IMPLEMENTATION OF COMMUNICATING SEQUENTIAL PROCESSES FOR DISTRIBUTED ROBOT SYSTEM ARCHITECTURES

Gen'ichi Yasuda* and Keihachiro Tachibana**

**Department of Mechanical Engineering, Nagasaki Institute of Applied Science, 536 Aba-machi, Nagasaki 851-01, Japan*
***Department of Information Machines and Interfaces, Hiroshima City University, 151-5 Ozuka, Numata, Asaminami-ku, Hiroshima 731-31, Japan*

Abstract: This paper describes an implementation procedure of distributed control software for robot-based manufacturing systems. Robots and other devices such as conveyors and manufacturing machines are defined as object-oriented communicating sequential processes. A modular and hierarchical approach is adopted to define a set of Petri net type diagrams, which represent concurrent control processes for such devices, and asynchronous and synchronous interactions in global process interaction nets. Operational specifications are directly transformed to executable codes in a parallel programming language, such as occam. Implementations of control software on a multiprocessing environment and transputer networks are comparatively evaluated.

Keywords: Multiprocessing systems; parallelism; Petri nets; object-oriented programming; software development; industrial robot systems.

1. INTRODUCTION

Robot-based manufacturing systems, where individual robots pursue its own goals in such a way that some sorts of cooperation and conflict avoidance among related robots are performed in real time, can be viewed as complex discrete event systems of asynchronous, concurrent processes with distributed data and decentralized control. In such multirobot applications, run time synchronization of actions without rigorous scheduling should be achieved by direct communications among robots. Thus, distributed autonomous control architectures are desirable for effective real-time control performance, flexibility of system configuration and reduction of software development time (Hatvany, 1985). In this respect, object-oriented bottom-up approaches seem to be natural for such complex systems (Bucci and Vicario, 1992).

For the specification of flexible manufacturing cells, Petri nets have been utilized as a language suitable for purposes of verification, analysis and simulation. However, ordinary or extended Petri net modelling of complex manufacturing systems involves a highly large number of places and transitions. Petri net based interpreted or compiled program codes are inefficient because they do not exploit the parallelism of Petri nets. Owing to an increasing degree of operational integration of robots and the complexity of real-time control software, suitable techniques and tools for the specification and validation of requirements and for their systematic transformation to executable codes are required.

In this paper, robot systems are specified as a collection of communicating sequential processes, and multiprocessing control structures and their implementations on parallel distributed processing

architectures are presented. Global Petri net modelling is described to specify the discrete event dynamics of robots and other industrial devices, and their asynchronous and synchronous interactions. Control structures of the data flows in control systems are defined as a set of protocols for interprocess interactions to perform cooperative tasks. A modular and hierarchical approach for task specifications is adopted toward the integration of more complex robotic manufacturing systems, such that some important structural properties for manufacturing systems, such as boundedness, liveness and reversibility, are maintained in the implementation of control systems. The operational specifications using Petri nets are independent of the implementation mechanisms, and directly transformed to executable codes in a general parallel programming language, occam, the language designed for Inmos transputers.

2. SPECIFICATIONS OF ROBOT SYSTEMS

The specification procedure first defines physical objects in manufacturing systems as state machines, a subclass of ordinary Petri nets. The basic idea behind the distributed autonomous realization of robot systems is the decomposition of the overall control system into a set of autonomous control processes or state machines which communicate asynchronously or synchronously among themselves.

2.1 Object-oriented Petri Net Modelling of Basic Robot Process

To model basic robot processes in manufacturing systems, strongly connected Petri nets, where tokens circulate according to the firing rules associated with transitions, with communication interfaces are introduced. The firing consists in removing the tokens from the input places and adding tokens to the output places after performing the associated actions. In the proposed specification procedure, the firing of a transition corresponds to the beginning or end of a robotic or machine task, or a communication operation, and at the transition firing some instruction in the program code is executed. Thus, a procedure which executes a task can be associated with the transition corresponding to the beginning and the place between the two successive transitions. Hence, a place represents a state of executing a task or waiting for some message by asynchronous communication which can be a precondition or postcondition of a transition.

State machines, a Petri net subclass where each transition has exactly one input place and one output place, model autonomous control processes of robots and other machines. State machines, like real single robot processes, do not include synchronization of concurrent processes. A modular, bottom-up approach is adopted, although an ordinary Petri net can be decomposed into a set of state machines (Breant and Paviot-Adet, 1992). The discrete event dynamics of a robot or machine can be defined in terms of state machines at various aggregate levels. Fig.1 shows a global Petri net representation of a robot process; the highest level control structure using global working and free states of the robot. Each robot performs an independent task when command from the coordinator is for independent action. On the other hand, when command is for cooperation, the robot perform a cooperative task synchronously with the other robot or machine. Additional pieces of information such as timing requirements and priorities can be associated with mutually exclusive transitions in charge of task request. Each robot process or autonomous control process can be viewed as an object which controls the data flow according to the current state and incoming messages.

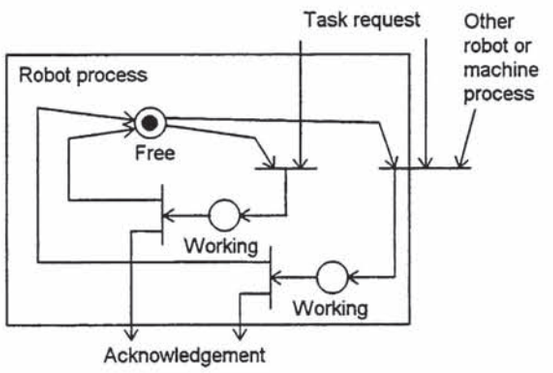

Fig.1. Global Petri net representation of robot process.

2.2 Interaction Mechanisms between Autonomous Processes

The modular approach affords to build complex validated Petri nets by specifying how autonomous control processes will interact with each other. For discrete event systems, two types of interactions among processes, asynchronous and synchronous ones, are distinguished. After the specifications of autonomous control processes, a process interaction net which describes interactions among these processes as asynchronous and synchronous communications, is specified (Breant and Paviot-Adet, 1992). Ordinary and asynchronous interaction is modelled by a place which does not belong to any autonomous control process. On the other hand, synchronous interaction is modelled by a shared transition as shown in Fig.2. Local data exchange within one process, or state machine, can also be specified by a place belonging to the process.

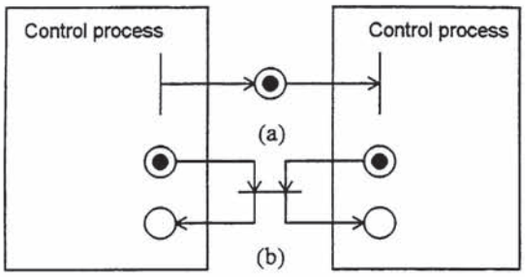

Fig.2. Schematic of (a) asynchronous and (b) synchronous interactions.

A process interaction net (Breant and Paviot-Adet, 1992) of a typical robot system composed of two cooperating robots is shown in Fig.3. The coordinator process and the two robot processes are autonomous control processes, that is, state machines. The places and the transition model asynchronous and synchronous interactions between the associated processes, respectively. The two places are in charge of control functions for each robot to perform its own tasks exclusively or cooperatively with the other robot. The transition has one precondition place and one postcondition place which models the monitoring process. The functions of these places and transition are managed by the dedicated asynchronous and synchronous interaction control processes.

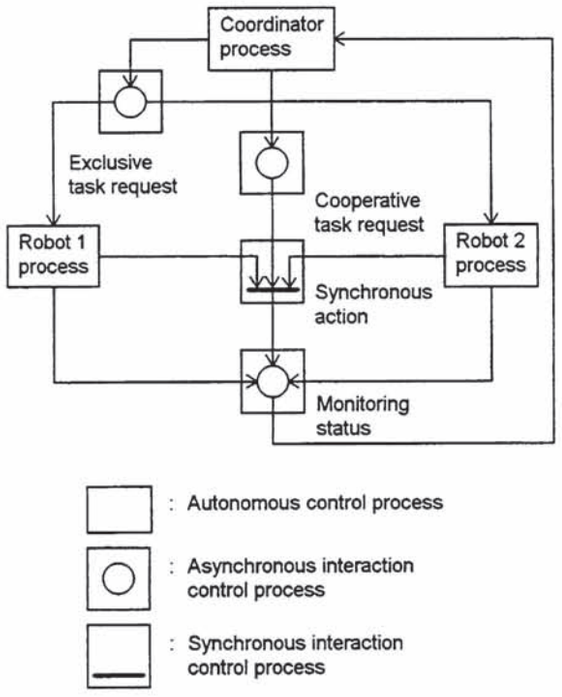

Fig.3. Example of process interaction net of two cooperating robots.

System coordination is performed based on the transition firing of the Petri net model. For example, a detailed Petri net representation of a loading task from a conveyor by a robot is shown in Fig.4. The coordinator process sends messages of request to the autonomous control processes, that is, the robot process and the conveyor process, for the execution of specified tasks. The robot and conveyor processes perform their tasks concurrently with mutual communications, which are managed by the dedicated asynchronous interaction control process. When the tasks are successfully completed, messages of acknowledgement or status are sent to the coordinator process in return through the monitoring process. Messages for exception handling and error recovery are sent from the monitoring process to the coordinator process if any error arises during task execution. Cooperation between two robots is represented by similar control structures (Yasuda and Tachibana, 1994).

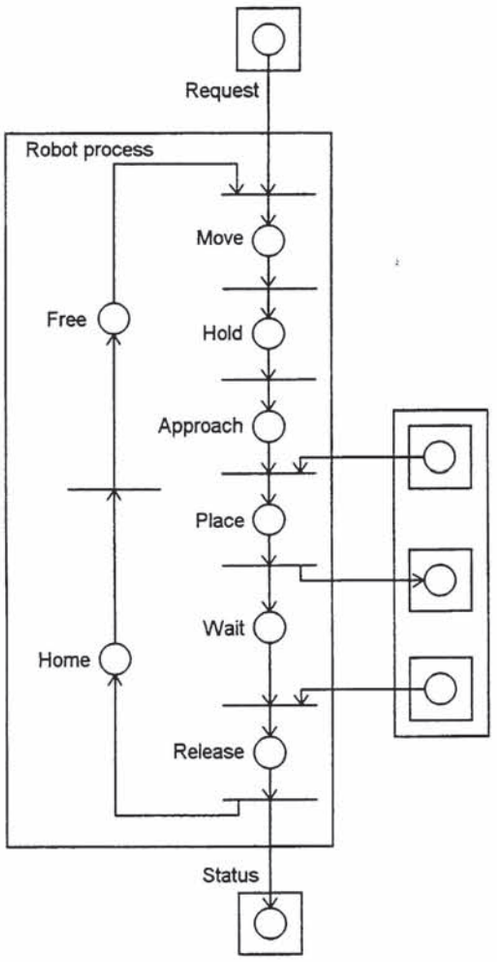

Fig.4. Petri net representation of robot process for loading task to conveyor.

3. MAPPING OF SPECIFICATIONS ONTO OCCAM CODE

The Inmos transputer with its associated parallel programming language, occam, was selected as the target architecture owing to the sound formal theory behind the model of parallelism and synchronous

communication (Hoare, 1985). The correspondence between Petri nets and a subclass of occam constructs is adopted. The nature of the mapping between specifications and occam processes is such that state reachability trees are analysed empirically using corresponding Petri nets (Tyrell and Holding, 1986) to show the desired properties for robot-based manufacturing systems. The details of the above control processes are described below.

3.1 Autonomous Control Process

Autonomous control processes are essentially sequential processes as state machines although they may include control flows with decisions implemented through IF and WHILE constructs in occam, which are also represented using corresponding Petri nets. Each place of a state machine is associated with a state of the sequential process, and the marking by a single token indicates the current state of the process. A simple sequential process is mapped onto occam code using an IF construct, where each case of the IF construct corresponds to one place or state of the Petri net representing the sequential process. Fig.5 shows an example occam program of the robot process, where there are three cases corresponding to the places of the Petri net model in Fig.3. An independent or cooperative task is selected alternatively by an ALT construct. After completion of each requested task, the process moves the robot to its home position.

Autonomous control processes can also include decisions which perform conflict resolution or choice between alternatives. Conflict resolution is transformed to guarded alternative processes; an ALT construct with guards such as input, Boolean and timing expressions in occam.

```
PROC robot.process( )
  SEQ
    . . . initialize process
    WHILE continue.flag
      IF
        state = AUTO
          SEQ
            . . .
        state = SYNC
          SEQ
            . . .
        state = HOME
          SEQ
            move.to.home( )
            ALT
              auto.task.request ? auto.ready
                state := AUTO
              SKIP
                state := SYNC
        . . .
    . . . end process
:
```

Fig.5. Example occam program of robot process.

3.2 Asynchronous Interaction Control Process

Asynchronous interaction is represented as a place and managed by the dedicated asynchronous interaction control process. Token passing between two autonomous control processes is implemented through asynchronous communication. When a transition is fired, one process sends a message and a token to the asynchronous interaction control process in charge of the postcondition place. The other process sends a message to the asynchronous interaction control process to evaluate the precondition place and to try to receive a token. Thus, the asynchronous interaction control process updates the token counter as a postcondition or precondition place when firing of any connected transition occurs. Fig.6 shows an example program of the asynchronous interaction control process for token passing from the coordinator process to the synchronous interaction control process in Fig.3. Acceptance of a cancelling message from the control process can be included to restore a token when precondition evaluation fails.

```
PROC async.control( )
  WHILE control.flag
    ALT
      . . . receive request from coordinator
        SEQ
          . . . evaluate token counter
          . . . send response to coordinator
      . . . receive request from synchro.control
        SEQ
          . . . evaluate token counter
          . . . send response to synchro.control
:
```

Fig.6. Example occam program of asynchronous interaction control process.

The monitoring process is the special asynchronous interaction control process and intermediates asynchronous communications regarding system status. In case of mutual exclusion, a token which means completion of the task by one robot process is sent to the place which the monitoring process manages. The coordinator process communicates with the monitoring process in order to generate the next command. The monitoring process also manages messages from external sensors to detect exceptional situations such as operation failure and machine stoppage, sends emergency messages to the coordinator process and other autonomous control processes, and coordinates some error recovery process.

3.3 Synchronous Interaction Control Process

Synchronous interaction or rendezvous among autonomous control processes is represented as one transition and managed by the dedicated

synchronous interaction control process. An example program for cooperating robots is shown in Fig.7. When every robot is ready to perform rendezvous, the procedure for synchronous action is called. The procedure includes an appropriate cooperative task among associated robots. If any robot is not ready for rendezvous, the interaction control process informs the ready robot process that the rendezvous can not be performed, then the process changes its state. Timed rendezvous can be implemented on the robot processes and the synchronous interaction control process.

```
PROC synchro.control( )
  WHILE control.flag
    SEQ
      . . . initialize
      WHILE continue.flag
        ALT
          . . . receive request from robot1
          . . . receive request from robot2
        SEQ
          . . . execute synchronous action
          . . . send a token in postcondition place
          . . . send messages to robot1 and robot2
:
```

Fig.7. Example occam program of synchronous interaction control process.

3.4 Hierarchical Specifications of Robot Process

Generally, for real robot-based manufacturing systems, Petri nets generate large, unmanageable nets. Based on the hierarchical approach of the proposed procedure, Petri nets are specified and translated into occam codes by stepwise refinements from the highest global control level to the lowest local control level. A global and simple Petri net model is first chosen which describes the aggregate system structure and satisfies the boundedness, liveness and reversibility. Then, at each step of specification, some parts of the Petri net, transitions or places, are substituted by a subnet in a manner which maintains the structural properties. This procedure enables information hiding, because the detailed behavior of each robot or machine process can be easily extracted from the net.

Fig.8 shows that the robotic task specified within the robot process is substituted by the detailed specifications of the robot motion process, the robot controller interface process and the sensor process. The motion process for trajectory generation and its real-time modification has the interface part, that is, places and transitions for asynchronous communication with the global robot process. Overall control software is implemented using simple asynchronous communications with messages of request and acknowledgement between two associated autonomous control processes.

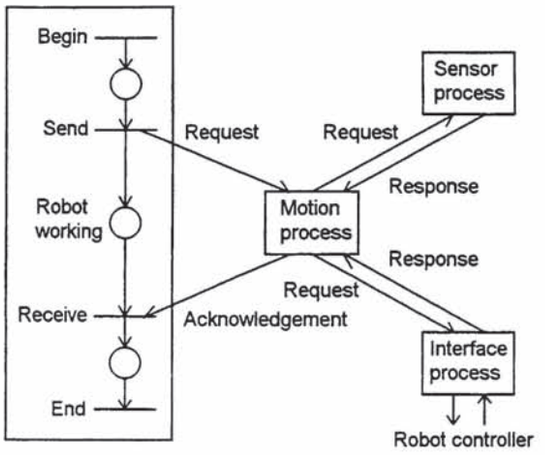

Fig.8. Hierarchical specification of robot process.

4. IMPLEMENTATION AND PERFORMANCE

4.1 A Multiprocessing Control System

To perform sensor-based robot control and cooperative multirobot control, a multiprocessing control system using a general real-time multitasking operating system based on object-oriented programming on a single microcomputer has been implemented (Yasuda and Tachibana, 1994; Yasuda, 1996). The preemptive scheduling and 10~150 ms time slicing with priorities are incorporated into the rescheduler's program. Context switching is successfully realized under the restrictions caused by the non-reentrant MS-DOS.

The multiprocessing control system has four types of processes: the coordinator process, the robot motion process, the robot controller interface process and the sensor process. Each process communicates with other processes via mailboxes. Each robot motion process handles a separate robot arm, executing high-level motion commands sequentially from its own mailbox. If the queue is empty, it suspends itself and waits for a new command. For the proposed procedure, the autonomous control processes as well as asynchronous and synchronous interaction control processes were successfully simulated and proved to have freedom from deadlock.

Currently, several examples of robot systems have been developed using the proposed specification and implementation procedure. Robotic handling operations synchronized with a conveyor were successfully implemented. For vision-based robot positioning, multitasking processing is effective owing to cooperation between a robot and its vision sensor system in comparison with sequential processing; the robot can reach, grasp and start moving the workpiece while the vision system is busy making a decision from its extracted feature

parameters. As an example of cooperative multirobot control, pick and place operations, in which the two robots perform synchronous communication with each other for exchange of a workpiece, were also successfully implemented and experimentally validated.

However, such control systems using conventional processors cannot handle the full complexity of the implementation of parallel distributed control structures due to the following reasons:

(1) in addition to preemptive or event driven scheduling, round robin scheduling, where the time slice is required to be shorter than the time required for processing and communication of one motion command, about 100 ms using NEC PC-9801FA, is essentially needed for two robots to move simultaneously;

(2) for smooth robot motion control, the sampling interval largely depends on the time required for processing of one motion command; and

(3) executable code is inefficient and the safety cannot be logically proven, because multiprocessing control software is written as polling loops or with interrupt routines.

Owing to these disadvantages, control system implementation using a multitasking operating system on a conventional processor lowers the performance, flexibility and extensibility.

4.2 Transputer Networks

For distributed implementation of Petri net based control structures on transputer networks with true parallel processing and message passing primitives effectively handled in hardware, a bidirectional communication link between transputers or a pair of channels within one transputer, is assigned to each directed arc in the process interaction net. The overall control software can be implemented as a parallel construct of the control processes on the transputer network, and does not need any operating system software for managing the transputer and the whole transputer network.

The control system can be implemented on different hardware and software structures. Hierarchical control structures are classified according to autonomy of low level control modules. Real-time control performance of distributed autonomous control systems is evaluated with respect to throughput, response time and synchronization between transputer modules for simultaneous or coordinated actions in response to user command.

The experimental results showed that the proposed procedure generates correct codes. Program code in occam is very efficient because of its small instruction size. Advantages of the transputer network for robot system architectures are confirmed with respect to real-time control performance, flexibility of system configuration and software development time in comparison with conventional approaches using commercial real-time multitasking operating systems.

5. CONCLUSIONS

The specification and implementation procedure of communicating sequential processes for distributed robot system architectures in robot-based manufacturing systems has been described. Distributed implementations of Petri net based control structures are illustrated aiming at high performance parallel processing of low level control modules on transputer networks. The procedure supports rapid prototyping and reusability, since a communicating sequential process or control process already described using a Petri net can be kept as a software module in a library and instantiated repeatedly to link it with other processes for real robot system applications.

REFERENCES

Breant, F. and E. Paviot-Adet (1992). OCCAM prototyping from hierarchical Petri nets. In: *Transputers'92*, 189-209, IOS Press, Amsterdam.

Bucci, G. and E. Vicario (1992). Rapid prototyping through communicating Petri nets. *Proc. 3rd IEEE Int. Workshop on Rapid System Prototyping*, 58-75.

Hatvany, J. (1985). Intelligence and cooperation in heterarchical manufacturing systems. *Robotics and Computer Integrated Manufacturing*, **2**, 101-104.

Hoare, C.A.R. (1985). *Communicating Sequential Processes*. Prentice-Hall International, London.

Tyrell, A.M. and D.L. Holding (1986). Design of reliable software in distributed systems using the conversation scheme. *IEEE Trans.*, **SE-12**, 921-928.

Yasuda, G. and K. Tachibana (1994). A parallel distributed control architecture for multiple robot systems using a network of microcomputers. *Computers & Industrial Engineering*, **27**, 63-66.

Yasuda, G. (1996). An object-oriented network environment for computer vision based multirobot system architectures. *Proc. 20th Int. Conf. on Computers & Industrial Engineering*, 1199-1202.