A parallel control architecture for industrial robot cells

Dominik HENRICH, Frank ABEGG, Christian WURLL and Heinz WÖRN

Institute for Process Control and Robotics, o. Prof. Dr.-Ing. H. Wörn, University of Karlsruhe, Depar t-ment of Computer Science, PO Box 6980, D-76128 Karlsruhe, Germany, e-mail: dHenrich@ira.uka.de. Further information can be found on the Web pages of the PaRo group (Parallel Robotics) of the IPR at http://wwwipr.ira.uka.de/~paro/

Abstract. We present a parallel control architecture for industrial robot cells. It is based on closed functional components arranged in a flat communication hierarchy. The components may be executed by different processing elements, and each component itself may run on multiple processing elements. The system is driven by the instructions of a central cell control component. We set up necessary requirements for industrial robot cells and possible parallelization levels. These are met by the suggested robot control architecture. As an example we present a robot work cell and a component for motion planning, which fits well in this concept.

Key Words. industrial robots, robot control architectures, parallel processing, distributed processing

1 Introduction

In spite of significant improvements in processing speed, sequential processors are far from rendering sufficient computing capacity for an advanced robot system. On the other hand, modern VLSI technology offers a unique opportunity to close this gap by parallel computing. It can be expected that the progress in the design of new VLSI circuits and the reduction of component costs will make parallel machines containing several *processing elements* (PEs) very economical.

In order to use a parallel system, it is important to show the feasibility of parallelizing existing problem solutions in robotics. In several cases, fundamentally new concepts have to be developed, so that a parallelization is possible. Specially designed computer architectures for robot control are surveyed in (Graham, 1989). The historical development of control structures of automated manufacturing can be followed in (Dilts et al., 1991).

A classification scheme for robot control architectures has been proposed in (Hörmann, 1989). It covers the extreme viewpoints of the historical development, hierarchical and distributed control. Additionally, function-oriented and behavior-oriented approaches are distinguished. Altogether, this results in four different classes. For parallel processing, each function or each behavior can be performed by an extra PE.

The paper is organized as follows: first, we will introduce a system concept of a parallel control architecture for industrial robot cells in Section 2. Then, a common communication mechanism for the involved components is presented in Section 3.

An example for a parallel control architecture and for parallel components are outlined in Sections 4 and 5, respectively. The related work is presented and analyzed in Section 6. Finally, the paper closes with a conclusion and the future work in Section 7.

2 System Concept

Before discussing parallel control architectures, it is important to explain what a control architecture is. According to (Dilts et al., 1991), a control *architecture* makes a control *system* from control *components*. The architecture determines the interrelationships between the components and the mechanisms for coordination.

Requirements on robot control architectures can be described from a general point of view (Fayek et al., 1993), for manufacturing systems (Dilts et al., 1991), and for software architectures of robot control (Fleury et al., 1994). Important requirements from the parallel processing point of view are reported in (Henrich and Höniger, 1997).

The goal of our system concept is to provide a scalable and flexible high level robot control architecture for a complex manipulation task that fulfills the requirements described in (Henrich and Höniger, 1997) and which covers many levels of parallelization. A definition of these levels can be found in the same reference. In order to provide high efficiency, our system is divided into subsystems with different functionality which may run on different PEs and communicate by an efficient message passing protocol.

By providing well defined and standardized interfaces for the subsystems, we make extensive use of commercially available software modules and processing hardware. For example, the real time kinematics control system of our robot is a commercial product purchased together with the robot. Thus, we can concentrate on developing the high level control architecture which is important for overall system efficiency.

In the following, a subsystem is also called a *component.* According to the definition of (Mehlhaus, 1994) a component implements a set of related functions and can either be a physical or a logical component. Logical components run as different processes in order to have the possibility to run them on different PEs for higher efficiency. Additionally, a component process can be parallelized at the algorithm level like the automatic motion planner described in Section 5.

The system of components from this abstract point of view is shown in Figure 1. Each component is linked to the interconnection unit. This *interconnection unit* may solely transfer messages from one component to another (communication) or may have some intelligence and decision capabilities (coordination). The set up of the system and the distribution of the component processes is done by the cell *configuration component* which is invoked only at the beginning of the cell process and is idle after the system has been investigated.

Additionally, a graphical *user interface* and a high level cell *control component* are introduced. The latter provides a common clock, resource management, task scheduling, action planning, action coordination and process monitoring according to (Mehlhaus, 1994). All instructions to other components are given or planned by it and all results return to it without passing several layers of components. Thus, our system establishes a flat hierarchy: the main process control at the top level and the other components at the second level with little functional dependencies in order to support short response times.

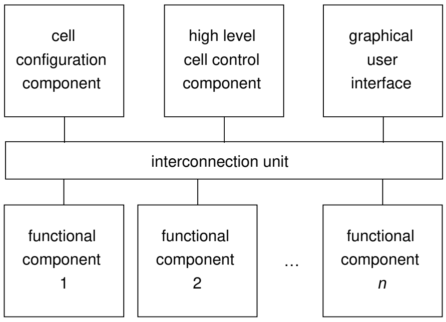

Figure 1: Concept for the parallel robot control architecture consisting of multiple components and an interconnection unit

In order to keep our system extensible and its components exchangeable, all software components need to have an identical structure. The components consist of three parts: a local data base, a set of functions to fulfill the task of the component, and an interface for coordination and communication (see Figure 2a). In the following, these three parts are discussed.

Each component provides a set of functions, which can be used by other components. Together with the interconnection unit, this enables a component to fulfill a given task by cooperating with the other components. This is the main part of each component and will certainly be modified during development. For example, with a motion planning component, such a function may generate a collision-free path from a given start to a goal configuration in a given environment (see Section 5).

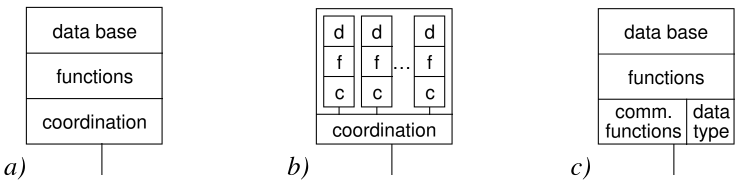

Figure 2: Different views of the component structure: a) basic parts of a component, b) component with subcomponents for parallel processing, c) component with coordination part split up in communication functions and data types.

Furthermore, the local data base may be accessed by the component itself as well as by other components. For that reason, other components must access the coordination/communication part of the local component. Global coordination and communication libraries containing common communication functions and data types allow easy exchange of components.

The function part of a component may further be divided into subcomponents (see Figure 2b). These subcomponents have the same structure as described above. The subcomponents fulfill the component task by cooperation and may be easily be processed in parallel. This introduces parallel processing on the algorithm level of a robot control architecture, confer (Henrich and Höniger, 1997). Still, the coordination part of the component is necessary to keep control of the subcomponents. Additionally, the coordination part hides the subcomponents from the other components of the system.

The proposed control architecture covers several levels of parallelism introduced in (Henrich and Höniger, 1997). It can be clearly seen that parallelism at the robotics level is supported by the component concept when running components on different PEs. Presuming that components consist of related functions and build closed units, the component concept also provides parallelism on the functional and on the abstraction level. We also make use of parallelism at this level when using different PEs for at least low level robot control and high level cell control. Parallelism at the kinematics and control level are beyond the scope of this paper. For this, we can make use of an industrial robot provided with an adequate real-time control system. Additionally, there are many investigations on parallel low level robot control.

3 Communication Concept

In this section, the communication part of the components and the message passing protocol are discussed. Since the components communicate by message passing, we are designing a system of communicating sequential processes, CSP (Whitcomb and Koditschek, 1990). Emphasis is laid on solutions to cope with the high time constraints of a complex manipulation task.

Regarding a parallel computer with a few PEs and wanting to make use of the full inherent parallelism of the system, we implement an asynchronous communication protocol: Neither the communication nor the computing processes have a common clock, and the cell control can invoke new tasks even if old tasks are not finished. Thus, we get a more efficient system with little waste of computing time (Gruhler, 1993) and a system which is not restricted to a synchronous remote procedure call concept, like the system reported in (Coulouris et al., 1994). For this reason and for its flexibility, we use the Parallel Virtual Machine System (Geist et al., 1994) for communication between the components. Another feature of our protocol is the use of messages including several instructions for different components, each used in a sequence of component calls which we call *instruction chains*.

Using a state transition model for components, every component has a defined state at any point in time (Mehlhaus, 1994). With the knowledge about the states of a communication partner, the coordination between the components becomes easier. Messages are sent to message handler modules, thus simplifying structural programming and detecting errors in component coordination.

In order to cooperate with other components, the components have a well defined, reusable and simple interface. It provides at least *send* and *receive* instructions with a syntax common to all components (Drachenfels, 1997). Thus, the coordination part of the components can again be subdivided into the communication functions and a set of data types (see Section 2). The data types are available to all other components enabling them to handle the transferred data (see Figure 2c).

In order to have reusable interfaces, the communication functions can be divided into standard functions, which are useful for each component, and into functions which are parameters for standard send and receive functions. Standard functions have to be written only once while instruction parameters require individual evaluation. Some examples for these two ways of implementing communication functions are listed in Table 1. These functions can also be of different types. Functions for coordination, for data transfer and for instructions are distinguished.

Communication	**Standard functions**	**Send/receive functions**
for coordination	terminate_component() acknowledge_message() wait_for_signal()	send(wait_for_robot_move) send(wait_for_planned_motion) send(send_tasks_to_component)
for data transfer	get_variable() get_result() send_result()	send(get_robot_pose) send(get_robot_configuration) send(get_task_list)
for instructions	function_call() component_begin_work()	send(trigger_sensor) send(plan _motion) send(shoot_ image)

Table 1: Several communication functions of the parallel robot control system

Now, the different types of message routing for instruction chains will be presented. According to the client/server mechanism, one component exchanges data with another component by calling functions from it. The other component executes this function with the transferred parameters and returns the result to the first component. This may either be the result of a complex computation or a simple acknowledgment message (Figure 3a). For saving communication overhead and for introducing parallel processing, we give special attention to the case where further components are called by the second component within such a function call (Figure 3b).

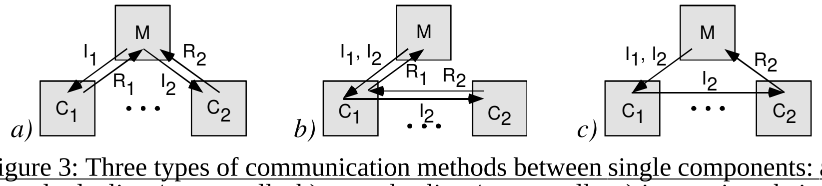

Figure 3: Three types of communication methods between single components: a) standard client/server calls, b) cascade client/server calls, c) instruction chains

Additionally, we may save time by using instruction chains, which include several instructions for several components using fewer messages (Figure 3c). Thus, instructions are routed to other components and multiple instructions are packed into as few messages as possible. Here, a message must contain the complete information of all instructions in a chain and information about all parameters, even of those which will be used later. There are three ways to invoke such instruction chains: (1) one message with a list of instructions and corresponding parameter sets; (2) one single message with a meta-instruction (invoking the instruction chain) and the list of necessary parameter sets; (3) multiple messages with one instruction and corresponding parameter set each (the instructions in total make up the desired instruction chain)[1]. We will choose the last method because it only needs to implement a few orthogonal functions per component. This allows components and their configuration to be easily exchanged.

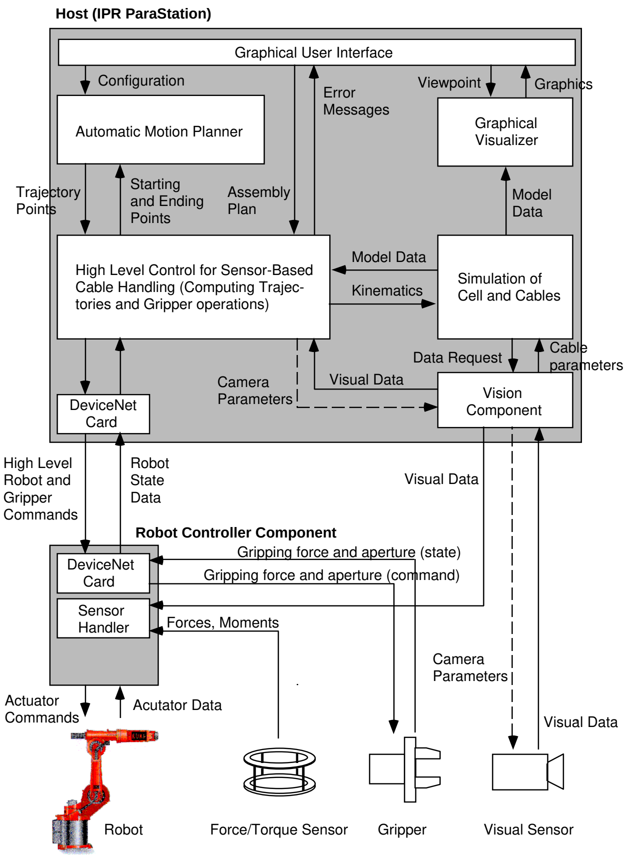

Figure 4: Location and interaction of the components of the example robot system

---

1 This method is similar to multiple client/server calls, but without the need to transfer huge result data via the initiating master component.

4 Example Robot Control System

For validating the above described system concept with its communication mechanism, we have built up an industrial work cell. It consists of an industrial robot, a KUKA KR15, with 6 DOF and the PC-based low-level control system KUKA KRC. For manipulation tasks, the robot manipulator is equipped with a pneumatic two finger gripper, supported by a force-torque-sensor and a vision system.

The platform for the high level control architecture is a PC-based workstation cluster, the IPR-ParaStation. Each of the nine workstations has a 133 MHz Intel Pentium processor and 64 Mbytes memory. The communication is established either by an Ethernet bus or a parallel mesh-based network described in (Wurll and Henrich, 1997).

For the robot system example, the location and interaction of the components are shown in Figure 4. Each physical component is reflected by a corresponding logical component which runs at least at one node of our (parallel) host computer. The communication between the workstation cluster and the robot control unit is established by an additional DeviceNet board at one node of the cluster. The DeviceNet protocol based on a CAN bus is supported by the KRC and enables us to control the robot externally. All other peripheral components use the Ethernet for message and data exchange.

5 Example Component

One essential component of a sophisticated robot control system is the motion planning component. The subject of robot motion planning has been studied for a couple of decades and many important contributions to the problem have been made (Hwong and Ahuja, 1992). Motion planning algorithms are of great theoretical interest, but are rarely used in practice because of their computational complexity (Kamal et al., 1996). Generally speaking, speeding up the computation will enable the motion planner to cope better with problems occurring in practice.

We have introduced a new approach to parallel motion planning for industrial robot arms with 6 DOF (Wurll et al., 1998). The algorithm works in an implicit and discretisized C-space and collisions are detected by distance computation in the Cartesian workspace. This avoids the time and memory consuming obstacle transformation and free C-space calculation and, thus, enables the motion planner to work reasonably fast.

In order to make the motion planning component faster, it has been subdivided into parallel subcomponents (see Figure 2b). To provide these subcomponents with work, the search domain (C-space) is decomposed into blocks, which are cyclically mapped onto the subcomponents. For searching in a (implicit) local C-space block, we apply the well known A*-search algorithm. The main task of the A*-algorithm consists of the expansion and the processing of configurations, which are saved in a priority list. Collisions are detected by a fast, hierarchical distance computation in the 3D workspace, based on the given CAD model of the environment and the robot (Henrich and Cheng, 1992)

6 Related Work

Much work has been done in designing and implementing robot systems but most of the presented systems have special control architectures for solving special problems. Distributed architectures using explicit parallelism at the high level control are especially rare. We will therefore review some of the approaches which deal with tasks and problems similar to ours or which use design principles we consider to be important.

Distributed high level robot control systems make use of functional parallelism by dividing the system into functionally different modules which run on different PEs. The system described in (Bar-On et al., 1993), for example, achieves high efficiency by concentrating on large and efficient software modules with little communication. Those systems typically provide extensible inter-processor communication bandwidth. In the following, some classes will be distinguished:

Shared memory systems communicate through a common memory. An efficient system which solves the dynamics of a robot and which is hierarchically organized is described in (Graham, 1989). Besides analyzing the different buffering strategies, this work investigates pipelining the data flow for achieving maximum parallelism.

Another possibility for running distributed robot task controllers which are independent from the network structure are the commonly used *distributed operating systems*. They have the advantage that basic communication and coordination functions are provided at the system level. Systems like DCE (Schill, 1993), therefore, provide a mechanism for *remote procedure calls*. With this simple form, only synchronous communication is supported and deadlocks are possible.

Even more convenient to program are *distributed object-oriented systems*. They offer all the possibilities of common distributed systems and support remote object access. Though often remote object access can be used asynchronously, the system performance still is dependent on the low level communication structure and protocol (Gaedke and Wicke, 1997). A fully object-oriented system, which is similar to our system, is reported in (Kim et al., 1997). A flexible software architecture was designed, which allows software to be easily adapted to system changes. In contrast to our approach, the system is hierarchically organized and the tasks are evaluated synchronously. Concepts in common are the following:

- Having a component called ”Workcell Manager” which orchestrates most of the cooperation activities with the other components
- Fixed cycle patterns and fixed programs for the interaction of the components solving a common task
- Interfaces of the components defined with a few simple and common operations, and components designed with as few assumptions about the cell as possible.

Another system architecture similar to our approach is the distributed simulation system presented in (Mehlhaus, 1994). Here, a distributed and flexible system divided into functional subsystems is suggested, which can be run in parallel. One main problem investigated is the temporal and functional consistency of the world model. The dataflow must correspond directly to the flow of the material. Temporal consistency is monitored and managed by a timer component. Unfortunately, the results show no remarkable speedup and require reengineering.

More loosely coupled system architectures are defined by the *multi-agent systems* (Längle, 1997) and especially the blackboard systems (Occello and Thomas, 1992). The subsystems called *agents* are of similar size to the components of our concept and also use asynchronous communication. But due to the fact that every agent negotiates for each task with the other agents, the system includes an unacceptable overhead. Such a system may work well for advanced manipulation tasks, like cooperating robot manipulators, but it is not efficient for industrial systems where the cycle times of the task are previously given.

7 Summary and Future Work

One promising method for mastering system complexity, such as a robot work cell, consists of breaking down the system into independent components. These components then can easily be mapped onto parallel PEs. The key issues of our system concept are a flat process communication hierarchy and a simple communication protocol, since the performance of parallel systems depends on both the number of PEs and the expense of communication. We therefore use rather closed and easily exchangeable components, which are driven by a high level cell control component. Communication between them is asynchronous and supports instruction chains. The high demands result from the aim to develop a robot cell for complex manipulation tasks, which force the robot control to be fast and to monitor the tasks carefully.

The main work so far has been in designing a parallel robot architecture. Future work will regard the way of implementing the proposed concept. This includes implementing both further components and the communication protocol. Then, a validation of the concept for a workcell for handling flexible material is regarded. Future implementations may also include an error detection and recovery component (Coulouris et al., 1994) which at least provides transitions into save error states and an on-line scheduling component in order to provide better system performance by redistributing the resources dynamically.

8 References

Bar-On D., Gershon D., Israeli A., Zuniga G. (1993). TRACK II: A multi-processor robot controller, In: *Proc. CompEuro Int. Conf. on Computers in Design, Manufacturing, and Production*, pp. 86-93, Prix-Erry, France.

Coulouris G., Dollimore J., Kindberg T. (1994). Distributed systems. Addison-Wesley.

Dilts D. M., Boy N. P., Whoirms H. H. (1991). The evolution of control architectures for automated manufacturing systems. In: *Jour. of Manufacturing Systems*, vol. 10.

Drachenfels H. (1997). Eine Definitionssprache für die Kommunikationsverbindungen in verteilten Automatisierungssystemen. In: *Informatik-Spektrum*, vol. 20, no. 5, pp. 286-293, Springer-Verlag, Germany.

Fayek R. E., Liscano R., Karam G. M. (1993). A system architecture for a mobile robot based on activities and a blackboard control unit. In: *IEEE Int. Conf. on Robotics and Automation*, vol. 2, pp. 267-274.

Fleury S., Herrb M., Chatila R. (1994). Design of a modular architecture for autonomous robots. In: *Proc. IEEE Int. Conf. on Robotics and Automation*, pp. 3508-3513.

Gaedke M., Wicke R. (1997). Softwaretechnik für Entwicklung, Betrieb und Wartung von Anwendungen im World-Wide Web. Master's Thesis, Telecooperation Office, University of Karlsruhe, Germany.

Geist A., Beguelin A., Dongarra J., Jiang W., Manchek R., Sunderam V. (1994). PVM: Parallel Virtual Machine, a user guide and tutorial for networked parallel computing. MIT-Press.

Graham J. H. (1989). Special computer architectures for robotics: Tutorial and survey. In: *IEEE Trans. on Robotics and Automation*, vol. 5, no. 5, pp. 543-554.

Gruhler G. (1993). Parallelverarbeitung von Anwenderprogrammen bei Industrierobotersteuerungen - Konzepte und Anwendungen. VDI-Bericht 1094, pp. 669-680.

Henrich D., Cheng X. (1992). Fast distance computation for on-line collision detection with multi-arm robots. *IEEE Int. Conf. on Robotics and Automation*, pp. 2514-2519, Nice, France, May 10.-15.

Henrich D., Höniger Th. (1997). Parallel processing approaches in robotics. In: *Proc. of the* IEEE *Int. Symp. on Industrial Electronics (ISIE'97)*, Guimaraes, Portugal, University of Minho, 7-11 July, pp 702-707.

Hörmann A. (1989). Steuerung und Systemarchitektur von fortgeschrittenen autonomen Systemen. *Robotersysteme 5*, Springer-Verlag, pp. 173-185.

Hwong Y. K., Ahuja N. (1992). Gross motion planning – A survey, *ACM Computing Surveys*, vol. 24, no 3, pp. 219-291.

Kamal L., Gupta K., del Pobil, A.P. (1996). Practical motion planning in robotics: current approaches and future directions. *IEEE Robotics & Automation Magazine*, December.

Kim Y., Jo J.-Y., Velasco V.B., Barendt N.A., Podgurski A., Ozsoyoglu G., Merat F.L. (1997). A flexible software architecture for agile manufacturing, In*: Proc. Int. Conf. on Robotics and Automation*, pp. 3043-3047, Albuquerque, USA.

Längle T.W. (1997). Verteiltes Steuerungskonzept für komplexe inhomogene Robotersysteme. PhD. Thesis, University of Karlstuhe, *VDI-Fortschritts-berichte Nr. 8/614*, VDI-Verlag, Düsseldorf, Germany.

Mehlhaus U. (1994). Verteilte Programmierung zur Integration von Simulation und Steuerung von Robotern. Reprinted in *VDI Fortschrittsberichte* vol 10.

Occello M., Thomas M.-C. (1992). Intelligent control in robotics through real-time distributed blackboards, In IMACS/SICE *Int. Symp. on Robotics, Mechatronics and Manufacturing Systems'92*, pp. 567-572, Kobe, Japan.

Schill A. (1993). DCE - Das OSF Distributed Computing Environment. Springer-Verlag.

Whitcomb L.L, Koditschek D.E. (1996). Robot control in message passing environment: Theoretical questions and preliminary experiments. In: *Proc. IEEE Int. Conf. on Robotics and Automation*, pp. 1198-1203, MN, USA.

Wurll Ch., Henrich D. (1997). Ein Workstation-Cluster für paralleles Rechnen in Robotik-Anwendungen. In: *Proc. of 4. ITG/GI-Fachtagung, Arbeitsplatz-Rechensysteme (APS'97)*, pp. 187-197, Koblenz-Landau, Germany, May 21.-22.

Wurll Ch., Henrich D., Wörn H. (1998). Parallel on-line motion planning for industrial robots. In: *Proc. of Robotics 98: The Third ASCE Speciality Conf. on Robotics for Challenging Environments,* Albuquerque, New Mexico, April 26.-30., 1998