Lueth, T.; J. Hellqvist (1996): Distributing Real-Time Control Tasks Among Multi Agent Systems. ISIAC Int'l. Symp. on Intelligence and Automation Control, Montpellier, France, May, Robotic and Manufacturing Systems, TSI Press, 1996, pp. 477-482

DISTRIBUTING REAL-TIME CONTROL TASKS AMONG MULTI AGENT ROBOT SYSTEMS

TIM C. LUETH
FZI Research Center, Haid-und-Neu-Str. 10-14, D-76131 Karlsruhe, Germany, email: t.lueth@ieee.org

JOHAN HELLQVIST, THOMAS LAENGLE
IPR Institute for Real-Time Computer Systems and Robotics, U. of Karlsruhe, D-76128 Karlsruhe, Germany, email: laengle@ira.uka.de

ABSTRACT

A task performed by an autonomous robot system can be described as a network of event-based and time-based control loops. If a single task must be distributed to several individual robot systems, it is necessary to separate the network into independent but coupled subnets. In this paper, a robot operating system, CAIC (Cooperative Architecture for Intelligent Control), is described that supports the distribution and execution of control tasks, i.e., control networks to/on several robots.

KEYWORDS: Intelligent Control, event-/time-based control, distributed control, cooperative architectures, robot operating system CAIC

TASK SPECIFICATION

A task for an individual robot or a group of robots can be described as one (or a set of) condition-event Petri net (CEP) [1, 2, 3]. In most cases, there is not only one possible CEP for a task but there are several CEPs that make different use of the available resources. Even for a part of a CEP there are several alternative sub-CEPs. The condition for executing a transition can be

- a specific time, which is used for time based control [4],
- an internal state of the robot, which us used together with explicit environment models and event based control [5],
- an external environment state, which us used together with implicit environment models and event based control [6], or
- a mixture of the conditions above.

The mechanism that is used to determine and broadcast the execution condition can be implemented centralized but also decentralized near the executing components.

A CEP or a transition can be generated in advance or dynamically generated (selected, adapted etc.) during task execution. Fig. 1a shows a CEP for time based control of two independent actuators. A CEP interpreter is reading and interpreting the CEP and is waiting for external events. Then it gives a transition as command to an execution controller (EC). The execution controller itself may be an CEP interpreter or a closed control

system (Fig. 2a). Such a control system generally consists of modules for sensing (S), observation/modeling (O/M), control/planning (C/P) and execution control (EC) for an actuator.

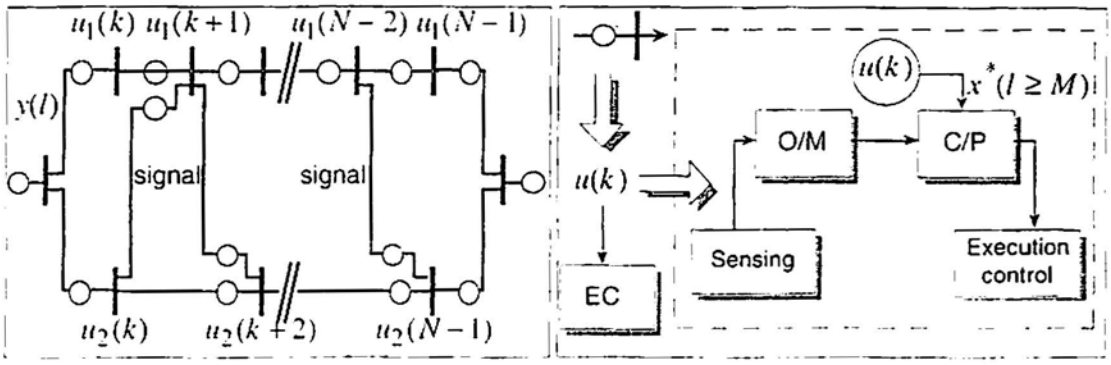

Figure 1. a) Task description by a condition-event Petri Net. b) A transition is a control net

CONTROL SYSTEM SPECIFICATION

An execution controller such as in Fig. 1b. can be described by the specification of both the control function or algorithm and the control architecture. An exact specification allows the implementation of an execution controller by using existing control modules (S, O/M, C/P, EC). Instead of sending a command to a known, i.e., previously specified and implemented, EC, it is also possible to install the EC during run-time when it's required.

An EC with a fixed control function has typically the structure shown in Fig. 2. The control modules are linked by the information flow and activated by the control flow. The example shows a planning module that activates the observation module first, which itself activates a sensor shot, generates a command for the following EC, and activates then the following EC.

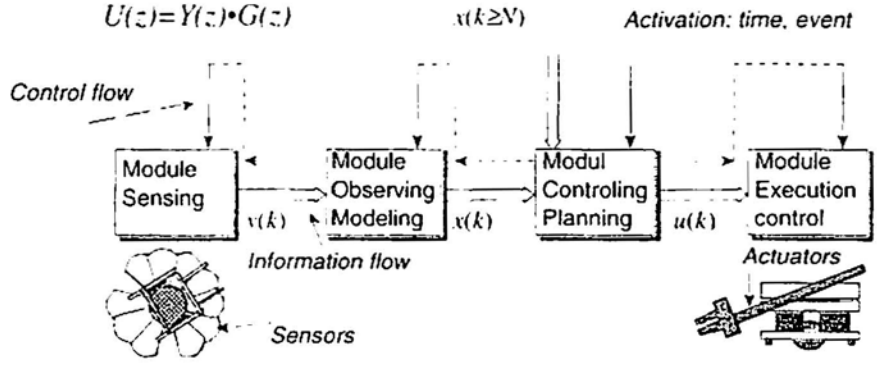

Figure 2. Information flow (double line) and control flow (dotted line) in control loops

The description of the control architecture must specify:

- whether a modules is activated by time, event or both,
- which events detected by whom will activate a module,
- which other modules must be activated before algorithm execution,
- which other modules must be activated after algorithm execution,
- which buffers, mail boxes, etc. are used for inter module communication,
- which buffers are used to change the own and other modules' architectures.

Fig. 3 shows simple control architectures for hierarchical control and concurrent behavior-based control.

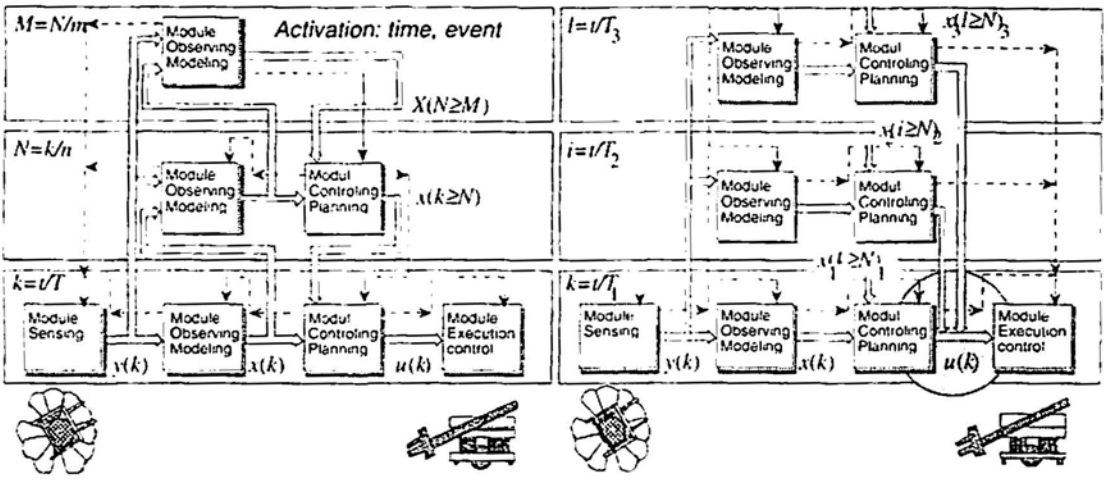

Figure 3. Examples for a) hierarchical control and b) behavior-based concurrent control .

CAIC • COOPERATIVE ARCHITECTURE FOR INTELLIGENT CONTROL

As pointed out, a task description may include an explicit specification of the control system (modules plus architecture) that is desired for task execution. If this is the case, it's possible to implement a desired control system during run-time on a robot's computer control system. The size of the overall operating control system depends on the available memory, computational power, and limited number of concurrent processes. This means, if there is capacity left in a robot's computer control system, a robot can take over additional tasks as long as there are not other physical resource conflicts. This idea has been published first in [7].

To dynamically install and change a robot control system, the CAIC (Cooperative Architecture for Intelligent Control) robot operating system has been developed in Karlsruhe. CAIC is an extension for existing real-time operating systems. It allows the flexible compilation of encapsulated control modules into a running control system [8, 9]. Furthermore, control modules can propagate the need for minimal or maximal information flow frequencies to preceding control modules.

In CAIC, all control algorithms are encapsulated into a module frame that work similarly to an actor [10]. A central round-robin scheduler switches fast between the individual control modules. Each module itself checks its activation conditions and suspends if the time or event for activation has not reached yet. This means, the modules perform a cooperative scheduling with statistical jitters. The modules can detect constant activation delays, which correspond to overloading of the computer. Overload occurs, if several complicated computations are performed concurrently. In this case, the task switching frequency of the round-robin scheduler limits the minimal activation frequency.

The modules communicate by reading and writing into buffers. Each buffer can be implemented as a size-one or as a multi-buffer, i.e., a queue. Modules and buffers have identifying names. Task switching is delayed during buffer-write operations.

Each module is using a previously defined control-in-buffer and control-out-buffer. The control-in-buffer of a module is used by other module to announce the request for changing the module's activation frequency or condition. The same buffer is also used to

check in and check out the request of a link to a module's output buffer. A module continuously reads and interprets its own control-in-buffer and reacts to the requests. If no other module is checked in any more, the module terminates itself and removes its buffers after a delay that corresponds to its installation time. The control-out-buffer of a module is used to announce changes of its activation frequency or condition. Furthermore, it is used to announce detected activation delays, when they are bigger than a predefined tolerance. Fig. 4a shows an example for a module frame.

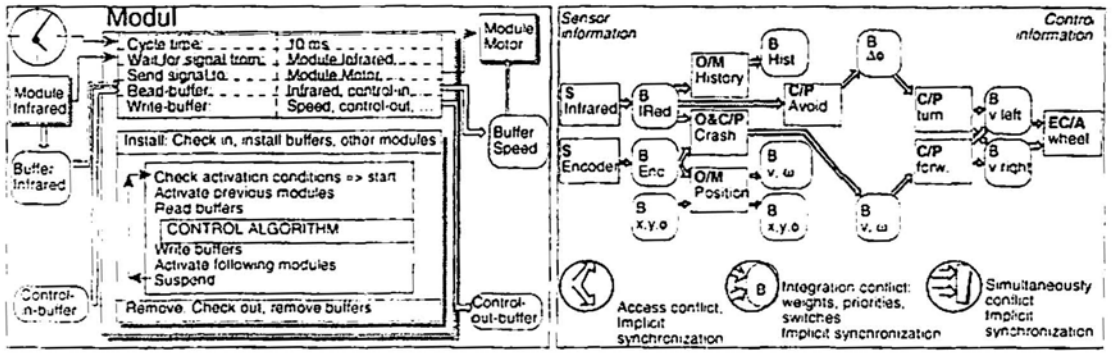

Figure 4. a) Example for a module (actor) frame b) Example of a concurrent control network

Fig. 4b (control flow is not included) shows a part of a simple control network used for the robots in the experiments. Nine modules operate in parallel to process the sensor information and generate the required wheel speed for moving around and avoiding collisions with other objects. The individual modules can have different cycle times and activate each other depending on the architecture specification. Independent on the control flow, the information flow adds some constraints regarding the module synchronization and information access:

1 Since discrete information units are exchanged, the information generating module will delay the information processing module.
2 If two or more information generating modules write into the same buffer, the problem of information integration arises, even if a queue-type buffer is used.
3 If a module is reading and processing information of two or more buffer, the problem of simultaneously reading and processing must be mastered.
4 Dynamic changes of the architecture during run-time, require the capability to manage the problems. Therefor, the specification of the architecture must include an optional a description how to deal with occurring synchronization and access problems.

The CAIC robot operating shell itself is continuously running as a module and interprets the robot's central control-in-buffer. It is responsible for installing new modules on external request. Furthermore, it distributes incoming messages of external control modules, which are active on other robots, to the appropriate modules of the local robot. In the future, the shell will be also responsible for moving operating modules or control nets to other robots' CAIC systems.

EXPERIMENTS

In the experiments, CAIC has been used to dynamically install, operate, and terminate control networks. Furthermore, the individual control modules use CAIC during run-time to synchronize themselves with other control modules, to suspend, delay or accelerate control subnets, to interrupt and install links between event-generating and event-processing modules, etc.

The whole control system of a robot is described by a set of control modules and the mechanisms that are used to link the modules. CAIC supports the mapping of the network description to an operating control system and allows also dynamic changes during run-time.

During the experiments, for instance, control loops such as obstacle avoidance were changed from a continuous time-based mode to a triggered event-based mode in which control is activated only if sensor information is available.

By using CAIC, assembly operations for the *Cranfield Assembly Benchmark* have been implemented for the small Khepera robots. The robots [11] are able to *search* for objects, to *distinguish* objects, to *move relatively* to objects, to *grasp* assembly parts (Fig. 5a), and even to perform tasks like *insert* assembly parts (Fig. 5b) into a fixture. The exact description of the capabilities has been published in [8].

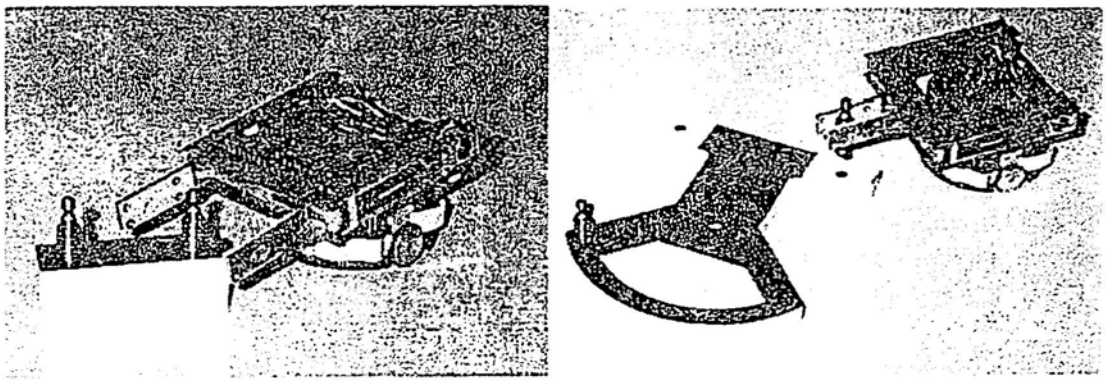

Figure 5. a) Grasping of a "spacer" and b) inserting the "spacer" into a "side plate"

For coupling real-time control subnets running on several individual robots' CAIC system, a local communication system is required. Such a communication system has been developed and integrated into the CAIC system [9] for the Khepera robots. By using the local communication system, it was possible to combine independent control systems. The main difference in comparison with other real-time operating system such as VxWorks or LynxOS, is the capability to couple control systems based on physical relations.

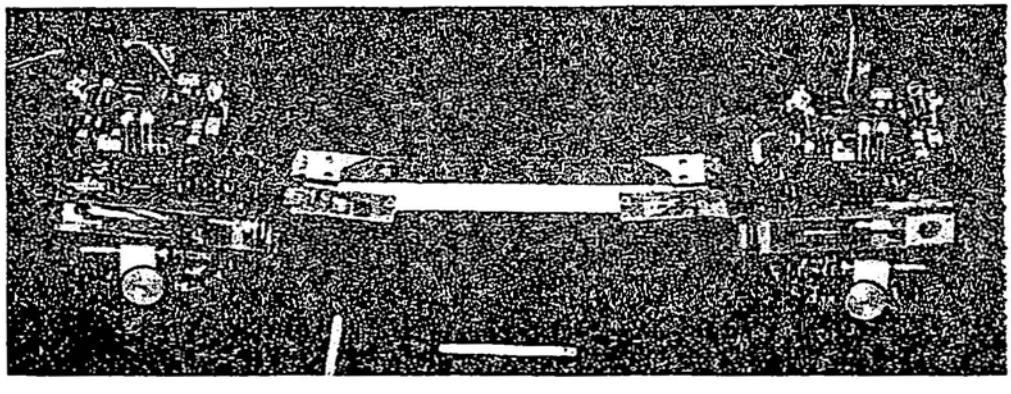

Figure 6. Grasping and transporting a "spacing piece" by control network coupling

The new CAIC capability was used to implement a grasping and transportation task in a closed kinematic chain of two robots (Fig. 6). During execution, the robots exchange in a dynamic master-slave configuration their speed, gripper angle, etc. The next step is to locally measure the applied force in each robot by evaluating the counter of the wheel-

speed's PID-controller. Then, not only master-slave configurations but also coupling of independent control system will be possible.

CONCLUSION

A task performed by an autonomous robot system can be described as a network of event-based and time-based control loops. The control loops consist of control modules that are linked by information flow between the modules and control flow for the activation of the modules. Different tasks may require the same control modules but a different linkage between the modules. The CAIC system supports the flexible linkage of robot control modules. By using a global or local communication system CAIC allows also the linkage of control modules running on different robot control processors. By using robot operating systems such as CAIC, the development of robot control systems becomes more flexible and allows a better description of the actual control architecture of a control system. Experiments show that CAIC is a powerful tool for control system design. Further research will show which features of a robot operating system have to be added to the current status.

ACKNOWLEDGMENT

This research is being performed at FZI • Forschungszentrum Informatik, Division TE&R • Robotics, and at IPR • Institute for Real-Time Computer Systems and Robotics (Prof. Dr.-Ing. U. Rembold, Prof. Dr.-Ing. R. Dillmann), University of Karlsruhe. Thanks to Ronald Grasman for designing the FZI's local infrared communication system.

REFERENCES

1. Hoermann, A.: A Petri Net Based Control Architecture for a Multi-Robot System. ISIC IEEE Int. Symp. on Intelligent Control. Albany, USA, 1989.
2. Freedman, P.: Time, Petri nets, and Robotics. IEEE Trans. on Robotics and Automation, 7, 4 (1991), pp. 417-433.
3. Gottschlich, S.: C. Ramos, D. Lyons: Assembly and Task Planning - A Taxonomy. IEEE Robotics and Automation Magazine, 1, 3 (1994), pp. 4-12.
3. Musliner, D.J., E.H. Durfee, K.G. Shin: CIRCA: A Cooperative Intelligent Real Time Control Architecture. IEEE Trans. on System Man and Cybernetics, 23,6 (1993), pp. 1561-1574.
4. Iserman, R.: Digital Control Systems, Vol 1., Springer-Verlag, 1989.
5. Sobh, T.M.; K. Valavanis, D. Gracanin, J. Owen: A Subject-Indexed Bibliography of Discrete Event Dynamic Systems. IEEE Robotics and Automation Magazine, 1, 2 (1994), pp. 14-20.
6. Kosecka, J., R. Bajcsy: Discrete Event Systems for Autonomous Mobile Agents. Robotics and Autonomous Systems, 12 (3&4) (1994), pp. 187-198.
7. Musliner, D.J., E.H. Durfee, K.G. Shin: CIRCA: A Cooperative Intelligent Real Time Control Architecture. IEEE Trans. on System Man and Cybernetics, 23,6 (1993), pp. 1561-1574.
8. Lueth, T.; Th. Laengle, J. Heinzman: Dynamic Task Mapping for Real-Time Controller of Distributed Cooperative Robot Systems. IFAC WS Distributed Computer Control Systems, Toulouse-Blagnac, France, September, 1995, pp. 37-42.
9. Lueth, T.C.; R. Grasman, Th. Laengle, J. Wang: Cooperation Among Distributed Controlled Robots Using Local Interaction Protocols. ISRAM Int'l. Symp. on Robotics and Manufacturing, Montpellier, France, May, 1996.
10. Agha, G.; C. Hewitt: Concurrent Programming Using Actors. In Object-Oriented Concurrent Programming, MIT Press, 1987.
11. K-Team: Khepera User Manual, Ver. 3.0, LAMI-EPFL, Lausanne, Swiss, 1994.