Rapid Setup of Shop-floor System Control in a Flexible CIM System

Han-Shen Huang and Li-Chen Fu
Dept. of Computer Science and Information Engineering
National Taiwan University, Taipei, Taiwan, R.O.C.
hshuang@clover.csie.ntu.edu.tw , lichen@csie.ntu.edu.tw

Abstract

In order to deal with demands for various products, flexible CIM systems are required to evade building dedicated systems for different products. However, the variety of products sometimes exceeds the flexibility of production systems, and new systems are inevitably needed. In the paper, we discuss the architecture of a flexible CIM system under which CIM components are well-modularized firstly. After that, EMFAK(Event-driven Multi-tasking Flexible Automation Kernel) is proposed to speed up the construction of shop-floor system control. At last, a flexible CIM system using EMFAK is described to show how to apply it to a real flexible CIM system.

1 Introduction

In order to deal with demands for various products, flexible CIM systems are required to evade building dedicated systems for different products. However, the variety of products sometimes exceeds the flexibility of production systems, and new systems are inevitably needed. Under such circumstances, it is desirable that new systems can be created rapidly or be transformed from the old ones easily.

Information technology can be applied on shop-floor system control to facilitate construction of new control systems. To solve the above situation, we developed EMFAK(Event-driven Multi-tasking Flexible Automation Kernel) to facilitate the job of building a new automation kernel, which is so-called system control[1]. The term control in this paper means the task of providing an information sharing and processing platform for all components, and coordinating them to make the whole system work properly[2].

First, we discuss the architecture of a flexible CIM system under which CIM components are well-modularized. In addition, information processing and sharing among components are defined in this architecture. Then, EMFAK is proposed to speed up the construction of shop-floor system control. At last, a flexible CIM system using EMFAK is described to show how to apply it to a real flexible CIM system.

2 Architecture

The architecture of a flexible CIM system consists of three kinds of component, that is, AK(Automation Kernel), scheduler, and device, as shown in Fig 1. The information sharing among schedulers and devices is through the AK, which can be built by EMFAK. Some special properties can be derived from this architecture. First, a scheduler becomes an easily replaceable module. Second, more than one scheduler is allowed in a CIM system to share the scheduling jobs because the information management and device conflict prevention are both handled by the AK[3][4][5].

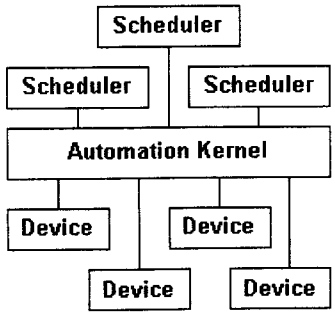

Figure 1: Architecture of a flexible CIM system

3 Automation Kernel and Device Model

It is important and practical to perform analysis on flexible CIM systems in detail before actually building them. Moreover, analysis of such systems is helpful in designing the automation kernel and reusable models for system devices. To exemplify this point, let us consider an example system in the following. After that, we propose an automation kernel and device models for the system.

3.1 Example System Specification

The example system produces two kinds of products simultaneously. There are two robot arms, one conveyor belt, one part loading device, one rotary buffer, and a system controller. The conveyor belt is equipped with two pairs of optical switches and a camera for identifying the position and orientation of incoming parts. Each robot arm has an eye-in-hand camera and several assembly sites for assembly jobs. Their detailed functions are described in the following:

1. ***Robot arm***: The two robot arms in the example system perform the same jobs. They can pick up parts from the conveyor belt, assemble them, and put them on the rotary buffer if necessary. Also, the robot arms can pick up parts from the rotary buffer.
2. ***Conveyor belt***: The conveyor belt transport parts into the example system. Equipped with a camera and optical switches, it can provide a guide to robot arms how to unload parts from the conveyor belt.
3. ***Part loading device***: The part loading device loads parts onto the conveyor belt when there is a robot arm available for assembly jobs.
4. ***Rotary buffer***: The rotary buffer is a temporary storage for parts. Both robot arms can reach the buffer.
5. ***System controller***: The system controller is in charge of the management of all other devices in the example system.

The communication among various devices of the system is via computer networks. If a device does not have any network interface, an interface translator is used as a bridge to the network. After knowing the jobs of the devices, we can develop device models for them.

3.2 Automation Kernel for Example System

It is time to demonstrate how to use an automation kernel (AK) as a system controller in a flexible automated production system. AK has to know the devices in the current environment, and how to process the system events. In our design, all events generated in the system are passed to AK first. When AK receives an event, it invokes a scheduler to process the event, as shown in Fig 2. According to the event handling routines and device models defined in AK, the scheduler chooses a routine and several needed devices. In event handling routines, an event is divided into several commands for the essential devices. There may be several steps in an event handling routine, and communication among devices is unavoidable. AK takes full responsibility in device communication and coordination.

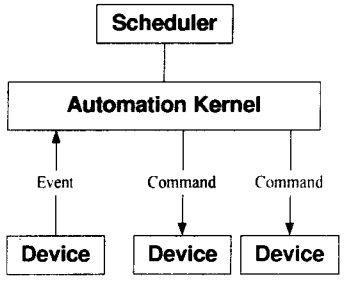

Figure 2: Translation of an event to commands

For our purpose, the AK is designed to receive the following events from other devices:

1. ***Registration*** and ***Dismissal***: *Registration* and *Dismissal* events bring a device into and out of the control of AK. They are submitted with the type and name of the device. On-line registration and dismissal allow dynamic reconfiguration of the whole production system.
2. ***Part_on_Belt***: The event informs AK that there is a part coming on the conveyor belt. Since the conveyor belt is equipped with a camera, the ID and orientation of the part are also obtained and passed to AK.
3. ***Result_Report***: The event is issued by devices to indicate whether they have processed a command successfully or not. The identity of the command and the result messages are also needed. The ID of a device is not necessary for two reasons. The first reason is that AK keeps track of command IDs and the corresponding receivers. The second is we take it into consideration that if a device can handle more than one command at the same time, we inevitably need the IDs of each command to distinguish them.

After AK receives an event, it passes the

event to its scheduler. In our design, a scheduler is a plug-and-play part of AK. AK can have more than one scheduler at its disposal. The scheduler selects the correct event handling routine and fills the needed parameters. After that, AK maintains the executing routines and issues commands to the related devices. A robot arm in the system receives the following commands from AK:

1. ***Unload_from_Belt***: This command notifies a robot arm to unload a part from a conveyor belt. Two parameters, the ID and orientation of the part, are needed.
2. ***Assembly***: When a robot arm receives this command, it performs the assembly job according to the parameters. The parameters are the site number and the part ID.
3. ***Load_onto_Buffer*** and ***Unload_from_Buffer***: These two commands order a robot arm to load a part onto buffer and unload from buffer. When a robot arm receives *Load_onto_Buffer* with a site number, it picks up the part on the site, and then moves to the buffer. If no site number is specified, the robot arm moves with the part that is already grabbed in its hand. As for *Unload_from_Buffer*, the robot arm goes to the buffer directly. It should be noted that the loading and unloading operations have not been finished yet because there must be synchronization between the robot arm and the buffer. For example, if a part is to be unloaded, the robot arm has to move to the loading/unloading port, and the buffer has to prepare the right part over there. The actions of the robot arm and buffer are simultaneously to save time. AK waits for the successful reports from the two devices, and issue the following command to the robot arm.
4. ***Synchronized***: When a robot arm receives the command, it knows synchronization is finished. Then it can proceed with the previous command.

AK sends the following commands to the buffer, that is,

1. ***Unloading*** and ***Loading***: The two commands tell the buffer to prepare a part, or an empty space to the loading/unloading port. The part ID and port ID are needed in the *Unloading* command. There is more than one port because the buffer serves two robot arms in our system. In the *Loading* command, only port ID is required because the buffer always finds an empty space for the robot arm.

The part loader receives only one command from AK, that is,

1. ***Load_Part***: When a part loader receives the command, it loads a part onto the conveyor belt.

After we introduce the events and commands, let us take the event *Part_on_Belt* as an example, as illustrated in Fig 3. Suppose that the names of the two robot arms are Adept and CRS. The conveyor belt sends *Part_on_Belt* to AK when a part comes. AK finds that Adept is doing assembly operation at that moment, whereas CRS is available. Hence the *Part_on_Belt* handling routine is called. "CRS" and the part information are the parameters for the routine. According to the routine, AK checks if the part can be used immediately. If not, AK orders CRS to put it on the buffer. Assume that CRS no assembly operation can be taken.

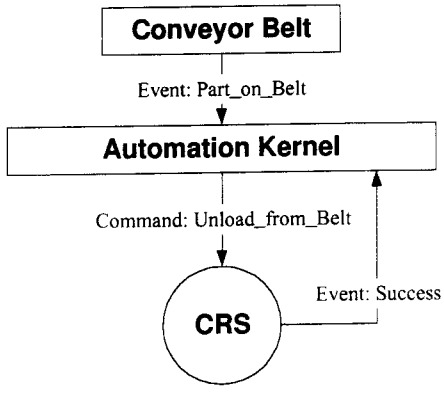

Figure 3: Event processing example

After the above inspection, AK sends *Unload_from_Belt* to CRS, waiting for the success report. After that, AK issues *Load_onto_Buffer* to CRS, and *Loading* to the buffer. When the two actions are finished, AK sends *Synchronized* to CRS, and CRS puts the part onto the buffer, as shown in Fig 4.

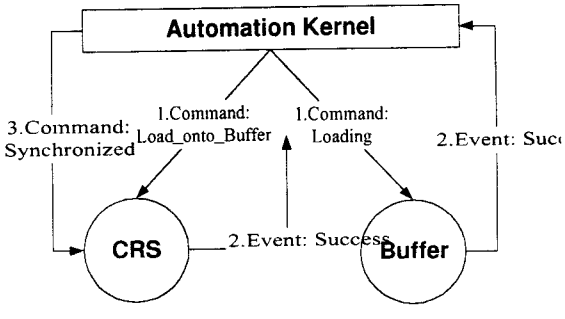

Figure 4: Synchronization example

3.3 Device Models for Example System

In EMFAK, we use finite state automata to model devices. The states in the diagram stand for the current state of a device, and the links represent the state transition when commands are received or commands are finished. Let us take the model of the rotary buffer as our illustrating example. Fig. 5 shows the model, which is derived straightly from the description of the previous subsection.

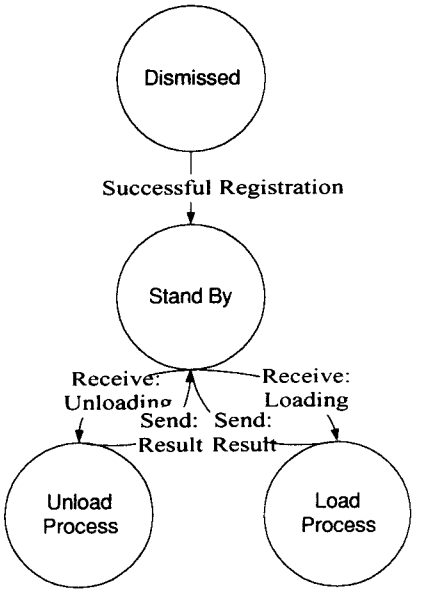

Figure 5: Device model of the rotary buffer in the example system

Since the rotary buffer serves one robot arm at a time, we can simplify the buffer model, as shown in Fig 6. The simplified device model illustrates two things. The first is that AK sends a command to a device and waits for a result report. It is not required for AK to know how a device finishes the mission. The second is that AK knows how many jobs a device can handle at a time. Suppose we have a more advanced buffer that can serve at most two robot arms simultaneously. The simplified device model can still be easily obtained in a systematic way, as shown in Fig 7. The state "Command Processing II" means that the device is handling two commands at a time.

From the example, it is clear that if we have a device that can handle n jobs at a time, there will be n "Command Processing" states. Although the device model seems to grow larger and larger in proportion to n, it does not matter to EMFAK. Since the device model grows in such a systematic way, EMFAK only needs to keep track of the number of commands which a device is executing.

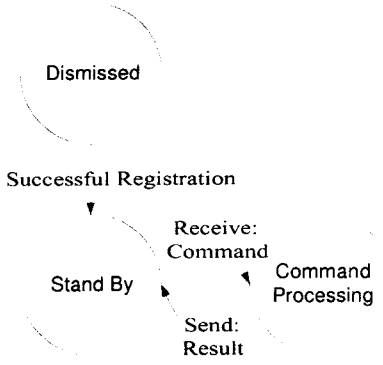

Figure 6: Simplified device model of the rotary buffer

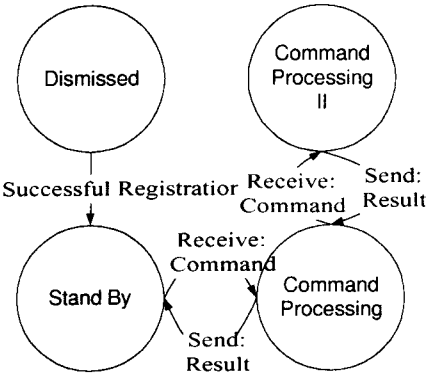

Figure 7: Simplified model for device that can handle at most two commands simultaneously

There is a special case for the modeling method. If a device can handle commands more than EMFAK can dispatch to it, we can model it in a more convenient way. Now that the number of the commands never exceeds the capability of the device in some environment, we can deem it a device that can handle infinite commands. It is not necessary any more to know how many jobs it is handling, and hence the device model can be depicted as in Fig 8.

The main purpose of the above example is to find out a way to build EMFAK. According to the discussion, we can retrieve the content of event handling routines and device models to form a programmable part for AK. In the next section, we take a further step toward the implementation of EMFAK.

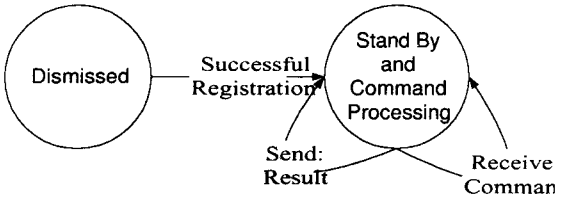

Figure 8: Special device model for a powerful device

4 EMFAK(Event-driven Multi-tasking Flexible Automation Kernel)

EMFAK is a programmable AK that can be used to facilitate the setup of system control. The organization of EMFAK is depicted in Fig 9. There are four components that make EMFAK effective, that is, EMFAK specification, device manager, task manager, and scheduler. The functions of these components are described below:

1. ***EMFAK Specification***: *EMFAK specification* is a file that describes the programmable portions of EMFAK, such as event handling routines and device models. To let EMFAK be able to run, EMFAK specification shall be referred to by all other components.

2. ***Device Manager***: When a device is registered, the device manager creates a device handler for it. A *device handler* maintain the information of the corresponding device and provides a common interface for Task Manager to communicate with a device.

3. ***Task Manager***: When an event handling routine is being executed, *Task Manager* creates a task to keep track of the current status of the routine. Task Manager has the ability of multi-tasking. If there is more than one task executable at a time, Task Manager selects the one with the highest priority to execute. If two or more tasks have the same priority, they are to be executed in turn. Multi-level queue method is used to implement multi-tasking ability[6].

4. ***Scheduler*** : It is a plug and play component in EMFAK. The communication between a scheduler and Task Manager is via computer networks, too. A scheduler must know the protocol with Task Manager, and the syntax of EMFAK Specification. Certainly, it should have domain knowledge of the system EMFAK controls.

In the following section, we explore a real example to see more details about using EMFAK.

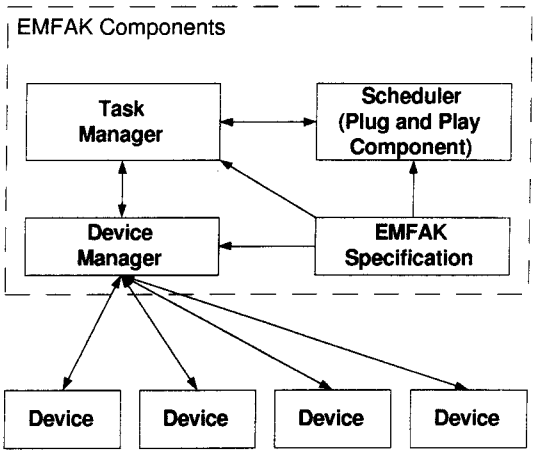

Figure 9: Organization of EMFAK

5 Experiment

In this section, we build an example CIM system to show how to set up a real system using EMFAK. The functionality of the system is described in the previous section.

EMFAK is a program that runs on UNIX. The communication between EMFAK and other devices is through computer networks.

Besides EMFAK, there are two robot arms, one buffer, one conveyor belt, and one part loader in the example. In this experiment, none of the devices has network interfaces. Hence, each of them is equipped with an interface translator. The two robot arms have RS-232 ports. The buffer and part loader only have discrete one-bit signal I/O ports. The conveyor belt has an optical switch and a camera. We allocate one PC for each robot arms, another PC with a special I/O card to deal with the buffer and part loader, and another for the conveyor belt to send the information of incoming parts to EMFAK.

The structure of the example system is shown in Fig 10.

6 Conclusion

We have introduced a way to set up system control rapidly. EMFAK provides an easy way to build a system controller based on the device

models. At last, we present an example to show how EMFAK can be applied to a real system.

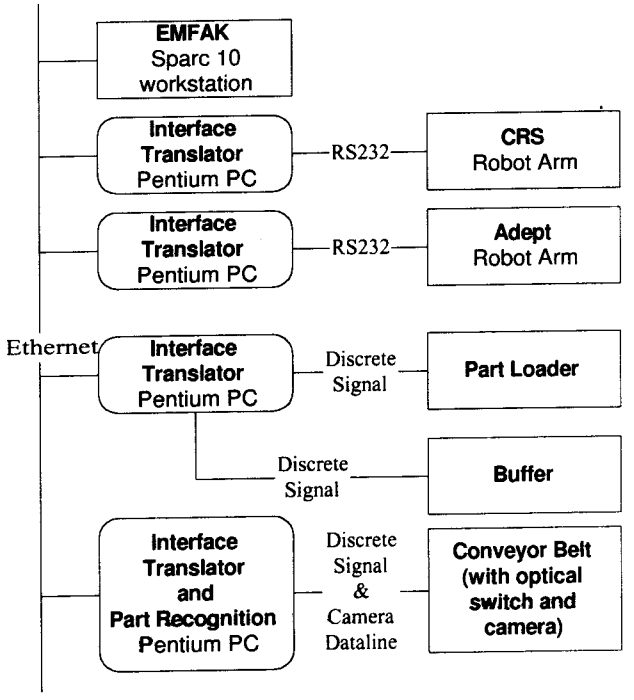

Figure 10: Structure of the experiment system

References

[1] L. Jann and L.-C. Fu, "Flexible control system for robot assembly automation", in *Proceeding of IEEE International Symposium on Assembly and Task Planning*, pp. 286-292, 1995.

[2] S. B. Joshe, E. G. Mettala, J. S. Smith, and R. A. Wysk, "Formal models for control of flexible manufacturing cells: Physical and system model", *IEEE Transaction on Robotics and Automation*, vol. 11, pp. 558-570, Aug. 1995.

[3] L. Lin, M. Wakabayashi, and S. Adiga, "Object-oriented modeling and implementation of control software for a robotic flexible manufacturing cell", *Robotics and Computer-Integrated Manufacturing*, vol. 11, no. 1, pp. 1-12, 1994.

[4] D. J. Miller and R. C. Lennox, "An object-oriented environment for robot system architectures", *IEEE Control Systems*, vol. 11, no. 2, pp. 14-23, 1991.

[5] Y. Jeon, J. Park, I. Song, Y.-J. Cho, and S.-R. Oh, "An object-oriented implementation of behavior-based control architecture", in *IEEE Int. Conf. On Robotics and Automation*, pp. 706-711, 1996.

[6] A. Sillberschatz and P. B. Galvin, *Operating System Concepts*. Addison Wesley, 1994.

[7] Liu, Song-Han and Li-Chen Fu, "Multi-agent Based Control Kernel for Flexible Automated Production System", in *Proceeding of IEEE International Conference on Robotics and Automation*, 1998.