

Rapid Setup of System Control in a Flexible Automated Production System

Han-Shen Huang, Li-Chen Fu and Jane Yung-jen Hsu
Dept. of Computer Science and Information Engineering
National Taiwan University, Taipei, Taiwan, R.O.C.

Abstract

In this paper, firstly we discuss the architecture of a flexible automated production system under which all components are well-modulized. Then, a solution, **EMFAK**(Event-driven Multi-tasking Flexible Automation Kernel) is proposed to speed up the construction of system control. At last, a flexible assembly system using EMFAK is described to show how to apply it to a real flexible automated production system.

1 Introduction

In order to deal with demands for various products, flexible automated production systems are required to avoid building dedicated systems for each product. However, sometimes the variety of products may exceed 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.

To solve the above situation, here 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 coordinating all devices in a flexible automated production system so that the whole system can work properly [2].

In this paper, firstly we discuss the architecture of a flexible automated production system, under which EMFAK can be properly used to control the system. Then, we present a simple example to explain the functions of an AK (Automation Kernel). After that, the organization of an EMFAK is described. At last, we present a real flexible assembly system that is made according to the architecture.

2 Architecture

The architecture of a flexible automated production system which is used in the paper is shown in Fig. 1. There are three kinds of component in it, that is, *AK*, *scheduler*, and *device*. The communication between schedulers and devices is through the AK [3] [4] [5].

There are some special properties derived from this architecture. First, a scheduler in a production system becomes an easily replaceable module. Then, more than one scheduler is allowed in a production system to share the scheduling jobs because the message flow management and device conflict prevention are both taken care of by the AK.

We can compare these components with those in a computer system to make the concepts of the whole architecture more clear. The AK is like an *operating system* in a computer system, which controls the system, ensures safety of the system during its operation, and provides users an easy way to use the devices. Then, the devices for material processing and handling can be viewed as the hardware, and the scheduler can be viewed as the application program because it will need to dispatch the devices during execution [6].

3 Example of an Automation Kernel

3.1 Example System Specification

The example system is a real prototype flexible assembly system. The system can produce two kinds of products simultaneously. There are two robot arms, one conveyor belt, one part loading device, one rotary buffer, two pairs of optical switches, two cameras above the conveyor belt, and a system controller. Each robot arm is equipped with an eye-in-hand camera and has several assembly sites for assembly jobs. Their functions are described below:

1. **Part loading device** : The part loading device here is a mechanism that loads parts onto the conveyor belt.
2. **Robot arm** : The two robot arms in the example system can perform the same jobs. They pick up parts from the conveyor belt and, then, either assemble them or put them on the rotary buffer. Also, the robot arms can pick up parts from the rotary buffer.
3. **Conveyor belt** : The only function of the conveyor belt is to transport parts into the assembly system.
4. **Optical switches** : There are two pairs of optical switches. They constantly detect the coming of parts, and then obtain the speed as a by-product.
5. **Cameras above the conveyor belt** : The cameras above the conveyor belt are for recognition of identities of the parts on the conveyor belt.
6. **Rotary buffer** : The rotary buffer provides temporary part storage. It can be reached by robot arms.
7. **System controller** : The system controller takes responsibility for the management of the material flow and all other devices of the whole system.

It should be noted that in this system, the communication among various devices of the system is through networks. If a device does not have a network interface, the so-called agent [1], which is an interface translator, is then needed as shown in Fig. 2. An agent can be implemented by a personal computer or other means which are capable of establishing both interfaces with the network and with the devices. Besides, the conveyor belt, optical switches, and cameras above the conveyor belt can all be viewed together as a transportation device which informs the system controller the identity of the coming part, the location of the part on the conveyor belt, and the moving speed of the part.

3.2 AK for the Example System

Now, we are going to demonstrate how to use an AK in a flexible automated production system. But before that, the terminology will be introduced first. If a scheduler would like to order other devices, or a device has to inform a scheduler of its situation, it sends a *request* to the AK first, then the AK translates the request to *commands*, as shown in Fig. 3.

For our purpose, the AK used here is designed to accept the following messages from other devices:

1. **Load_Parts** : When *Load_Parts* is received, the AK informs a part loading device to load a part onto a transportation device. This request has two parameters. One is the identity of the part loading device, and the other is the identity of the transportation device. It is because there may be more than one part loading device and one part loader may serve more than one transportation device.
2. **Robot_Unload_from_Belt** : This request asks the AK to dispatch a robot arm to unload parts from the transportation device. There are six parameters in this request, that is, the identity of the dispatched robot arm, the identity of the transportation device, the position, orientation, speed, and identity of the part.
3. **Robot_Do_Assembly** : When the request *Robot_Do_Assembly* is received, the AK notifies a robot arm to assemble the part in its gripper at the designated assembly site. Therefore, there are three parameters necessary for the request, namely, the robot arm identity, the identity of the part in the hand of the robot arm, and the working site.
4. **Robot_Load_onto_Buffer** : When the AK receives this request, not only does it inform the robot arm and the buffer involved in the request but also coordinate them so that the buffer is ready before the loading action. To fulfill this request, the identity of the robot arm, the identity of the buffer, the part that is going to be loaded, and where the part is now also have to be provided as parameters.
5. **Robot_Unload_from_Buffer** : The request has to be associated with a robot arm and a buffer. In this request, one parameter is needed, that is, the identity of the part to be unloaded.
6. **Scheduling** : Sometimes it is necessary for the system to ask the scheduler to be active. For example, when a part is loaded onto the transportation device, some recognition process must be executed. After that, the transportation device notifies the scheduler that it needs to perform job scheduling now by sending this request, *Scheduling*, to the AK.

7. **Registration** : The request brings a device or a scheduler into the production system. They can be controlled by the AK only after the registration process is performed. The type and the name of them are needed in the registration process.
8. **Dismissal** : The request is used to take a device or a scheduler out of the control of the AK. When they are going to cease working, the request must be submitted with the name of the device so that they will be excluded in further management of the system.
9. **Result_Report** : The request is used to indicate the success or failure of a command. The identity of the command and the result messages are needed.

After the AK receives a request, it has to take some proper responses, for instance, notifying a robot arm to perform some operation. But it should be noteworthy that notifications from the AK to devices are pre-defined, and all devices have to be able to recognize the notifications from the AK.

A robot arm in the system has to handle the following commands from the AK:

1. **Pick_from_Site** : This command informs a robot arm to pick up a part from the specified assembly site. The site number and the part identity shall be attached to the command because the robot arm must know where and how to pick up the part.
2. **Put_on_Site** : The command is the reverse action of *Pick_from_Site*. It tells a robot arm to put down a part on an assembly site. The parameters are the same as those in *Pick_from_Site*.
3. **Assembly** : When a robot arm receives the command, it performs the assembly job according to the parameters. The parameters are the site number and the part identity.
4. **Move_to_Buffer** : The command dispatches a robot arm to move over to the loading/unloading port of the specified buffer. When a robot arm loads a part onto or unloads a part from a buffer, both devices should act together to save time. Since the loading or unloading action of the robot arm must be performed after the action of the buffer is finished, synchronization between the two devices is necessary. As a result, the whole loading action is divided into two steps, *Move_to_Buffer* and *Load_onto_Buffer*. Similarly,

the unloading action consists of *Move_to_Buffer* and *Unload_from_Buffer*.

5. **Load_onto_Buffer** : This command tells a robot arm to load a part onto a buffer. The command follows a *Move_to_Buffer* command (theoretically speaking, it can also follow *Unload_from_Buffer*, but the robot arm will put the part which is just unloaded back onto the buffer). The identities of the buffer and of the part are also necessary because the loading action may differ in accordance with different buffers and different parts.
6. **Unload_from_Buffer** : This is an opposite command to *Load_onto_Buffer* concerning the physical action. The command also follows a *Move_to_Buffer* command, and the identities of the buffer and the unloaded part will be needed in this case.
7. **Unload_from_Belt** : This command notifies a robot arm to unload a part from a transportation device. Two parameters, the identities of the transportation device and the part, are needed.

The AK only sends two kinds of commands to the buffer, that is,

1. **Unloading** : This is a command that tells the buffer to send the specified part to the specified Loading/Unloading port. There are two parameters for the command, namely, the identities of the robot arm and the part. The identity of the robot arm indicates where the loading/unloading port is, and the identity of the part specifies what part the robot arm needs to handle. How to send the part to the port is planned by the local controller of the buffer.
2. **Loading** : When the buffer receives this command, it prepares an empty space for the specified robot arm. Hence, only the identity of the robot arm is needed as the parameter of the command.

When a new part is entered or a request is fulfilled, the AK will send the following command to the scheduler :

1. **Scheduling** : The command tells the scheduler to start the scheduling jobs.

The part loader receives only one kind of command from the AK, namely,

1. **Load_Part** : When this command is received by a part loader, it loads a part onto the transportation device.

Now, let us take the request *Robot_Load_onto_Buffer* as an example to explain the request and command. When the AK receives this request, it issues out two commands, *Move_to_Buffer* and *Loading*, to the specified robot arm and buffer, respectively. Then, the AK waits for the responses from both devices. If the responses are both ACKs, the AK sends the command *Load_onto_Buffer* to the robot arm. After all commands are executed successfully, the AK sends an ACK back to the component which sent the request *Robot_Load_onto_Buffer*.

The main purpose of the suggested example is to clarify the concept of the AK. After that, we will take a further step toward the implementation of an AK in the next section.

4 Event-driven Multi-tasking Flexible Automation Kernel

In this section, an event-driven multi-tasking flexible automation kernel (*abbrev. EMFAK*), which runs on SUN OS, is developed for an automated production system in which the communication among various devices is via computer networks. The organization of an EMFAK is depicted in Fig. 4, where there are six components that make an EMFAK effective, namely, *AK specification*, *task*, *task manager*, *task executor*, *network manager*, and *device handler*. The functions of all components are described below.

1. **AK Specification** : The *AK specification* is a file that describes the setting about the flexible portions of an EMFAK, such as the size of the transmission frame, the *tasks*, the socket port number for connection, the types of devices in the system, the products produced in the system, etc. To let an EMFAK be able to run, the AK specification shall be referred to by the *task executor*, *task manager*, and *network manager*. The details of the AK specification will be described in the next section.
2. **Device handler** : When a device is under the control of an EMFAK, a *device handler* which is a representative entity of this device exists in the EMFAK. A device handler is used to retain the information of the corresponding device and to take responsibility for the communication between this device and the EMFAK.
3. **Task** : A *task* is a dynamic component that is not always in an EMFAK. It is created according to

the AK specification when a request triggers the task manager. A request is the message from a device handler to generate a new task. Usually, a task is kept in the task manager. But when a task is to be executed, the task manager sends it to the task executor for execution. There is no task component appearing in Fig. 4 because it only appears in the task manager and task executor which will be explored more in the sequel.

4. **Task Manager** : The *task manager* is a process that accepts requests from device handlers and creates corresponding tasks, keeps track of the unfinished tasks, and decides which task should be executed next in accordance with the multi-tasking requirement. After making a decision, the task manager notifies the task executor of the next task for execution.
5. **Task Executor** : The *task executor* is a process that executes tasks which are sent from the task manager.
6. **Network Manager** : The *network manager* is a process which takes responsibility of the jobs that are related to the networks, such as building connections with other devices, sending messages to them, and receiving data from them.

In the following section, we will explore a real example to be more acquainted with the EMFAK.

5 Experiment

In this section, we introduce the structure of the example FAS to show how to set up a real system using an EMFAK.

Besides an EMFAK, there are two robot arms, one buffer, one part loader, one transportation device, and one scheduler in the example FAS. Except the EMFAK and scheduler, there is no network interface on other devices. Hence, in order to be connected to the EMFAK, those devices have to be equipped with an agent, which is a media translator. The two robot arms, *Adept* and *CRS*, both have RS-232 ports. The buffer, part loader, transportation device only have discrete one-bit signal I/O ports. Therefore, the agents of *Adept* and *CRS* robot arms must have RS-232 ports, whereas those of the buffer, part loader, and transportation device should have discrete signal I/O interface.

In the system, a 486 PC and a Motorola 68040 workstation are used to perform the agent job. The

PC has two RS-232 ports, but does not have signal I/O interface, but, the Motorola 68040 workstation has both kinds. Consequently, the agents of the buffer, part loader, transportation device, and the Adept robot arm are executed on the Motorola workstation, whereas an agent of the CRS robot arm runs on the PC.

Then, the structure of the FAS can be depicted in Fig. 5.

6 Conclusion

In this paper, we firstly introduce the concept of AKs. Then, in section 4, the concept of flexible automation kernels is described and a real implementation, EMFAK, is proposed. The existence of EMFAKs makes it even simpler to build a new AK. After that, in section 5, one example is presented to show how an EMFAK can be actually applied to a real 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. Joshi, 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.

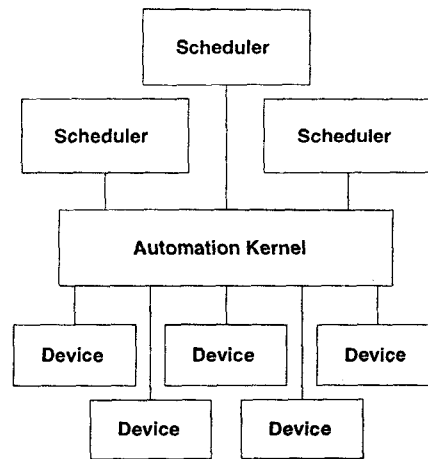


Figure 1: Architecture of a flexible automated production system

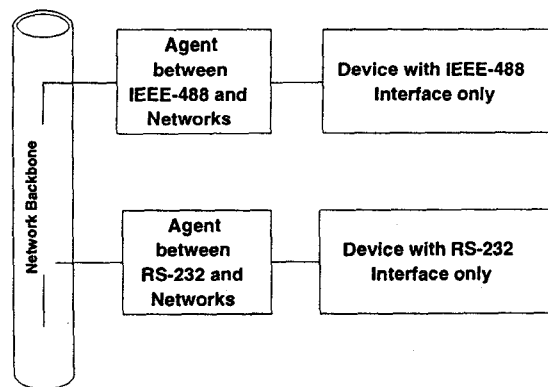


Figure 2: Example of agents

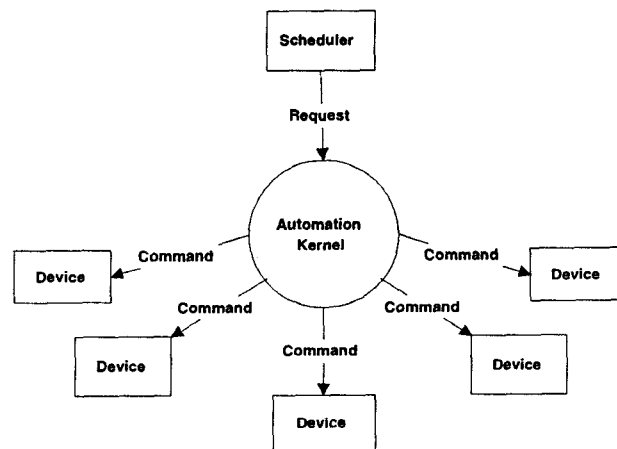


Figure 3: Translation of a request to commands

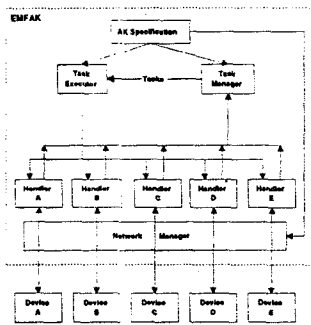


Figure 4: Organization of an EMFAK

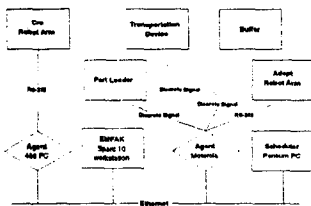


Figure 5: Structure of the example FAS