Proceedings of the 1998 IEEE
International Conference on Robotics & Automation
Leuven, Belgium • May 1998

Multi-agent Based Dynamic Scheduling for a Flexible Assembly System

Yung-Yu Chen, Li-Chen Fu and Yu-Chien Chen
Dept. of Computer Science and Information Engineering
National Taiwan University,Taipei,Taiwan,R.O.C

Abstract

This paper proposes a multi-agent based dynamic scheduling approach for a flexible assembly system. We first introduce a flexible control system developed by IntelligentRobotics and Automation Laboratory in National Taiwan University. Based on that control system, the agents can communicate with each other conveniently. A generic agent architecture is proposed to model the pieces of equipment in the flexible assembly system. With a distributed architecture, the agents make their scheduling decisions using their local rule base. The agents acquire the resources following the distributed resource allocation protocol. The scheduling complexity is reduced to meet the real-time response requirement in the applications for flexible automated production. The present work is applied to the experimental robotized flexible assembly system in the above laboratory.

1 Introduction

A scheduler is called dynamic (on-line) if it makes its scheduling decisions at run time on the basis of the current requests for service. Dynamic schedulers are flexible to adapt to an evolving task scenario and have to consider only actual task requests and execution time parameters [2]. Research on multi-agent systems is mainly concerned with how to coordinate intelligent behavioral activities among a collection of autonomous agents [3]. The work in [3] discusses the negotiation among agents for the allocation of resource with time taken into account.

Jennings [4] detailed the design of the multi-agent structure and discusses the principle of handling errors. The work in [5] discusses the rescheduling issue in a decentralized manufacturing system. Manufacturing systems with multi-agent modelling can be found in [6, 7]. Resolution of real-time conflicts in multi-robot systems appears in [8]. The work in [9] proposes some criteria to set up a robotic assembly cell and analyzes the interaction among multiple robots.

Recently, agent-based approach is applied to robotic assembly environment [6, 7, 12, 13, 14]. The work [14] introduces an idea of cooperative action with group organization and a strategy for cooperative task processing using communication. There is an efficient negotiation algorithm using his proposed groupcast communication and a learning mechanism with reference to historical records on the past negotiation.

This paper consists of six sections. Section 2 describes a robotic assembly environment and introduces a general control system for that environment. The scheduling problem to be solved is also defined. Section 3 introduces an *agent architecture* and describes the functionality of the agent's module. Section 4 describes the strategy for resolving resource contention among agents. Section 5 describes the implementation of the dynamic scheduling for the flexible robotic assembly cell. Section 6 describes the experimental result. Finally, some conclusions are made in Section 7.

2 System Description

In our laboratory, we have a two-robot assembly system that is dedicated to assemble various types of mechanical parts sent serially into the conveyor belt by the part loader. There are two products currently assembled in this system, and each product has four parts that are assembled by the robot manipulator. The operations include vertical insertion, horizontal insertion, and rotation in assembling with the subassembly fixed at the assembly sites. The parts are fed into the system without a specific order, and the scheduling is made on-line. We proposed a rule based dynamic scheduling approach to schedule for the whole assembly system [17].

The cell is equipped with several pieces of hardware that work together to assemble parts. The brief description is given below:

0-7803-4300-x-5/98 $10.00 © 1998 IEEE

- **Robot**: There are two robots in the system, namely **ADEPT** and **CRS**. Each of them is equipped with an automatic tool changer(**ATC**) in order to assemle different types of mechanical products. Each robot has a mounted CCD camera which serves the high-precision localization of the part. Also, each robot has its own force-torque sensor that can be used to correct the measurement error during assembly.

- **Part Loader**: It is composed of a Cartesian manipulator and a pallet that holds the parts waiting to be assembled. The Cartesian manipulator will pick up the parts from the pallete and put it on the coveyor belt.

- **CCD Camera**: We have one overhead CCD camera that can determine the type and orientation of the incoming part on the conveyor belt. Two Eye-In-Hand cameras are mounted on the robots, respectively.

- **Conveyor Belt**: It is responsible for carrying parts from outside into the cell.

- **Rotary Buffer**: This is used to temporarily store the incoming parts that are not suitable for immediate assembly. Both robots can access the buffer, but only one robot can be served at one time.

- **Proximity Sensors**: These are to detect the moving speed of each arriving part and act as anchors for the pick-up operation.

Based on the control kernal EMFAK [19], we can easily integrate the different pieces of euipment together. The scheduling problem is to provide a quick response and safety guarantee for the working pieces of equipment. We model each equipment as an agent. An agent has its own capability and thus the scheduling can be viewed as a group of schedulers working simutaneously. This is a distributed system architecture. The coordination among agents is crucial in making the entire system running smoothly.

3 Agent Architecture

There are different kinds of agents in the robotic assembly cell. They can communicate with each other through the communication center *EMFAK* [19]. Fig 1 shows the layout of the multi-agent based system.

Each agent is composed of two main processes which are manager process and controller process. A controller process is related to the domain level system,

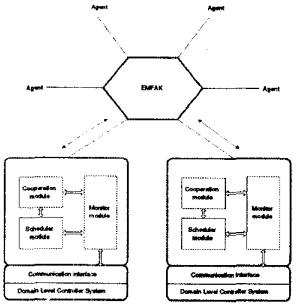

Figure 1: Muti-agent architecture

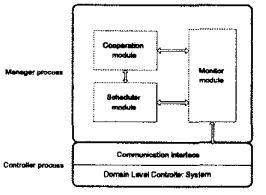

Figure 2: The agent architecture

which includes communication interface module and domain level controller module. Communication interface module is responsible for protocol conversion if controller moudule does not support the pre-selected communication protocol.

On the other hand, manager process is responsible for making the social contact with other agents and scheduling the local tasks. Its objective is to ensure that the agent's domain level activities are coordinated with the other agent and that its associated hardware runs efficiently.

The agent architecture is shown in Fig 2. The components of the generic agent architecture is described in the following sections.

3.1 Cooperation Module

This is an entity that handles the agent-to-agent coordination through message exchange. It is responsible for inter-agent communication. This module has two functional roles:

Message processing Since this module is directly connected to EMFAK, it is responsible for sending message on behalf of the agent and receiving message from the other one.

Coordination In the later section, the distributed resource allocation protocol will be described. Resource contention is the motivation for agent co-

ordination. The agent's scheduler module is concentrated on making its local scheduling and depends on the cooperation module if it is in need of a resource managed by the other agent.

3.2 Scheduler Module

The function role of the scheduler module is to keep the controlled equipment working efficiently, and it will focus on the local affairs. If there is a need of the information or resource from the other agent, it will pass the request to the cooperation module to serve for it.

The agent has a degree of autonomy to make its scheduling decisions. We can view the scheduling problem as a single agent problem virtually. This is a divide-and-conquer approach, and scheduling becomes easier. The scheduler module searches for possible next moves with its local state taken into consideration, and it uses the cooperation module to cooperate with the other agent.

3.3 Monitor Module

This module is to monitor the status of tasks submitted from the scheduler module. It must detect whether the controller module finishes its work on time. Thus, a time-out mechanism is used to assure that the link between two components is active. There are two links in this architecture. They are the link between EMFAK and manager process and the link between manager process and controller process, respectively. For the link between EMFAK and manager process, the manager process periodically informs EMFAK of its liveness. EMFAK keeps a timer that is reset whenever EMFAK receives the liveness message. On the other hand, the other link is treated in another way. With the aid of the monitor module, the scheduler module can make decisions while the underlying domain level system is executing.

3.4 Communication Interface

It serves as the protocol conversion module that can translate different communication media into a preselected protocol. Currently we adopt TCP/IP as our standard protocol [20, 21]. In order to integrate different types of equipment, there should be a selected protocol for communication. If the underlying equipment does not follow that, the communication interface module will be a translator.

3.5 Domain Level Controller System

It is the actual task execution unit that waits for task given from the scheduler and then executes it. For example, they are programs that control robot movement or buffer rotation, etc. Domain level controller system directly controls the physical device. It is the entity that is dedicated to serve the request from the manager process. Basically, it possesses less intelligence and may be seen as a server that receives requests and offers the coresponding service.

4 Distributed Resource Allocation Protocol

In a decentralized environment, the resource allocation problem is complicated. We must consider the mutual exclusion problem when multiple agents want to access to the same resources. There should be a protocol to resolve the conflict on resource. Following the agent's architecture, the resource agent uses its manager process to allocate the resource. For clarity of discussion, the agent that needs the resources is mentioned as *consumer agent*. The consumer agent must get all the permission of all its required resources before utilizing them. Each consumer agent has its own priority that the resource agent uses to decide the precedence of the resource usage.

The consumer agent first sends requests to all relevent resource agents and the resource agent follows a decentralized protocol to allocate the resource. The resource agent has a request list that keeps the incoming request for the consumer agents. This protocol allows the consumer agent to try using mutiple resources at one request. The following sections introduce the distributed resource allocation protocol. The *consumer agent* executes the following protocol:

1. Send request messages to all needed resource agents.
2. Wait for each resource agents to respond with the status of reply.
   (a) If any reply is REJECT, **GOTO** 3.
   (b) If all replies are OK, **GOTO** 5.
3. Send all other resources an REJECTED message. The rejected agent's request will be removed from all resources' lists.
4. **GOTO** 1. (Try again)
5. Inform all resource agents that it is allowed to use the resources.
   (send the ACCEPTED messages to the resource agents)
6. Use the resources.

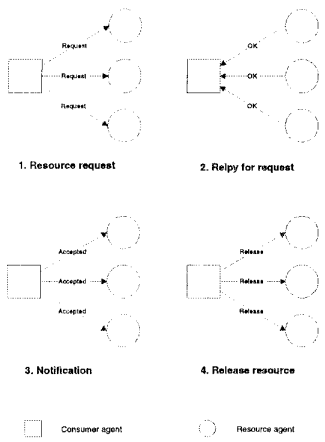

Figure 3: Distributed resource allocation protocol(1)

7. Release all resources.(Send RELEASE messages to resource agents)

The *resource agent* uses the following rules to make decision:

1. Reject all requests(send a REJECT message) when there is a request of higher priority already enqueued in the list or when the resource is *busy*. Otherwise, add the accepted request at the rear of the list.

2. Upon receipt of an REJECTED message, remove the sender's request from the list.

3. Send an OK message to the owner of the first request in the list or the owner of any request that is promoted to the front of the list when the requests in front of it is removed.

4. Upon receipt of an ACCEPTED message, send REJECT messages to the owner of all other requests in the list, empty the list, and mark the resource as *busy*.

5. Upon receipt of a RELEASE message, mark the resource as *free*.

Fig 3 illustrates the case that the consumer agent will get the resources it needs. In the first step, the consumer agent sends requests to all relevant resource agents. The resource agents all reply with OK messages to the consumer agent in the second step. The consumer agent notices the resource agents that it will use the resources in the third step. Finally, the consumer agent free its resources and send RELEASE message to the resource agents.

The protocol is flexible because it operates in a distributed manner. The new agents, whether consumer agents or resource agents, can be added into the system dynamically without interfering the running of the system. For one paticular agent, the arrival order for resource requesting messages does not affect the correctness of the protocol. Also, the message sendings of different consumer agents can be mixed. These above two characteristics can make the system have less assumption about the quality of the communication media and requirement of the communication protocol.

This protocol makes the resource allocation efficient. The resource is allocated to the consumer agent that owns the permissions of all its required resources. Once the resource is released, the contention for them starts again. With this way, resources are kept as busy as possible. The deadlock-free guarantee is proved. This property is important since it is concerned with safety issue and continuous running of the whole system.

5 Dynamic Scheduling in the Experimental FAS

Our experimental environment is a two-robot assembly cell which is dedicated to assembling various types of mechanical parts serially sent in through a conveyor belt. The cell is composed of several pieces of hardware.

Each product has four parts respectively. The first product is assembled with only vertical insertion operations. The second product includes more complex operations. To assemble the second part with the base part for the second product, the robot needs to do vertical insertion and then a rotation to fasten the part with the base part. Sixteen parts can be placed randomly in the pallete of the loader. The loader will load the part onto the conveyor belt one at a time on request.

We model our cell as multiple agents that work together. The modelling of the cell is depicted in Fig 4.

5.1 Error Recovery

Sometimes, there are machine failures during the assembly process. The proposed agent architecture can solve some of the failures. The manager process is assumed to be more reliable. The controller process is the main component that causes error for an agent.

Fig 5 shows the error recovery processs. There are two communication links in the manager process. The

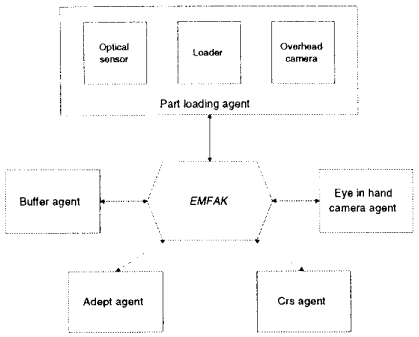

Figure 4: The agents in the FAS

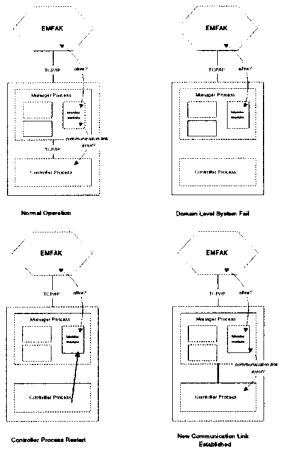

Figure 5: Error recovery for equipment failure

monitor module detects the liveness of the link between the manager process and the controller process. When the physical equipment is faulty, we just shutdown the machine and kill its associated controller process. The manager process is still alive and will wait for the reconnection of the controller process once the physical equipment is recovered. The agent's internal state is kept and the agent restarts from the last kept status.

6 Experiment

The simulation is performed on the pseudo robotic assembly cell. We replace the physical equipment with software. In other words, the agents can be easily modified to work in the real environment. In the simulation, part loading machine loads part into the assembly cell randomly, and the robot will either assemble it or store it on buffer. There are total seven types of parts that may be fed into the cell. removed. The parameters that we will examine are the robot utilization, the histogram of the finished product, and buffer utilization. In this simulation, we give *Adept* a high priority than *CRS*.

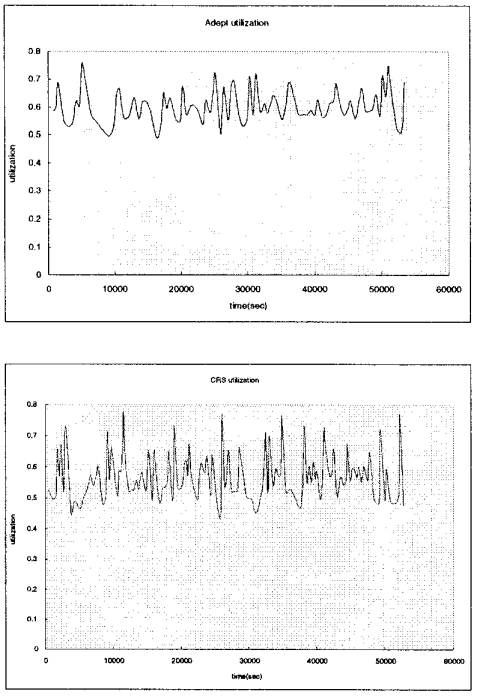

Figure 6: The utilization of the robots

6.1 Simulation result

We assume the robot operation time ranges from 2 to 12 seconds in this simulation, and the time depends on the complexity of that operation. The average operation time is less than 10 seconds.

In the two robot simulation environment, the utilization of the robot is between 0.5 and 0.8 or so as shown in Fig 6. There are resource contention between the two robots. For example, the robots compete for the common workspace, for the shared buffer, and for the shared *Eye-In-Hand* camera PC. Instead, if only one robot is in operation, the utilization of the robot will be above 0.8. The scheduling time is thus 1 to 2 seconds for each robot operation.

7 Conclusion

This paper is an extension of [17]. The relationship between the agents and communication backbone EMFAK is discussed in section 2. To meet the real-time

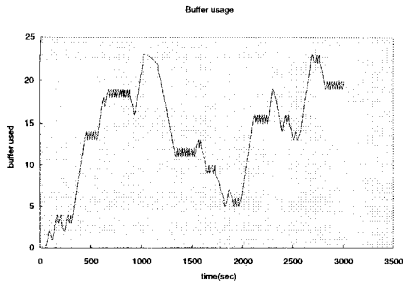

Figure 7: The buffer utilization

requirement in the robotic assembly environment, we use a decentralized approach to solve the scheduling problem. The basic idea is using the concept of agent which is an autonomous entity that can communicate with each other to achieve a coorinated behaviour. Section 3 introduces the agent architecture and the functionality of the agent's module. Section 3 also introduces the distributed resource allocation protocol that is used to allocate the resources among a group of distributed agents. The benefits of using multi-agent based modelling exhibits flexibility, robustness, and modularity. Section 4 describes the implementation of the agents for the flexible assembly cell. The experimental result is given in section 5.

References

[1] M. P. Groover, *Automation, Production Systems, and Computer Integrated Manufacturing.* Prentice-Hall International, 1987.

[2] H. Kopetz, *Scheduling*, ch. 18, pp. 491–509. Addison-Wesley publishing company, 1993.

[3] S. Karus, J. Wilkenfeld, and G. Zlotkin, "Multiagent negotiation under time constraints," *Artificial Intelligence*, vol. 75, pp. 297–345, 1995.

[4] N.R.Jennings, "Controlling cooperative problem solving in industrial multi-agent systems using joint intentions," *Artificial Intelligence*, vol. 75, pp. 195–240, 1995.

[5] T. K.Tsukada and K. G.Shin, "Priam: Polite rescheduler for intelligent automated manufacturing," *IEEE Transaction on Robotics and Automation*, vol. 12, no. 2, pp. 235–245, 1996.

[6] E. Oliveria, "Cooperative multi-agent system for an assembly robotics cell," *Robotics and Computer-Intergrated Manufacturing*, vol. 11, no. 4, pp. 311–317, 1994.

[7] R. J. Rabello and L.M.Camarinha-Matos, "Negotiation in multi-agent based dynamic scheduling," *Robotics and Computer-Intergrated Manufacturing*, vol. 11, no. 4, pp. 303–309, 1994.

[8] G. Cohen, "Concurrent system to resolve real-time conflicts in multi-robot systems," *Robotics and Computer-Intergrated Manufacturing*, vol. 8, no. 2, pp. 169–175, 1995.

[9] P. M. Pelagagge, G. Cardarelli, and M. Palumbo, "Some criteria to help the experimental setup of assembly cells with cooperating robots," *Robotics and Computer-Intergrated Manufacturing*, vol. 12, no. 2, 1996.

[10] Georgeff, "Communication and interaction in multi-agent planning," *Proceedings AAAI-83*, pp. 125–129, 1983.

[11] F.-Y. Wang and G. N. Saridis, "A coordination theory for intelligent machines," *Automatica*, pp. 833–844, 1990.

[12] J. S. Barsran, E. M. Petriu, and D. C. Petriu, "Flexible agent-based robotic assembly cell," *Proceedings IEEE International Conference on Robotics and Automation*, pp. 3461–3466, 1997.

[13] A. A. Rizzi, J. Gowdy, and R. L. Hollis, "Agile assembly architecture: An agent based approach to modular precision assembly systems," *Proceedings IEEE International Conference on Robotics and Automation*, pp. 1511–1516, 1997.

[14] H. ASAMA, K. OZAKI, Y. ISHIDA, K. YOKOTA, A. MATSUMOTO, H. KAETSU, and I. ENDO, "Collaborative team organization using communication in a decentralized robotic system," *Intelligent Robots and Systems*, pp. 816–823, 1994.

[15] R. G. SMITH, "The contract net protocol: High-level communication and control in a distributed problem solver," *IEEE Transactions on computers*, vol. C-29, no. 12, pp. 1104–1113, 1990.

[16] V. K. Garg and B. Waldecker, "Detection of weak unstable predicates in distributed programs," *IEEE Transcations On Parallel And Distributed Systems*, pp. 299–307, 1994.

[17] T.-S. Huang, L.-C. Fu, and Y.-Y. Chen, "Design and analysis of a dynamic scheduler for a flexbile assembly system," *Proceedings IEEE International Conference on Robotics and Automation*, pp. 3334–3339, 1997.

[18] C.-S. Jann and L.C.Fu, "Flexible control system for robot assembly automation," *Proceedings IEEE International Symposium on Assembly and Task Planning*, pp. 286–292, 1995.

[19] H.-S. Huang, L.-C. Fu, and J. Y. jen Hsu, "Rapid setup of system control in a flexible automated production systems," *Proceedings IEEE International Conference on Robotics and Automation*, pp. 1517–1522, 1997.

[20] D. E. Comer, *Internetworking With TCP/IP Vol 1: Principles, Protocols, and Architecture.* Prentice-Hall, 1991.

[21] W. R. Stevens, *UNIX network programming.* Prentice-Hall International, 1991.

[22] H. F. Wedde, B. Korel, S. Chen, D. C. Daniels, S. Nagaraj, and B. Santhanam, *Transparent Access to Large Files That Are Stored accross Sites*, ch. 9, pp. 490–510. IEEE Computer society press, 1994.