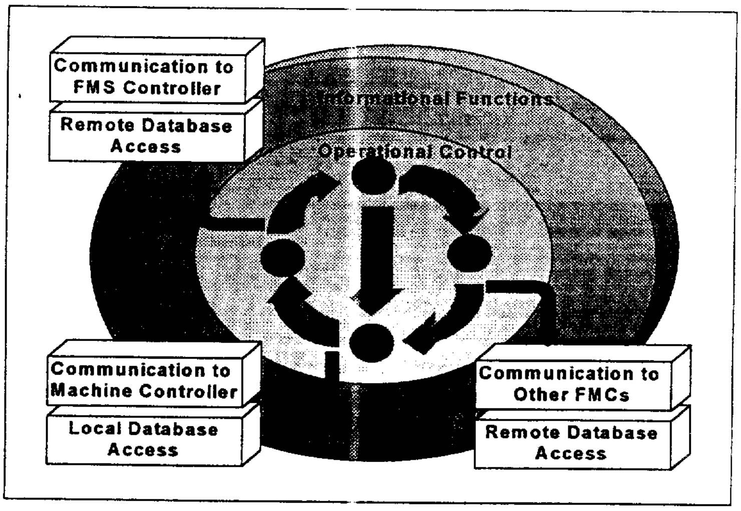

Figure1. Modeling Scheme for FMC Controller

Detailed description of the modeling scheme follows in the next sections.

3-2. Formal specification of an FMC control software

In this section, we develop a formal specification method for the cell control software.

In general, software design problem entails structured analysis and design of systems requirements. Various documents are produced such as data flow diagrams, entity-relationship diagrams, etc. by structured analysis and design. Those diagrams can be classified into data model, functional model, or dynamic model according to their characteristics. Under object-oriented paradigm the above models are presented in the form of an object model. An object model describes objects and their relationships which constitute the target software system. Several design methodologies exist which represent different set of names, symbols, diagrams and procedures for producing the models mentioned above. Each methodology has their own methods for identifying data objects, relationships, and procedures of the models. Our design scheme is primarily targeted at defining the *operational model* of the cell control software which is ready for being translated to a programming language and has a rapid prototyping property. To reach this goal, we concentrated on the ***dynamic model*** of the cell and defined the ***data structures and procedures*** of the system object model based on data and communication requests of the dynamic model. If these object models are implemented in the form of a library of reusable component modules, we can prototype the cell controllers rapidly. This will be explained further in the last section. Our scheme can be viewed as another method in terms of the above statements. Because we are able to build the dynamic model of a target system to describe the discrete operations of the system. And then, we can identify the entities and procedures for defining the object model of the manufacturing system using this dynamic model.

In general, the data generated, updated and maintained in the cell are related to some events occurring in the cell. For example, status of each cell component, production information and resource status should vary according to the occurrences of some appropriate events. Similarly, communications between the cell computer and the low-level equipments or the high-level host are initiated via occurrences of some events. For example, alarms for the occurrences of machine failure or part program upload/download are initiated by the occurrences of machine failure or part input events, respectively. We elaborate on this relationship between operational events and the corresponding data management/communication tasks for specifying the requirements of the control software for an FMC.

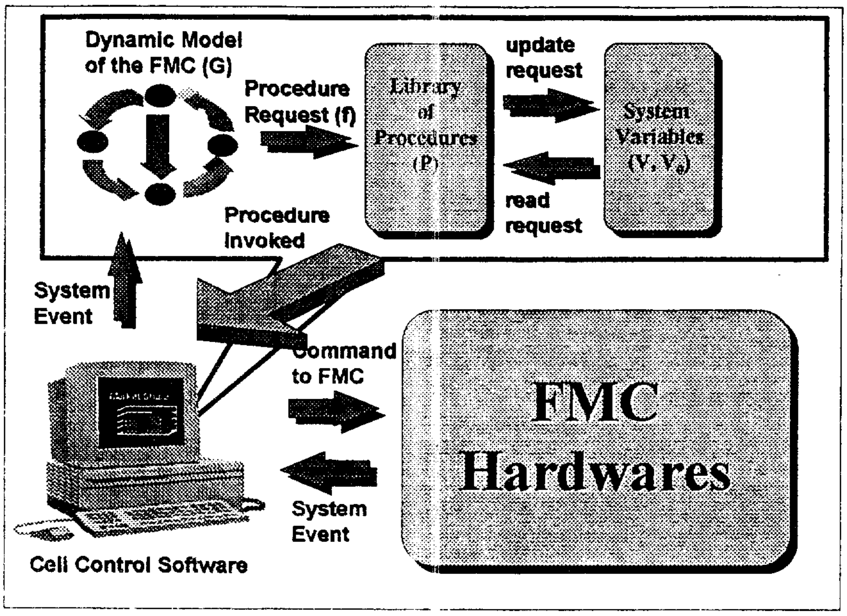

Figure 2. Conceptual View of a Cell Control Software

A cell control software can be viewed as a program which performs various managerial operations on the system variables in the occurrences of system events(Figure 2). We can define this relationship in a 5-tuple :

$$CS = (G, V, V_0, P, f)$$

where CS is a control specification, G is a Mealy machine that has an identity output function (Mealy machine is a finite automaton that associates an output symbol with each state transition [4].), V is a set of system variables, V_0 is the initial values of the system variables in V, P is a set of procedures expressed in terms of the variables in V and f is a set function(f:$E(G)\rightarrow P$) defined as $f(\sigma)=\{p\in P \mid p$ performs the functions required by the occurrence of the event $\sigma\in E(G)$, where $E(G)$ is the set of event symbols occurring in $G\}$. In terms of our previous explanation G will represent the *dynamic model* of operations of the controlled cell, where "controlled cell" means the coupled system of a cell supervisor and physical cell components. Originally, we were interested in the way cell supervisor (G) is constructed. Cell supervisor is an agent which monitors the discrete events occurring in the cell and operates appropriate control actions into the cell. It is modeled as a finite state automaton synthesized from the finite state machine model of the cell and control constraints. The cell model is constructed from finite state machine models of the cell components through the ***shuffling operation*** of automata theory [4]. But the cell model in itself causes unwanted behavior of the cell components. We impose control constraints on the cell model to resolve this phenomena. Currently the control constraints regarding *deadlock prevention, buffer overflow/underflow prevention and obedience to a given process plan* are implemented. After constraining the behavior of the cell model, we obtain the so called supervisor which, when coupled with the cell model, guarantees desirable operations of the cell by regulating the occurrences of system events. Detailed synthesizing process is described in [16]. So we can obtain the dynamic model of the controlled cell, G, through a mathematically proven way. G will determine the trajectory of the system behavior by regulating the occurrences of events. The other components of the CS (V, V_0, P, f) are related to the description of data and communication requests of the cell control function. V is a set of variables which can be a column name of database table, important global variables, communication ports, etc. Thus V can be referred to as a data dictionary for describing informational part of the cell controller. Initial values of the variables are defined in a separate set V_0. Using these system variables we can express the procedures required to handle the data management or communication requests. P is a set of such procedures required to embody the controller, and each procedure in P carries out some operation such as inserting a data recc rd into database table, setting a register to a specified value, initiating machine operation, or :nvoking a packet analyzer. Thus by ^xecuting certain procedures in a specific order we can acl ieve required handling function for each event occurring in the cell. The mapping f is a procec ure request for each event occurred as shown in Figure 2.

Therefore, if all the components of the CS str icture are specified completely, we have a formal specification model of the cell control softv are. This is due to the fact that CS can be interpreted as a program that executes a set of operations upon system variables according to the occurrences of events which is guided by ε cell supervisor (G), and by generalizing the system variable concept the communication need; of the controller can also be handled in addition to the data management tasks. This stri cture will ease implementation process of the cell control software. Moreover, by incorporatin; the framework of supervisory control theory into our scheme, we can guarantee control ii variance in the sense of the Ramadge and Wonham [16].

In the next section we will present our contr ol software architecture. Then we will give an automaton model of an example FMC for showin; the supervisor synthesis process.

4. Software engineering aspects on the proposed FMC control specification

In this section, we propose our control softwai e architecture and building steps based on the cell supervisor synthesized through the procedur described i the previous section(detailed procedure will be illustrated in an example to be shown later in this section). Then event synchronization and complexity reduction probleı ıs are resolved within the context of software implementation. A simple example closes the sect ion.

4-1. FMC control software archite ture

Figure 3 shows the architecture of the prop ›sed cell controller. Our controller consists of four modules(Event Monitor, Cell Supervisor, Event Dispatcher, Data and Communication Manager) and local database not shown in the :igure. The four modules work together to implement the semantic operations of the CS as lescribed in the previous section. First, Event Monitor receives several types of events(σ) oc urred from the FMC. The sequence of events are determined by the operations of the Cell Supervisor and Event Dispatcher. When a new event occurrence is detected, the mapping f of CS is invoked to perform necessary operations (e. g. database transactions, communications with other devices) using handlers in Data and Communication Manager. After those operations are performed, the Supervisor receives the event names (ω) to transit to the new state. These event names(ω) are not always the same as the event name just detected (σ). This is due to the synchronization problem between actual occurrences of events and the logical processing of the events in the controller, which is treated in section 4-3. Supervisor now produces a new control pattern(ϕ) corresponding to a new state. This pattern specifies allowable events to occur next time. Event Dispatcher schedules the next event (σ') to occur. No matter what selection rules are used, from a random rule to a simulation-based rule, the system operations are not influenced. Uncontrollable events such as machine breakdown are not considered since they are not forced to occur but only detected from the FMC. Event Dispatcher then executes appropriate commands lying in Data and Communication Manager to enable that event. When this event (σ') is completed, it is detected by sensors and fed into the Event Monitor to signal the beginning of a new cycle. As explained later, we need to reconstruct the Supervisor while the cell is operating. Modules for reconstructing supervisors are also stored within the Data and Communication Manager and initiated by the occurrence of a new part input event. This new supervisor just replaces the old one and all others remain intact.

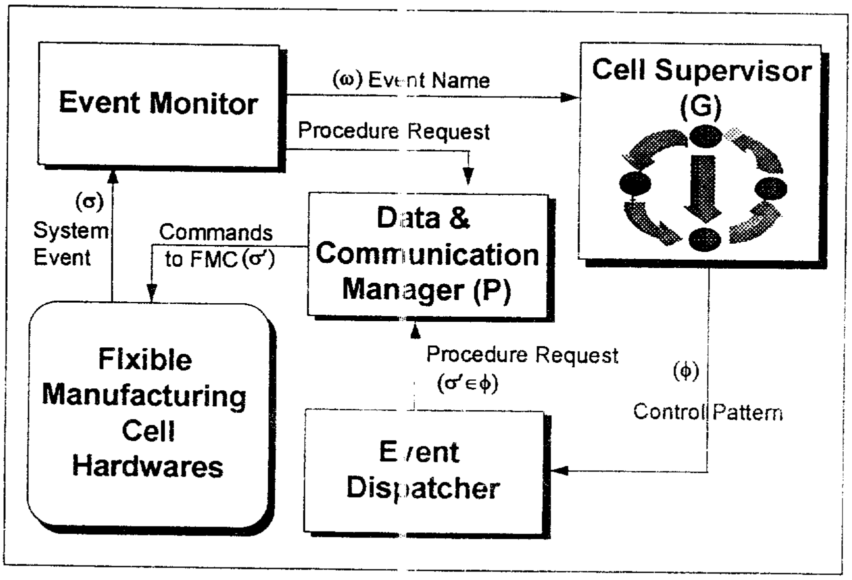

Figure 3. The Architecture of the Control Software

4-2. Steps to designing FMC control software

Careful examination of the controller architecture presented in the previous section reveals that it is a software duplication of CS discussed in section 3-2. From this fact, CS can be viewed as an intermediary between the formal model of FMC and the software realization of a controller. Indeed, CS specifies the operation of a controller in response to the occurrences of system events.

We can characterize the controller building processes by three stages as follows.

(i) Model Building

An automata corresponding to cell components are defined and then aggregated cell model and supervisor are constructed from that automata.

(ii) CS Definition

Based on the events identified during the model building process, components of CS(V, V_0, P, f) can be defined by system experts. For each event, related system variables(including DB table fields, global variables, etc.) and required operations(DB transaction, communication, operator alarm, etc.) are specified. This is a kind of software specification and makes it easy to develop the final software.

(iii) Construction of Data and Communication Manager, Event Monitor, and Event Dispatcher

Data and Communication Manager is a library of software modules corresponding to the procedures in P of CS. Event Monitor is a module that monitors the occurrences of events and calls the modules in Data and Communication Manager by applying the mapping f of CS to the detected event. Finally, we need Event Dispatcher to select the next event.

By the discrete nature of FMC, event-driven control is appropriate for it and our automata and CS structure helps to implement event-driven programs. By following the proposed scheme, we can expect several benefits in relation to software implementation as follows:

(i) Modular Structure of Controller

Software component such as Event Dispatcher and Event Monitor can be replaced by any module irrespective of the internal logic if the module has the same interface as the original one. Handlers in Data and Communication Manager can also be updated, appended or deleted as modification is necessary, with small impact on other components.

(ii) Rapid Implementation of Controller

Our procedure is additive: i. e., as we proceed with the procedure, software requirements are added to the basic cell model of the system. So several logical problems related to system behavior are resolved in the earlier stages of our procedure. In cases where the model is not suitable, model reconstruction is performed easily because only the problematic component is required to be changed. After then, it is integrated with other components mechanically. These overall properties shorten the period for control software development.

4-3. Event synchronization

One of the assumptions of the supervisory control theory is that events occur instantaneously. Therefore, the discussion in section 4-1 describes the operation of the controller exactly if the occurrences of events are instantaneous. In real situations, however, some events are mechanical movement of physical objects and this consumes time in order of magnitude from a few seconds to several minutes. Thus the trajectories of multiple events could be interwoven: for example, even if the part loading command (σ') to machine B was issued earlier, the completion event (σ) of machining at machine A may occur while part loading operation onto machine B is still doing. This can cause an inconsistent transition of the supervisor in the sense that the supervisor is fed with an event $\sigma\sigma'$ sequenc $\sigma\sigma'$ are not allowed under the current state.

Thus some synchronization mechanism is needed to handle time consuming events consistently with the supervisor(which was synthesized without timing consideration). We adopt an event queue to manage this problem. This queue contains event names corresponding to either the events enabled by Event Dispatcher or the uncontrollable events detected from the FMC. Basically events in this queue are processed using FIFO (First-In-First-Out) rule.

The procedure for event queue management is listed below:

(i) Wait for an event signal. If a signal arrives, go to (ii).

(ii) If the signal is issued by Event Dispatcher indicating the start of some event, add it to the event queue and go to (i).

If the signal is indicating the occurrence of an uncontrollable event detected from the plant, add it to the event queue. If this is the only event in queue go to (iii); otherwise go to (i).

If the signal is indicating the end of an event issued by Event Dispatcher, go to (iii).

(iii) Search the event queue to find an event name corresponding to the received signal.

If it is the first event of the queue, process the events in queue sequentially until an event issued by the Event Dispatcher but not finished yet is encountered or the event queue is empty. Event processing means executing routines in Data and Communication Manager and transiting Cell Supervisor. If the found event is not the first event in the queue, set the event as finished and go to (i).

By managing the event queue as above, we can guarantee synchronization of event occurrences and supervisor transition behavior. This is crucial to the generation of appropriate control patterns reflecting correct status of the cell.

4-4. Complexity reduction

In general, state machine approach is not accepted as a good modeling tool to discrete event system because of the state explosion problem for complex systems [20]. We also found, while solving example problems, that our current approach has some complexity problems. For example, while we were synthesizing a cell supervisor it was necessary to construct an automaton that describes the possible flow patterns of multiple parts existing concurrently in a cell. In this case, if we denote the number of part types by n, the maximum allowable number of parts in the cell by m and the minimum number of operations for the parts by p, then the number of states of the required automaton will be O(mnp). This causes severe complexity problem in the supervisor synthesis process. In this section we propose a scheme to overcome this complexity problem in part.

The key idea to our scheme is very clear and straightforward: we reconstruct small supervisors repeatedly. The idea follows from the fact that parts not within the cell are excluded from the control envelope. Thus we first construct a supervisor incorporating only those parts residing in the cell. When a new part enters the cell, we synthesize a new supervisor incorporating that part, the parts that have left the cell are, of course, discarded from the supervisor synthesis process.

A formal procedure for implementing that scheme is:

(i) Make a transition corresponding to part input for each component automaton.

(ii) Assuming the current state as the initial state, take trim components of automata[16] for parts in the cell.

(iii) Make an automaton for the newly input part.

(iv) Make an automaton for part flow pattern by shuffling the automata generated in (ii) and (iii)

(v) Synthesize a supervisor using the cell automaton which already exists and the automaton made in (iv).

As is apparent from the procedure, we need to maintain component automata as well as a supervisor. But these storage burden is outweighed by the reduction in computational complexity.

4-5. Example : Operational control of a machining FMC at ASRI

We confined our control scheme to satisfy the requirements of

(i) deadlock prevention,

(ii) buffer overflow/underflow prevention and

(iii) obedience to process plan.

While buffer capacity management and obedience to process plan is basically required, deadlock prevention is not essential. There are two different approaches to deadlock prevention : one is a deadlock detection and resolution, the other is a deadlock avoidance. We simply adopted the deadlock avoidance approach for illustrating our approach in the following example.

The ASRI FMS is a model plant constructed within the ASRI(Automation and Systems Research Institute) located within Seoul National University, Seoul, Korea. ASRI FMS consists of 4 flexible manufacturing cells with one loading/unloading station; flexible machining cell, flexible assembly cell, material handling cell composed of 1 AGV and AS/RS in 16×5 racks, and a vision cell. We apply the proposed controller design scheme to one of these cells; flexible machining cell in Figure 4. It is composed of one turning center consisting of a CNC-lathe with a material handling robot transferring parts between CNC-lathe and I/O buffer, and one machining center composed of one horizontal type machining center with an APC(Automatic Pallet Changer).

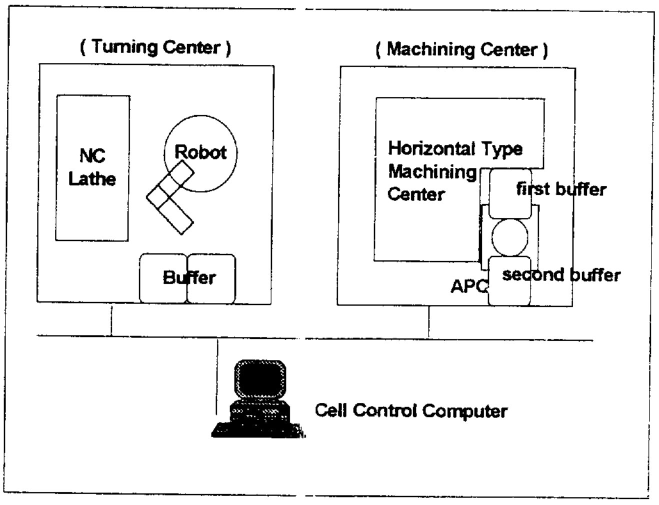

Figure 4. ASRI Machining FMC

First, we modeled each component of the machining cell as a finite state automaton. Figure 5 shows an automaton for the machining cente . Meanings of each symbol are also shown in the figure. Similarly Figure 6 shows an automaton for the APC. APC can be regarded as a rotating buffer of capacity 2. If we impose the constraint that requires the machining center to be in a non-working state when parts are not prepared, we can obtain a legal behavior of the machining center as shown in Table 1(The states in Table 1 are formed by the Cartesian product of the sets of states of each automaton). A legal behavior is physically possible and does not violate given control constraints. In this case, we did not take a process plan into consideration as explained below. This table can be obtained by intersecting the language generated by the automaton for machining cente and the language by the automaton for APC [4]. The effect of this process is to manage buffer capacity since the automaton for APC prevents events from occurring which cause buffer overflow/underflow, the automaton generated by the intersection operation also has the same property. In the table each cell (i, j) of the matrix, corresponding to the i-th row and j-th column of the matrix, represents state transition of an automaton from the state i to the state j. The content of the cell is the event which causes the transition. However, the automaton allows yet an undesirable behavior. For example, the automaton in table 1 cannot guarantee input parts will follow suitable routing sequences within the cell. This guidance of part flow is given by a process plan automaton. In this example, for simplicity, we excluded a process plan consideration from the model which describes the possible part flow pattern within the cell. But, it is a simple matter to adjoin process plans to the current model. All the work required for adjoining are just a "shuffling operation". An example which includes a process plan constraint can be found in [6]. The last step of our approach is to find a supremal controllable sublanguage of the system by applying the algorithm of Wonham et al. [18]. This supremal controllable sublanguage (Table 1) guarantees that the legality will be preserved in spite of the existence of uncontrollable events. As mentioned in earlier section 2-2. this supremal controllable sublanguage defines a supervisor. In this case, the supremal controllable sublanguage generated by the automaton in Table 1 is just the language generated by the automaton in Table 1 because the language is controllable in itself. More abstract and reduced supervisor can be constructed from the generated supervisor using the method in [17].

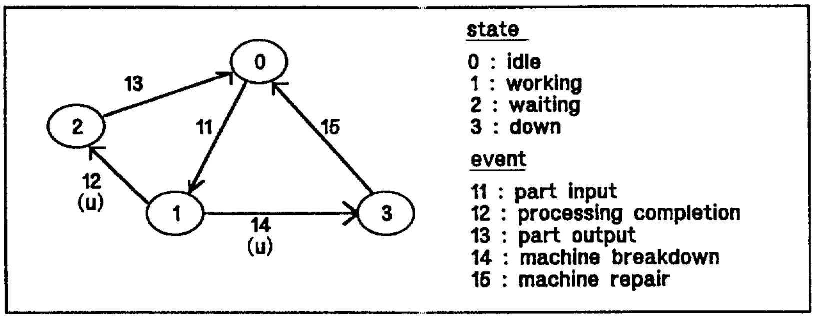

Figure 5. Automaton of ASRI FMS machining center
((u) represents uncontrollable event)

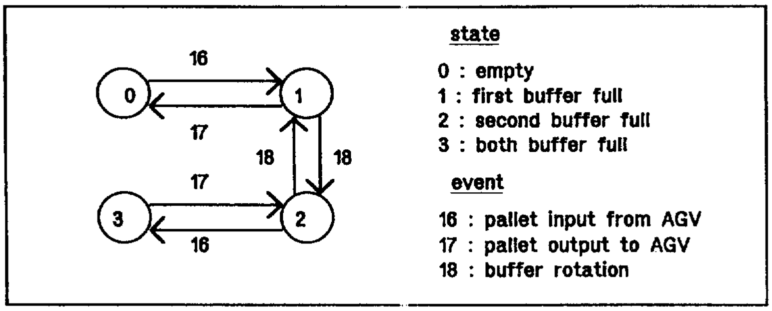

Figure 6. Automaton of ASRI FMS machining center buffer

Table 1. Legal behavior of machining center(without process plan)
(= Supremal controllable sublanguage for machining center)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0					16											
1																
2																
3	15							16								
4	17								18							
5																
6																
7				17	15							18				
8					18					11			16			
9											12	14		16		
10									13						16	
11								18	15							16
12									17					11		
13										17					12	14
14											17		13			
15												17	15			

If we continue in this way, we can also obtain the supremal controllable sublanguage for the turning center(Table 2).

Currently the turning center and machining center of ASRI FMS does not have any interaction. So the supervisor for the total machining center can be constructed from the component supervisors for each workstation. This is a kind of decentralized control [9]. All the results of this section were generated using the software modules implemented in C programming language based on the above algorithms.

Table 2. Supremal controllable sublanguage for turning center

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
0					7				9							
1																
2																
3	6							7				9				
4	8					0,1							9			
5							2	5						9		
6					3,4										9	
7				8	6											9
8	10									0,1			7			
9										2	5		7			
10									3,4						7	
11				10					6							7
12					10				8					0,1		
13						10				8					2	5
14							10				8		3,4			
15								10				8	6			

5. Conclusion

A new architecture for FMC control software was proposed in this research. The purposes of this research are three-folded. First, the proposed controller should reflect the discrete nature of cell operation. Secondly, it should be stable mathematically; i. e., the controller should be synthesized based on the mathematical model of the target system and should assure a kind of control invariance under the existence of some uncontrollable events. These two objectives were resolved by the utilization of the supervisory control theory under the framework of Ramadge and Wonham. Lastly, the proposed control scheme should be easily transformed to software. To meet this requirement, we introduced the CS structure that aims to assist in specifying event-driven operation of a cell controller. By integrating supervisor with CS, we could obtain a specification of control software for an FMC. As a result, we can prototype an event-driven control software rapidly.

In adopting the supervisory control scheme, we are faced with complexity problem. As a partial solution to the complexity problem encountered in the supervisor synthesis process, we developed an adaptive supervisor concept in which cell supervisors are reconstructed according to variations of the system status. By using this scheme, we can reduce much computational cost. In addition to the complexity problem, there is also a synchronization problem when using the supervisory control theory. This could be attributed to the instantaneity assumption of the theory. We resolved this problem by introducing the notion of event queue. An event queue processes events in a manner consistent with the transition property of supervisor. By utilizing all these concepts, we proposed a controller architecture that can be constructed in a modular fashion.

Since that we are confronted the problem of expanding our control scheme to FMS (at ASRI) level where the time bound of an event is longer than that of FMC level, our future research will go toward concentrating on the development of a real-time controller based on the temporal logic to manage the timing problem.

Another possible area for expanding our current research is applying object-oriented paradigm (OOP) in the modeling and implementation of the proposed control scheme. Application of OOP is expected to be very promising because our controller has the modular structure.

Acknowledgement

This research is funded in part by ERC-ACI by Korea Science and Engineering Foundation at Seoul National University.

References

[1] Anuar, Z. and B. Mohamad, "A Hierarchica Distributed Micro-Computer Control System for Controlling a Flexible Manufacturing Cell," *Proceedings of the ICCIM*, (1991), pp. 299-302.

[2] Brave, Y. and M. Heymann, "Formulation and Control of Real Time Discrete Event System," *Proceedings of the 27th IEEE Co* ference on Decision and Control, (1988), pp. 1131-1132.

[3] Cieslak, R., C. Desclaux, A. Fawaz and I . Varaiya, "Supervisory Control of Discrete Event Processes with Partial Observations, *IEEE Transactions on Automatic Control*, Vol. 33(1988), pp. 249-260.

[4] Hopcroft, J. E. and J. D. Ullman, *Introd ıction to Automata Theory, Languages and Computation*, Addison-Wesley Pub. Co., Read ng, MA., 1979.

[5] Inan, K. and P. Varaiya, "Finitely Recursiv Process Models for Discrete Event Systems," *IEEE Transaction on Automatic Control*, Vol. 33(1988), pp. 629-639.

[6] Kim, S., "A Supervisory Control Approach for Rapid Implementation of Low-Level Control Software for FMC," Master's Thesis, Dept. Industrial Eng., Seoul Nat'l. Univ (1993).

[7] Kim, S. H., "An Automata-Theoretic Fran ework for Intelligent Systems," *Robotics and Computer Integrated Manufacturing*, Vol. 5(1989), pp. 43-51.

[8] Kim, S. H. and N. P. Suh, "Mathematica Foundations for Manufacturing," *Journal of Engineering for Industry* Vol. 109(1987), pp. 213-218.

[9] Lin, F. and W. M. Wonham, "Decentralize l Supervisory Control of Discrete Event System," Systems Control Group Report No. 8612, Dept. Electrical Eng., Univ. of Toronto (1986).

[10] Maimon, O. and E. L. Fisher, "An Object-B ısed Representation Method for a Manufacturing Cell Controller," *Artificial Intelligence in Engineering*, Vol. 3(1988), pp. 2-11.

[11] Maimon, O. and G. Tadmor, "odel-Based Low-Level Control in Flexible Manufacturing Systems," *Robotics and Computer Integrated Manufacturing*, Vol. 4(1988).

[12] Mettala, E. G., S. B. Joshi, and R. A. Wysk, "CIMGEN-A CASE Tool for CIM Development," *Proceedings of the Third ORSA/TIMS Conference on FMS*, (1989), pp. 239-246.

[13] Naylor, A. W. and R. A. Volz, "Design of Integrated Manufacturing System Control Software," *IEEE Transaction on Systems Man and Cybernetics*, *Vol.* 17 (1987), pp. 881-897.

[14] Ostroff, J. S., "Synthesis of Controllers for Real-Time Discrete Event Systems,"

Proceedings of the 28th *IEEE Conference on Decision and Control*, (1989).

[15] Ostroff, J. S. and W. M. Wonham, "A Framework for Real-Time Discrete Event Control," *IEEE Transaction on Automatic Control*, Vol. 35(1990).

[16] Ramadge, P. J. and W. M. Wonham, "Supervisory Control of a Class of Discrete Event Processes," *SIAM Journal of Control and Optimization*, Vol. 25(1987), pp. 206-230.

[17] Vaz, A. F. and W. M. Wonham, "On Supervisor Reduction in Discrete Event Systems," *International Journal of Control*, Vol. 44(1986), pp. 475-491.

[18] Wonham, W. M. and P. J. Ramadge, "On the Supremal Controllable Sublanguage of a Given Language," *SIAM Journal of Control and Optimization*, Vol. 25(1987), pp. 637-659.

[19] Wonham, W. M. and P. J. Ramadge, "Modular Supervisory Control of Discrete Event Systems," *Math. Contr., Signals, and Syst.* Vol. 1(1988), pp. 13-30.

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