'95 KACC (1995. 10. 23 ~ 25)

An Object-Oriented approach to the cell activity representation for an Intelligent Manufacturing System (IMS)

Kyunghyun Choi*, Kyu-Kab Cho**

* Research Institute of Mechanical Technology, Pusan National University, Kumjeong Ku, Pusan 609-735, KOREA
Tel: +82-51-510-3057; FAX: +82-51-512-9875

** Department of Industrial Engineering, Pusan National University, Kumjeong Ku, Pusan, 609-735, KOREA
Tel: +82-51-510-2418; FAX: +82-51-512-7603

Abstracts A new methodology for representing the interaction between machines and the interlock signals required in FMCs has been developed. Object-Oriented Philosophies (OOPs) lend themselves to the development of such a scheme. A methodology developed here regards the tasks to be performed by the cell or any of its constituent machines for being primal. Sensory signals indicating the changes of state of machines are used to trigger or initiate tasks. A task may be simple and require a relatively short time to execute, or may be complex and lengthy. This methodology may be depicted by a set of diagrams called Task Initiation Diagram (TID) and their accompanying rules.

Keywords Flexible Manufacturing, Intelligent Manufacturing Systes, Object-Oriented Paradigms, Virtual Cell Control.

1. Introduction

As product life cycles reduce, manufacturing organizations need to become more responsive and can increasingly be considered to represent dynamic system subject to frequently changing requirements. The effort to response to these demands bring into focus the development of manufacturing cells with high levels of flexibility, Flexible Manufacturing Cells (FMCs). Since the main advantage of FMCs is that they are flexible and can therefore be scheduled to manufacture different products or can be easily reprogrammed to handle changes in product design, FMCs may be considered as basic units to compose an Intelligent Manufacturing Systems. The representation scheme of cell activity as a decision-making logic structure is essential to control FMCs.

The vast majority of these approaches are based on Petri Nets, extended Petri Nets, timed Petri Nets [1,2,3], or E-Nets [4]. These allow the behavior of the system to be represented as event graphs for which rules governing operation of the system can be derived and verified. They prove to be a powerful tool in the operation stage where the structure of manufacturing environment is variant, however, would require physical and logical mapping when a change in the structure of the manufacturing environment occurs.

Other approaches utilizing knowledge bases and concepts of artificial intelligence have also been proposed [5].

Lately object-oriented approaches have emerged. The reported approaches were very closely geared to a specific application,e.g. tool loading in an FMC [6], or mould filling [7], and would require a great deal of knowledge about Object-Oriented Programming (OOP) to set up an FMS or FMC.

In these approaches the cell or machine state is primary. The state of the cell or one of its constituents changes as result of the firing of an event. This approach is not suitable for control or verification of FMCs and FMSs.

The approach reported in this paper uses object-oriented concepts to develop the new methodology, called Task Initiation Diagram (TID), for representing cell activities.

2. Virtual Cell Controller Operation

In the object-oriented approach, any entity involved in the cell control is considered as an object. Every object has a unique object identifier. To describe an object and objects in object-oriented programming languages, four general terms are frequently employed: abstraction, encapsulation, inheritance, and polymorphism. Each object is unique. However, abstraction removes certain distinctions so commonalities between objects are recognized. Without abstraction, only differences between objects can be descended. With abstraction, it is easy to selectively omit certain distinguishing features of one or more objects, allowing one to concentrate on the features they share. Encapsulation refers to the act of capturing the state and behavior of an object entirely within the object. An object should be a black box so that users do not have, and do not need access to the internal details or construction of an object.

Inheritance is the ability of objects to obtain information about their internal states and behaviors from more abstract ancestors. Object-Oriented languages contain a mechanism that allows inheritance between classes. Inheritance involves a *superclass* and *subclass(es)*. Superclass is a term that refers to the next most abstract object in the ancestral hierarchy. Subclass is a term that refers to the class that inherits from the more abstract class. For instance, classes Milling Machine, Lathe, and Drilling Machine are subclass of class NC machine. Polymorphism is the capability for different objects to respond to the same meaning of a message. Polymorphism allows the same message selector to be used by different classes for different methods.

Cell operation involves tasks that are performed on single machines independent of others, and tasks that require the cooperation of two or more machines. Where that task calls for the coordination of two or more machines, the cell controller has to be involved to ensure proper execution of the task. For tasks involving a single machine, the controller's main task is to schedule the start of the task, and waits for its completion in

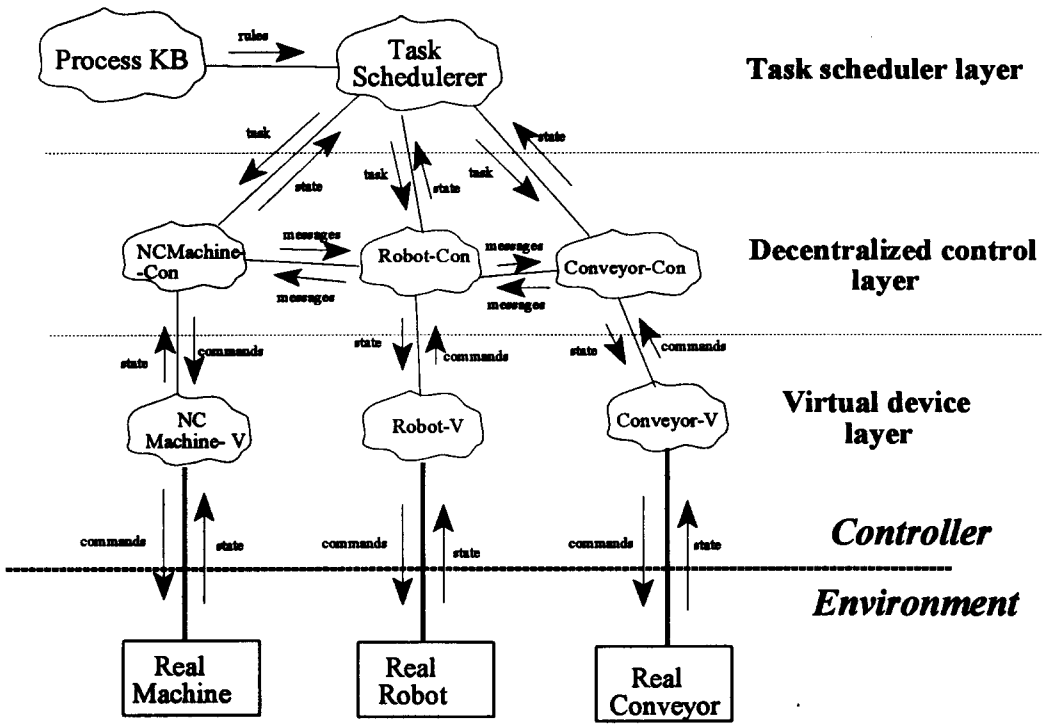

Fig. 1. Control Structure of the Virtual Cell Controllert

order to command the nest task.

In order to accomplish both functions, the Virtual Cell Controller is designed by Object-Oriented paradigm as a hybrid structure of both hierarchal controller and decentralised controllers as shown in Fig. 1. The cell controller consists of three different layers. The Task Scheduler, the Decentralized Control layer, and the Virtual Device layer. In the figure, dark lines indicate physical connections and light lines indicate logical connections. Information and message passing are indicated by arrows.

The Task Scheduler is a core component which receives the states of all the machines in the cell from the Decentralized Control layer, and decides the next task. It then dispatches the nest task to be executed to the Decentralized Control layer. It uses the process knowledge bases that contain the routine cell task rules which are generated from the Task Initiation Diagram (TID) which will be addressed in next section. As has been discussed earlier, these rules are in the form of IF-THEN statements. The arguments of the IF part indicate the cell states that must prevail in order for a task to be started. The arguments for the THEN part contains the required task. It is an inference engine that will select the required rules to handle the cell operation based on a forward chaining reasoning procedure. The cell state describes, at any instance in time, is given by the individual states of the cell constituents obtained from the Virtual Devices.

The Decentralized Control layer consists of Virtual Controllers for the physical machines. Their main role is to perform the harmonization and the cooperation between the cell components in order to carry out the task called for by the Task Scheduler layer. They provide a device independent interface to the actual cell components by translating the generic commands and error messages of the corresponding machine. The Virtual Controllers in the layer communicate and pass messages with each others. Each is correlated to a Virtual Devices. A Virtual Controller sends commands to the dedicated Virtual Device and has no connections with other corresponding physical machine, and receives the state of that machine, through that machine's Virtual Device in the Virtual Device layer. If a give task includes an interaction between machines, the Virtual Controllers concerned exchange the necessary information to coordinate the interaction between the machines. These controllers utilize the Operation Initation Diagram (OID), which will be described in next section, to carry out their function.

The lowermost layer of the controller consists of the Virtual Devices which monitor and continuously mirror, in real time, the state of the physical machine they represent. Each machine's state is analysed by its Virtual Device and reported to the corresponding Virtual Controller as required. The Virtual Devices also act as conduits for commands from the Virtual Controllers to the physical machines.

If the cell does not experience machine or tool failures, the cell control will not need to intervene to stop the production cycle, and to carry out error recovery procedures. In practice such situations are very likely to occur during cell operation. While error detection and handling can be tailored for different cells by implementing fault sensors and declaring them during the cell setup stage, errors can also be detected when a device or machine fails to report operation completion after a specified time. This time is a function of the expected duration for performing the task. The error is detected by the Virtual Controllers. The current mechanism implemented for dealing with errors, call on the Virtual Controllers to immediately abort

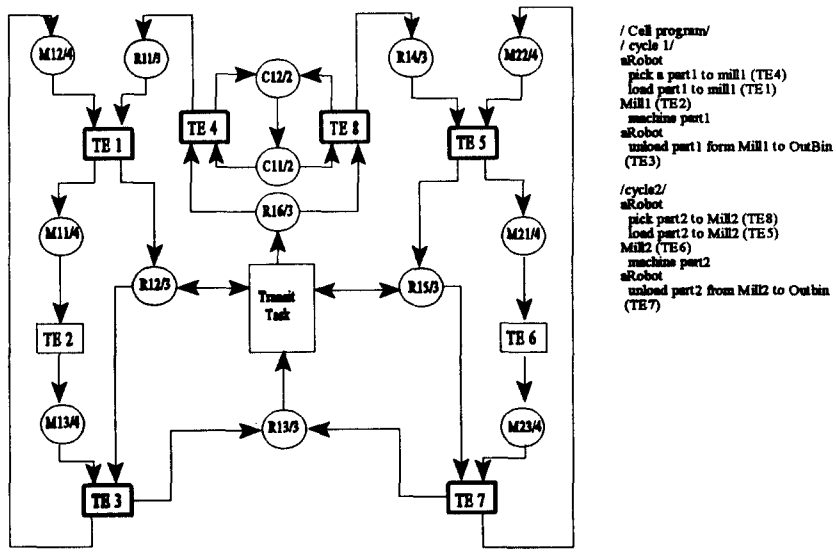

Fig. 2. Task Initiation Diagram and Cell Program

operation of the cell, and to report to the topmost layer in the hierarchy, the Cell Scheduler, for action.

The error knowledge that is contained in the Task Scheduler layer, includes different rules for dealing with different problems that occur during cell operation. These rules are contained in workstation, cutting tool, and cell operation knowledge databases. The rules consist of antecedent and consequence parts, which have the form of IF-THEN clauses. The problem solving system needs to diagnose the cause of the breakdown when it occurs. All the facts are related to the conclusions of the rules (the THEN part of the rules). This system tries to find out the causes of the breakdown through these conclusions and to correct the causes. Backward chaining is used to search rules to solve the problem. The error state stores the status of workstations, robot, and part. After receiving the status of the cell, the status of the error state will be updated.

3. Cell Operation Model

The system does not have any task programmed because there is no logic to guide the succession of events. In this section, cell operation model including the cell process knowledge is addressed. Cell processes are represented by a Task Initiation Diagram using an object-oriented approach. An automatic system has been developed to construct rules to control the sequence of events in a given manufacturing cell. The operation model developed by this automatic system can be simulated and evaluated before implementation.

Many of the approaches that have been developed so far, and reported in the literature, to represent the cell activities are adaptations of ones used in digital logic or electric circuits. The most common of these are petri nets and their derivatives. In these approaches, the cell or machine states are considered primal. The state of the cell or one of its constituents changes as a result of the firing of an event. This approach is not suitable for control or verification of flexible manufacturing cells or systems. For such systems, the task to be performed, rather than the state, must be regarded as primal. A task is executed when a particular set of conditions is met.

The methodology developed regards the tasks to be performed by the cell or any of its constituent machines for being primal. Sensory signals indicating the change of state of machines are used to trigger or initiate tasks. A task may be simple and require a relatively short time to execute, or may be complex and lengthy. The methodology developed here may be depicted by a set of diagrams called "Task Initiation Diagrams" and their accompanying rules

3.1 Conventions and Terminology used for the Task Initiation Diagram

The task initiation diagrams are multi-layered. The topmost layer shows the dynamic behaviour of the task-level program of the cell. Fig. 2 shows an example of a Task Initiation Diagram and corresponding cell program. For conciseness, states and tasks are cryptic instead of verbose. The task initiation diagram consists of two components: tasks and states. Tasks in the task initiation diagram would generally consist of a concerted group of subtasks or operations involving more than one constituent of the cell, and in such a case are termed composite tasks. These are shown by the framed boxes in the diagram. As an example **TE1** (load part1 to Mill1) involves concerted operations involving the Machine and the Robot. Tasks involving a single machine are called simple tasks and are shown by a box, e.g., **TE2** (machine part).

The cell state is given at any instant by the collection of states of its constituents. For initiation of a given task, a composite state consisting of a subset of the cell state needs to

exist. These composite states are shown in the Task Initiation Diagram by ellipses, e.g., *R11/3* or *M13/4*. The last number of the symbols indicates how many individual states are required to determine this composite state. A task level cell program does not include explicit information about the states necessary to initiate the tasks. This is because such information is embedded in the implementation of each task.

3.2 Structure of Task Initiation Diagram

Task Initiation Diagrams are composed of two basic components: a set of Rest states S_R and a set of tasks T. Tasks, in turn, are classified into three groups: the cell configuration dependant task (T_d), the cell configuration independent task (T_i), and the cycle transit task (T_t). T_t tasks are used for the transition from one cycle to another. These are tasks derived automatically by the system in order to complete a production job, thus, a programmer does not need to create them.

To complete the diagram, it is necessary to define the relationship between the states and the tasks. This can be done by specifying two functions connecting states to tasks: the condition function C, and the output function O. The condition function C defines, for each task T_i, the set of states for the task $C(T_i)$. Some condition functions may use guiding parameters in addition to a set of states. As an example, in the listing below, C(TT) uses a Remaining Processing Time (RPT) to cause transition to the desired state. The output function O defines for each Task T_i the set of output States for the transition $O(T_i)$.

These four terms define the structure of a Task Initiation Diagram. Formally, a Task Initiation Diagram TID is defined as the four-tuple TID=(T, S_R, C, O).

Consider the example in figure 2, the components of the TID structure are given below:

TID = (T, S_R, C, O)
T = {TE1, TE2, TE3, TE4, TE5, TE6, TE7, TT_i}
S_R = {Mnm/4, R1k/3}
n =1,2 m=1,2,3 k=1,2,3,4,5,6,7

C(TE1) = {M12/4, R11/3} O(TE1) = {M11/4, R12/3}
C(TE2) = {M14/4} O(TE2) = {M13/4}
C(TE3) = {M13/4, R12/3} O(TE3) = {M12/4, R13/3}
C(TE4) = {C11/2, R16/3} O(TE4) = {C12/2, R11/3}
C(TE5) = {M22/4, R14/3} O(TE5) = {M21/4, R15/3}
C(TE6) = {M21/4} O(TE6) = {M23/4}
C(TE7) = {M23/4, R15/3} O(TE7) = {M22/4, R13/3}
C(TE8) = {C11/2, R16/3} O(TE8) = {C12/2, R14/3}
C(TT1) = {R12/3,M22/4},[RPT(TE2)>Δt] O(TT1) = {R16/3}

4. Conclusions

A new methodology of representing the interaction between machines and interlock signals required in FMCs has been developed. Contrary to the commonly used representation approaches for cell activity which concentrate on the states of the cell constituents and their change of states, the approach TID presented here focuses on the tasks and operations that need to be performed and regards the states of the cell constituents are mere triggers to the cell activities.

Based on a set of process rules derived from the Task Initiation Diagrams and imbedded in the implementation of Virtual Cell Controller, task and operations are coordinated and executed.

REFERENCES

[1] B.Besombers, E.Marcon and A.Alla, "Application of Generalized Bond-Graphs and Continuous Petri-Nets to modeling Industrial Processes and Manufacturing Systems", IFAC-INCOM'92, 1992, pp.617-622.

[2] M.D.Jeng and F.DiCesare, "A review of synthesis techniques for Petri Nets", Proceedings of the second International Conference on CIM, 1990, pp.348-355.

[3] R.Ravichandran and A.K.Chakravarty, "Decision support in Flexible Manufacturing Systems using Timed Petri Nets", Journal of Manufacturing System, 1987, Vol.5, No.2, pp.89-101.

[4] S.Nof, A.Whinston and W.Bullers, "Control and decision support in automatic manufacturing systems", AIIE Transactions, 1980, Vol.12, pp.156-167.

[5] P.J.O'Grady anf K.H.Lee, "An Intelligent Cell Control System for automated Manufacturing", Knowledg- based Systems in Manufacturing (Edited by Kusiak.A,), 1990, pp.151-172.

[6] M.I.Bu-Hulaiga and A.K.Chakravarty, "An Object-Oriented Knowledge representation for hierarchical real time contro of flexible manufacturing", International Journal of the Production Research, 1988, Vol.26, No.5, pp.777-793.

[7] Joannis Robert and Krieger Moshe, "Object-Oriented Approach to the specifcation of Manufacturing Systems", Computer-Integrated Manufacturing Sysytems, 1992, Vol.5, No.2, pp.113-145.