A distributed implementation of a mode switching control program

Holdgaard, Michael; Eriksen, Thomas Juul; Ravn, Anders P.; Andersen, Torben Ole

Published in:
Proceedings of the Seventh Euromicro Workshop on Real-Time Systems

Link to article, DOI:
10.1109/EMWRTS.1995.514307

Publication date:
1995

Document Version
Publisher's PDF, also known as Version of record

A distributed implementation of a mode switching control program*

Michael Holdgaard, Thomas J. Eriksen, Anders P. Ravn†
Department of Computer Science, bldg. 344

Torben O. Andersen
Institute of Control Engineering, bldg. 424

Technical University of Denmark
DK-2800 Lyngby, Denmark
Email: apr@id.dtu.dk
Fax: +45 42 88 45 30

Abstract

A distributed implementation of a mode switched control program for a robot is described. The design of the control program is given by a set of real-time automatons. One of them plans a schedule for switching between a fixed set of control functions, another dispatches the control functions according to the schedule, and a final one monitors the system for exceptions that shall lead to a halt.

The implementation uses four transputers with a distribution of phases of the automatons over the individual processors. The main technical result of the paper is calculations that illustrate how to justify that the implementation meets real-time constraints.

1 Introduction

A promising paradigm for the control of complex dynamical systems is to use an automaton that during the activation of the system switches between a number of reasonably simple control algorithms. Such a system is hybrid [4] because it combines continuous states of the plant with discrete states of the automaton. The area of hybrid systems is still very young and raises many questions about the mathematical properties of such controllers, and also how to implement them, In order to investigate these problems we have experimented with hybrid control. The preliminary results indicates that more precise plant control can be achieved than with state of the art model based or adaptive control algorithms alone. The plant in question is a hydraulically powered dual axes robot (a continuous path manipulator), as used for instance in grinding and arc welding. It is an experimental facility for investigating the problems arising in the digital control of oil hydraulic systems [3]. The instrumentation includes sensors for the measurement of various dynamic variables and some on/off switches. Each cylinder can yield a static force up to 20 kN. With maximum pay-load there is still adequate power to obtain tool center point velocities around 3.5 m/s. In this situation the centrifugal and Coriolis couplings as well as the gravity force and change of inertia moments implies a non-linear and coupled dynamic relation. Furthermore, the flow pressure relations in the valves are non-linear: the same control input signal gives a different response for different arm positions. These characteristics motivate the hybrid control approach.

The system is controlled through a network of 4 transputers. Two T225 without floating point directly connected to interface electronics for each of the two separate joints, and two T805 processors with floating point capability, where one is placed in a PC-based development system.

A design in terms of timed automatons within a chosen architecture is outlined in Section 2 below. The challenge is then to implement this architecture as a collection of communicating processes on the hardware platform. The approach taken and the resulting implementation is given in Section 3. The conditions for the implementation to preserve the real-time constraints of the design are given in Section 4 that relies on the general approach explored by the ProCoS project [1].

2 The controller design

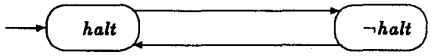
The architecture of the implemented system is based on the concept of a multi-layered control system as defined in [6, 5]. It consists from top to bottom (closest to the plant) of the following layers:

Analysis layer: performs dynamic selection of the control tasks.

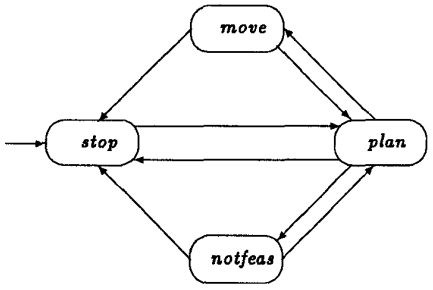
Rule layer: determines the control algorithms for a given task.

*Supported by the Danish Technical Research Council under the Co-design and IMCIA projects.

†Currently Visiting Professor at Institut für Informatik und Praktische Mathematik, Christian-Albrechts-Universität zu Kiel, Germany.



Monitor



Planner

Figure 1: Monitor and Planning Automatons

Process layer: executes the current control algorithm including analog/digital and digital/analog conversions.

The layers interact through state variables that denote functions of time. For discrete states, the rate of change increases while the complexity of the value range decreases from top to bottom. Thus, at the bottom, a state value is a simple scalar representing a setpoint or a measurement, and the states change in the order of milliseconds. At the rule layer, a state value is a sequence of control algorithm identifiers and start and stop times. Such a state will change only in the order of 100 milliseconds. Finally, the analysis layer is given full descriptions of desired trajectories and information about major events that shall lead to a new control task. This layer also monitors safety-critical conditions.

The design is given by phase automatons, shown in Figure 1. The analysis layer is a two phase automaton called *Monitor*. It changes between a *halt* and ¬*halt* phase dependent on whether the safety conditions are satisfied. The rule layer is given by a four phase *Planner* automaton, that enters *stop* whenever the analysis is in *halt*, and otherwise proceeds to a preplanning *plan* phase from where it either enters a *move* phase when a *schedule* of control algorithms is found, or a *notfeas* phase if none can be found. In the *move* phase, the schedule may be improved dynamically.

3 The implementation

The implemented components are a collection of occam processes, distributed over the transputers as illustrated in Figure 2.

Analysis layer

We have chosen to consider the input trajectory a parameter for a given "run" of the system. The implementation of the analysis layer then becomes rather simple as only the safety critical conditions are to be monitored and reacted on. The implementation consists of only one process:

Monitor Determines on the basis of data received from the process layer within every period of sampling whether or not the robot should be halted. The data is received on the channel *PLANT.ENVIR*. In case of a change in safety critical conditions a flag is communicated to the rule layer through the channel *HALT*.

Rule layer

In the implementation of the rule layer we do not consider dynamic improvement of a schedule. The layer consists of three process:

Coordinator A coordinating process which - by the use of a state with a value domain directly corresponding to (the names of) the phases - ensures correct serialization of the phases of *Planner*. In case a schedule for the trajectory can be found the schedule is send down to the process layer on the channel *COORD.2.SC*. Coordinator reacts on input from the analysis layer on the channel *HALT* and in case the robot needs to be halted, Coordinator interrupts the possible execution of a schedule in the process layer on the channel *COORD.2.SC*.

Planner Determines whether or not a schedule of control algorithms can be found for the trajectory. The planning of a schedule may involve iterative time consuming calculations which is why Planner is a separate process instead of just a subroutine called from Coordinator.

User A simple interface to the host system. Reads the trajectory from a host file and enables the writing of results to the host. The results are received from the analysis layer on the channel *MON.2.USER*.

Process layer

The task of the process layer is to perform the actual switching between the control algorithms in the schedule. Thus major considerations in implementing the layer are to decide what is meant by *switching* and what information will be required and sufficient to perform a switch. A schedule is a list of *events*, where each event consists of:

MP Identifier for a control algorithm (a motion planner)

T_{Act} Activation time of motion planner.

T_{Out} Output enabling time (time of actual switch).

Info Trajectory information.

The rationale for operating with the two time points is firstly that it facilitates for instance the use of so called learning controllers and secondly that it ensures that the transmission of data to a motion planner will not delay the actual switch to it.

The intuitive processing of a schedule is to regard it as a queue of jobs to be executed:

```
WHILE TRUE
  ALT
    coord.2.sc ? event
      enq(event) -- receive new event.
    NOT empty() & clock ? AFTER Tout()
      SEQ
        e:= deq()
        -- perform actual switching.
:
```

where Tout() returns the value of the T_{Out}-field of the first event in the queue. The actual processing is more complicated:

Scheduler Scheduler is in charge of maintaining the event queue. By the use of an occam-timer Scheduler makes sure that the activation and switching times mentioned in the events (the T_{Act} and T_{Out} fields respectively) are respected. At the time of activation of a motion planner Scheduler transmits data to the motion planner and notifies Dispatcher. At the time of output enabling the front event Scheduler *deq*s the event and notifies Dispatcher that the motion planner is to be enabled for output.

Dispatcher Dispatcher maintains an array of flags - one for each of the motion planners. The flag of a motion planner is set just when it is active. At sampling time Dispatcher receives measurements from Estimator and sends them to the active motion planners. Dispatcher also maintains the identity of the output enabled motion planner. Whenever a motion planner is to be activated or enabled for output Dispatcher updates its state.

Estimator Receives measured data from the hardware interface to the robot (Lower.- and Upper.DATS) and sends it on to Dispatcher as well as to the analysis layer.

Effector Receives calculated data from one of the MP processes and sends it on to the hardware interface to the robot (Lower.- and Upper.DATS).

MP A motion planner. An MP is simply a process that can be activated and en-/disabled for output in the sense explained earlier on. We do not assume anything else about it, especially not about how an MP handles the data it receives at activation time and how the calculations are carried out. Hence an MP could be anything from a process which constantly outputs a null value when enabled for output (useful for halting the robot) to a process which might perform heavy calculations and on-line change of parameters while enabled for output.

Lower.-/Upper.DATS The hardware interface to the robot. These processes are in charge of the sampling rate of the control system, converting the input from the A/D channels (i.e. the sensors) and converting the output to the D/A channels (i.e. the actuators). Lower./Upper.DATS is concerned with the lower/upper axis of the robot only. They utilize facilities provided by the hardware (called DATS[1]) to synchronize at sampling time.

The configuration of the processes is shown in Figure 2. The robot is equipped with two actuators and as shown in the figure we have dedicated a Scheduler and a Dispatcher to each of the actuators along with each their set of motion planner (the sets are most likely not equal). This division has the consequence that the whole process layer is divided into separate parts, one for each actuator. Since the interface to the other layers of the system is simple and well-defined, each part of the process layer could be placed on separate transputers and would then be a part of the actuator more than a part of an overall control system.

4 Checking Real-Time constraints

The critical constraint is the sample time for the motion planners. If the implementation was simple with one motion planner on each DATS, the time would be

$$T_{simple} = t_r + t_c + t_w$$

where t_r is the time to read sensors, t_c is the computation time for the control algorithm and t_w is the write time to the actuator. Typically, t_r and t_w is less than $20\mu s$ leaving sample time for computation with a sampling time in the 1 ms range.

We shall now estimate the corresponding cycle time for the actual implementation with dynamic scheduling. The estimate is pessimistic and uses some simple rules as seen in [8, 9].

The cycle time consists of: DATS time (t_{DATS}), time in Estimator (t_E), in Dispatchers (t_D), and in planners (t_{MP}). Furthermore, there is an overhead because Schedulers and Communicators may interfere (t_S).

The DATS are physically parallel, but readings are serialized by Estimator and Effector

$$t_{DATS} = t_r + 2 \cdot c + t_w + 2 \cdot c$$

where c is a communication time for a reasonably short message ($10\mu s$). The factor 2 is caused by a delay of c due to the serialization in Estimator and Effector.

The Estimator communicates serially to two Dispatchers, and the Monitor. We assume the Monitor to be the only high priority process on its board, thus it has no delay.

[1] Short for: Data Acquisition Transputer Subsystem

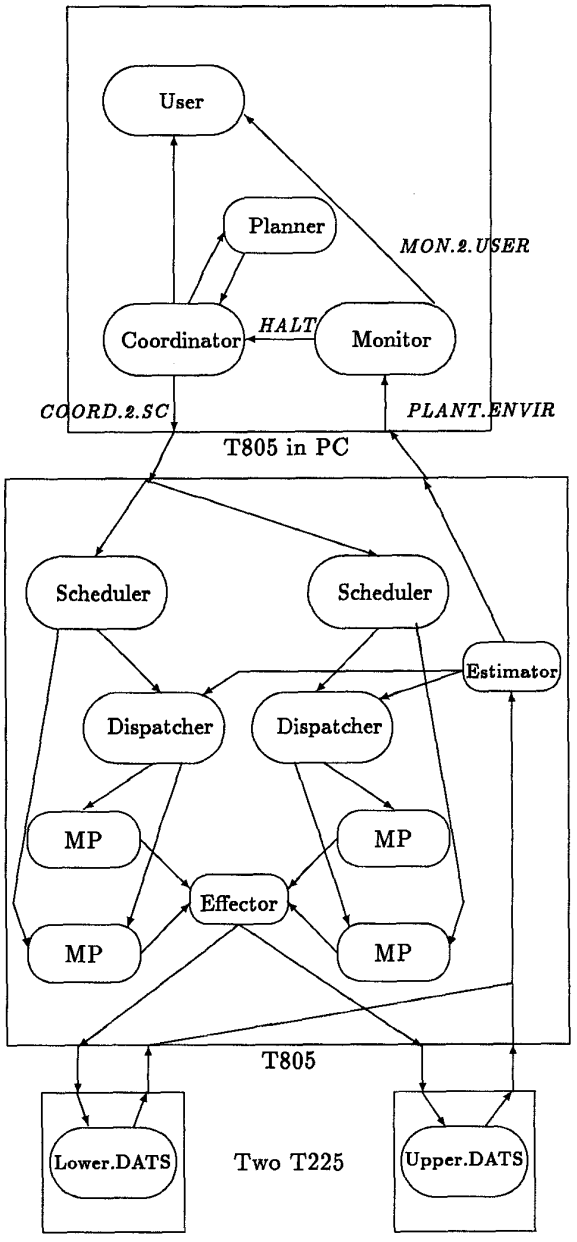


Figure 2: The implemented components and the transputer configuration.

$$t_E = 3 \cdot c$$

Each Dispatcher shall communicate the values to active MP's (assume two).

$$t_D = 2 \cdot c$$

The planners do the control algorithm. The output is to the estimator.

$$t_{MP} = t_c + c$$

The estimator has been added to the DATS.

Multiprogramming means that if the Estimator, two Dispatchers and four planners were the only active processes we would have a cycle time of:

$$\begin{aligned} T_{MIN} &= t_E + 2 \cdot t_D + 4 \cdot t_{MP} + t_{DATS} - \\ &\quad (14 \cdot c + t_r + t_w + 4 \cdot t_c) \end{aligned}$$

This would also be the case if these were high priority processes on the board ($t_S = 0$). Otherwise, we must assume that the 3 communication and scheduler processes get their fair turn. Since care has been taken to make them communicate often by breaking event-lists into smaller pieces in the interfaces, it is reasonable to think that the processes are equally. Thus T_{MAX} is defined by:

$$\begin{aligned} T_{MAX} &= \frac{\textit{processes in cycle}}{\textit{total processes}} \cdot T_{MIN} \\ &= \frac{8}{11} \cdot T_{MIN} \end{aligned}$$

The calculations above demonstrates that there is an acceptable overhead ($14 \cdot c$) on the dynamic scheduling, provided that the number of active planners are kept low ($3 \cdot t_c$), and provided that the scheduler is given lower priority than the real time tasks. An improvement would be to place planners in the DATS assemblies. This would reduce the computation part of the overhead to a minimum.

5 Conclusion

We have presented the considerations leading from a top level mode switching controller architecture to a distributed implementation. Inevitably, distribution incurs an overhead in communication, which may be partly or fully offset by multiprocessing. In order to check that real-time constraints are met, a calculation with estimates of communication, computation and multiprogramming delay times is performed. In [7] a compilation framework is outlined that allows the computation times to be checked automatically.

References

[1] J. P. Bowen, M. Fränzle, E.-R. Olderog and A. P. Ravn. Developing Correct Systems, In *Proc. 5th Euromicro Workshop on Real-Time Systems*, IEEE Computer Society Press, pages = 176–189, 1993.

[2] F. Conrad, P. E. Hansen, and T. O. Andersen. Design and evaluation of adaptive controllers for hydraulic robots. In *Proc. 2nd Biennial Europ. Joint Conf. on Engineering Systems Design and Analysis*, volume 8-Part B, 1994. London, England, 4-7 July.

[3] F. Conrad et al. On mechanical design and digital adaptive control of the fast tud-hydralic test robot manipulator. In *ASME WAM'91*, volume 91-WA-FPST-9. American Society of Mechanical Engineering, 1991.

[4] R. L. Grossman, A. Nerode, A. P. Ravn, and H. Rischel, editors. *Hybrid Systems*, volume 736 of *LNCS*, 1993.

[5] S. Nadjm-Tehrani and J.-E. Strömberg. From physical modelling to compositional models of hybrid systems. In H. Langmaack, W.-P. de Roever, and J. Vytopil, editors, *Formal Techniques in Real-Time and Fault-Tolerant Systems*, volume 863 of *LNCS*, pages 583–604, 1994.

[6] Simin Nadjm-Tehrani. *Reactive Systems in Physical Environments*. PhD thesis, Dept. Comp. and Inf. Science, Linköping University, Sweden, May 1994. Linköping Studies in Science and Technology, Dissertation no. 338.

[7] J. He et al. Provably Correct Systems. In H. Langmaack, W.-P. de Roever, and J. Vytopil, editors, *Formal Techniques in Real-Time and Fault-Tolerant Systems*, volume 863 of *LNCS*, pages 288–335, 1994.

[8] A. P. Ravn, H. Rischel and H. H. Løvengreen. A Design Method for Embedded Software Systems. BIT 28, 427–438, 1988.

[9] C. Fidge. Real-Time Refinement. In J. C. P. Woodcock and P. G. Larsen, editors, *FME '93: Industrial-Strength Formal Methods*, volume 670 of *LNCS*, pages 314–331, 1993.