'90 KACC 1990. 10. 26~27

Design of an Embedded Intelligent Controller

Hiromitsu Shirakawa, Tsunetoshi Hayashi and Yutaka Ohno

Department of Computer Science and
Systems Engineering
Ritsumeikan University
Kita-ku, Kyoto, 603 Japan

Abstract

There is an increasing need to apply artificial intelligence to the real application fields of industry. These include an intelligent process control, an expert machine and a diagnostic and/or maintenance machine. These applications are implemented in AI Languages. It is commonly recognized that AI Languages, such as Common Lisp or Prolog, require a workstation. This is mainly due to the fact that both languages need a large amount of memory space and disk storage space. Workstations are appropriate for a laboratory or office environment. However, they are too bulky to use in the real application fields of industry or business. Also users who apply artificial intelligence to these fields wish to have their own operating systems. We propose a new design method of an intelligent controller which is embedded within equipment and provides easy-to-use tools for artificial intelligence applications. In this paper we describe the new design method of a VMEbus based intelligent controller for artificial intelligence applications and a small operating system which supports Common Lisp and Prolog.

1. Introduction

Icon is the name of an embedded intelligent controller specifically designed for artificial intelligence applications. **Icon** has these characteristics.

- It supports Common Lisp and Prolog.
- These languages and operating system are built into ROM for fast startup.
- It has an open operating system, any part of which is reprogrammable by users.
- It is a compact computer which may be embedded within a control equipment.

Common Lisp was chosen as it has become the de facto standard. Prolog was also chosen as it is well suited for artificial intelligence. We chose KCL (Kyoto Common Lisp)[1] and IF/Prolog[1] because porting those programming languages to the intended machine is easy and their quality is high. KCL and IF/Prolog are written in C language and run on various workstations based on the Motorola 68020 and 68030. **Icon** uses the Motorola 68030. Also all built-in predicates of IF/Prolog are the same as described in the textbook [2]. Also full screen editor EZ [3] is supported.

An embedded controller specifically designed for artificial intelligence applications needs a user-friendly operating system which supports storage, retrieval, viewing, manipulation and communication of knowledge. Operating systems tend to be intricate and intractable. The cost of commercial operating systems is increasing faster than the cost of hardware. However, controller designers desire to modify operating systems for specific applications. The operating system of **Icon** provides easy-to-use tools suited for artificial intelligence applications.

The characteristics of the operating system are that it employs threads and tasks for realizing lightweight processes [4], and use the mutex and the conditional variables for synchronization. And interprocess communication facility using ports is provided. This operating system supports real-time processing which is necessary in the embedded controller. The operating system is written in a programming language gcc.

2. System Organization

We chose an MC68030 based CPU board providing as much as 4Mbyte RAM and 4Mbyte EPROM for **Icon**. The board is build for the VMEbus/IEEE 1014 standard. An earlier version of **Icon** was an MC68020 based machine. It was called SLIM[5]. We chose FORCE Computers' VMEbus products [6] because it supports a large amount of EPROM required for storing KCL and IF/Prolog. **Icon** supports KCL but do not support IF/Prolog now. However, SLIM supports IF/Prolog[2].

The applications of embedded controller are an intelligent process control, an expert machine and a diagnostic and/or maintenance machine for factories. **Icon** is used in industry. Typical application of **Icon** is described in section 4. Figure 1 shows **Icon**.

3. Operating System

The operating system is intended for control process applications. The operating system has several important features:

- Lightweight processes using threads,
- Efficient interprocess communication primitives,
- Minimal kernel,
- Openness,
- Flexible control flow,
- ROMable kernel,
- UNIX compatible interface,
- Real-time processing.

[1] IF/Prolog is a trade mark of InterFace Computer GmbH.

[2] We have a plan to port IF/Prolog.

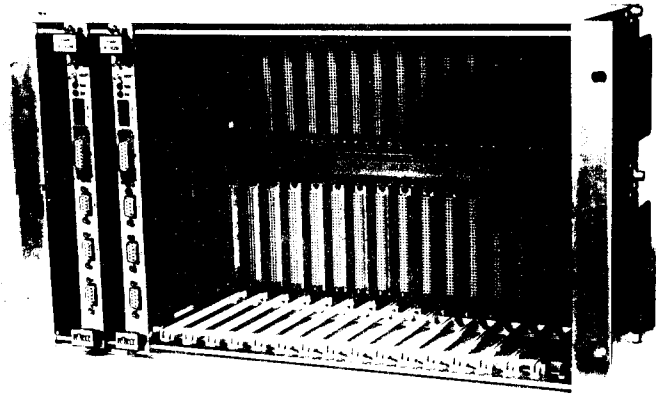

Figure 1. **Icon** system.

Theta (Threads and Tasks) is the name of the operating system built to support a distributed and parallel computing environment. Mach has been designed for multiprocessors and workstations [4]. However, Theta has been implemented for small and embedded system.

Principal features of Theta are that most of the facilities in the conventional kernel functions are managed by system servers. Clients make requests for services by remote procedure calls. System servers emulate UNIX system calls. So user can call servers without software interrupt.

3.1 Tasks and threads

Task is an abstraction object for dealing with computer resources, i.e., I/O, CPU, memory, ports for interprocess communication, and threads. All threads within a task share the resources and a single address space. Conventional process corresponds to a single thread within a task. Therefore, the Theta scheduler maintains threads. However, the threads are scheduled by the user. Exceptions are also issued to independent threads. Thread data structure contains the thread descriptor. The thread descriptor contains a less amount of processor state information than that of the process of the conventional operating system. Lightweight processes can be realized by the low overhead of context switch and by sharing a single address space.

The states which a thread assumes are execute, ready and blocked. The threads in ready state are kept on the ready queue. The thread blocks if it is waiting a resource or message. Relinquishment of the CPU may be initiated either by the thread itself issuing thread_reschedule() or by two another occasions. First occasion is by interval timer. Second occasion is by waiting for interprocess communication. For each priority level the ready queue is maintained.

3.1.1 Library functions of tasks and threads

The following primitives handle the management of tasks and threads.

task_create,
task_detach,
task_self,
task_join,
thread_priority,
thread_detach,
thread_exit,
thread_kill,
thread_fork,
thread_join,
thread_name,
thread_self,
thread_reschedule.

3. 2 Interprocess communication

Interprocess communication of Theta means task to task communication. All threads executing within a task are equally eligible to communicate with other threads in other tasks. Intraprocess communication is usually done cheaply using shared memory within task.

Communication is based on message passing mechanisms. Messages are not directed to the task but to a port. Two tasks which want to communicate each other obtain a port with the same port identification. A sender may send message to the port, and receiver may receive the message through the port. A port is a data structure consisting of a message queue, a queue for blocked threads waiting the messages, and management information for message transmission.

A fixed-length 20 bytes data is employed as a message. Only nonblocking communication is implemented. Therefore, the sending thread does not block, however, the receiving thread blocks if there is no message.

Sending a message is performed by issuing msg_send() operation. The sender does not block even if receiving thread does not exist. Message sent by the sender is entered into the message queue associated with the port. Receipt of a message is done by msg_receive(). If there is no message in the queue when a thread issues a receive operation, then it blocks and is entered into the waiting queue associated with the port. Blocked thread may resume execution when a sender put a message into the message queue.

Blocked communication may be realized if rpc (remote procedure call) operation is used. A client issues msg_rpc() to the server and followed by the receive operation and waits for receiving reply from the server. The server issues receive operation to receive message from the client, and issues reply operation.

3.2.1 Library functions of interprocess communication

The eight library functions are provided for interprocess communication.

msg_rpc,
msg_rpc_adr,
msg_send,
msg_send_adr,
msg_receive,

msg_reply,
task_port,
thread_port.

3.3 Synchronization

Mechanisms for synchronization of threads concurrently accessing shared data and resources are implemented by combination of mutex and condition variables [7, 8, 9].

3.3.1 Mutex

Mutex is an only facility doing mutual exclusion in Theta. Mutex_lock() returns immediately upon locking the mutex. If the mutex has already been locked by some other thread, the thread relinquishes the CPU. The thread is returned to the end of queue to be serviced at a later time. When the thread acquires the CPU again, it tries to lock the mutex. This continues until it locks the mutex. So this busy-waiting phenomena is usually called spin-lock.

The following primitives to handle the mutex are supported in Theta.

mutex_lock,
mutex_unlock,
mutex_allocate,
mutex_deallocate.

3.3.2 Condition variable

Sometimes threads are captured by a mutex and may keep staying in spin-lock state for long time. Condition variable can help to release such the threads. Suppose a thread tries to lock the mutex to access shared data or resources. If it cannot access and executes condition_wait(), then it unlocks the mutex and blocks itself. This means that the thread is placed at the back of the queue associated with the condition variable. If other thread issues condition_signal(), then one of the thread blocked is unblocked.

The five primitives concerning the condition variable are provided.

condition_wait,
condition_signal,
condition_broadcast,
condition_allocate,
condition_deallocate.

3.4 System servers

Most facilities within the kernel of the conventional operating system are implemented by system servers in Theta. Only remaining function is a dispatcher which performs a context switch. Client sends a message to the system server. The most outstanding feature of Theta is that even the scheduler is realized by a system server.

Thus minimal kernel may be realized. This makes it easy to prepare schedulers as users need. Also interrupt disable time is only while dispatcher is running. So it minimizes the time of disabling interrupt. This plays a crucial role in realizing realtime operating system.

Followings are brief illustrations of system servers. Figure 2 shows the relationship of the kernel and the system servers.

3.4.1 Memory server

Memory server manages the memory allocation and reclamation of the memory.

3.4.2 Task server

Task server is involved in creation and termination of tasks and threads. The thread creation is required to prepare pre-

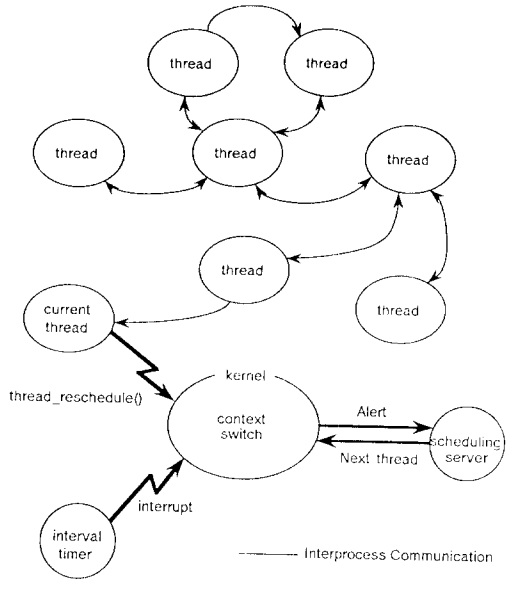

Figure 2. Kernel and system servers.

allocation of memory space. At system startup, several skeletons of thread descriptor are linked to a free queue in advance by the memory server. New thread may be created immediately using the skeleton. Terminated thread descriptor is effectively linked to the queue.

3.4.3 Port server

The port server provides the services of allocation of a port and freeing of a port. This port server pre-allocates necessary skeleton of port in advance.

3.4.4 Name server

Name server maps a name onto an address in that node or in other node. The addresses are registered in a table.

3.4.5 Input and output server

The operating system of Theta uses the concept of streams to describe the I/O functions [10, 11]. The scheme of streams provides a unified treatment of input and output operations.

The characteristics of the streams are as follows:

- Modularity and flexibility in input and output operations,
- Uniform treatment of I/O operations,
- Device independence,
- Avoidance of errors in operation of devices,
- Sharing of devices,
- Transparency and virtual devices.

The input output operations supported in the Theta operating system are open, read, write, close, seek and ioctl.

3.4.6 File management server

The file management server of Theta deals with two kinds of jobs. One is the virtual file system using a common

memory. The second is the file server which enables the UNIX machine to be a file server of Theta. Path names of files defined in the Theta operating system are shown in Table 1.

path name	device name
d:tty0	console 0
d:tty1	console 1
d:tty2	console 2
d:aux	serial I/O
u:filename	UNIX file server
r:filename	common memory

Table 1. Path names.

3.4.7 Scheduling server

The scheduler of the Theta operating system is realized by a thread which manages the ready queue. Kernel issues alert to invoke the scheduler to make it obtain the first thread in this queue. Return value of the scheduling server is the next thread to activate.

3.5 Exception handling

Two kinds of exception handling primitives are prepared. One is UNIX-like signal and the other is alert. Alert is used to make an attention to a thread which is ready to accept it. While a signal is used to interrupt any thread whether the thread likes it or not. Alert is used in events handling, cancellation of processing, and exception handling. Signal is used to monitor and debug a process.

3.5.1 Alert

The thread which issues alert_wait unlocks the mutex and blocks until other thread alerts it. If it is awakened by condition_signal() or condition_broadcast(), it locks the mutex and returns 0. If it is awakened by alert() or alert_broadcast(), it locks the mutex and returns type number of alert.

3.5.2 Signal

Signal is a facility for monitoring and debugging for parallel and distributed programming environment. It is implemented as a special alert. A thread accepts a signal only when it unlocks the mutex or when it changes its state from executing to ready after executing in a time quantum. This is treated when the dispatcher starts in the kernel.

Signal affects the thread in two ways. One is to kill the thread and related threads immediately after signal is sent. The other is to suspend the thread. The thread which issues the signal checks the return value of signal.

Using alert and signal primitives flexible control flow of concurrent programming may be described.

3.5.3 Library functions of exception handling

The five exception handling primitives are provided in Theta.

alert,
test_alert,
alert_wait,
test_alert,
alert_broadcast,
signal.

4. Application

In this section we describe the application of Theta operating system.[3]

4.1 An intelligent process monitor

In several plants, the process monitoring depends on human skill of well trained operators because of its complexity. For example, a furnace process, a mixing process and a fermentation process are still manipulated by human operators. The goal of intelligent monitoring is to detect the process and decide control operation by obtaining the required knowledge from a human expert. Human intelligence is based on subjective and experimental understanding about the process. Generally speaking, it is difficult to describe the whole mechanism of such a complex process as a mathematical model. Here, we have introduced a concept of "Intelligent Process Monitor". It is defined as an integration of the process monitor, the expert system and the man-machine interface.

The process monitor is a systematized equipment which displays the process feature, detects the status and stores data for standardization. The Dough monitor is a product as a typical process monitor. The expert system utilizes distributed knowledge base model and has the proper structure for rapid inference for process monitoring. The man-machine interface provides the interactive operation to store knowledge through the system utilization. In order to realize the intelligent process monitor, we have to develop a system, which supports concurrent execution of multiple tasks. We developed so-called "The Dough Manager" , an integrated process monitor for the dough mixing in a bakery plant, utilizing the basic facilities of the Theta.[4]

4.2 The outline of Dough Monitor

The Dough Monitor is now utilized in major bakery industries in Japan to standardize and manage the dough mixing process. The Dough Monitor consists of micro processor, IC memories, sensor base inputs and a graphic display unit. The Monitor is connected to the dough mixer driving motor through an electric power sensor, and the significant feature of this monitor is that it is realized by ROM. This monitor presents process status transition in a trend curve combined with an amplitude pattern. The monitor detects the process status by making a comparison with a standard trend and amplitude pattern previously defined in the monitor. Furthermore, the monitor shows the end point of mixing and stops the mixer by a pre-defined power consumption criterion. These criteria for the mixing are easily defined by an operator whenever he makes certain proper mixing.

4.3 The distributed expert shell on Theta

We utilized a distributed expert shell, ESPARON, which incorporates distributed problem solving agents. Each agent model consists of three parts, data part, rule part and metarule part. ESPARON has been applied in many problem solving system developments. ESPARON is usually realized on MS/DOS personal computer or UNIX work station. We also developed ESPARON/Theta, expert shell executable on **Icon.** It is compatible to other ESPARON, so we can develop the knowledge base model on a personal computer or a workstation for ESPARON/Theta.

4.4 Dough Manager implementation on Theta

Figure 3 describes an implementation of the Dough

[3] This section is written in cooperation with S. Inabayashi, System Sougou Kaihatsu Co. Ltd. of Japan.

[4] Dough Manager utilized an early version of Theta which is a message passing based operating system.

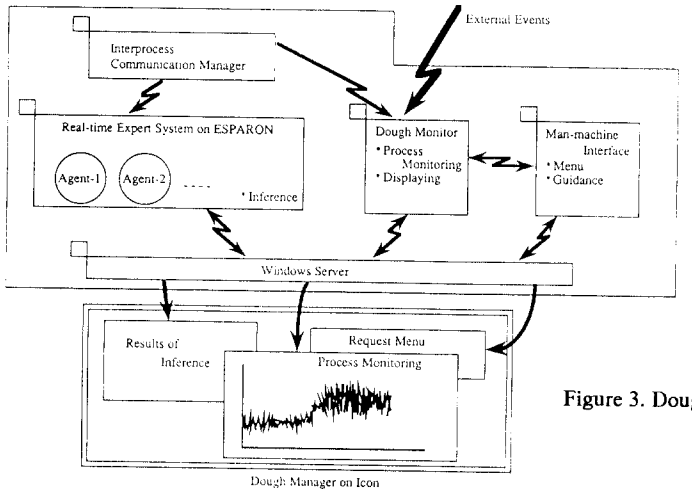

Figure 3. Dough Manager on **Icon**.

Manager on **Icon**. The features of the Dought Manager are described as follows.

- The Theta operating system has been ported to the VMEbus based CPU board so that we may develop various combinations of functions depending on the purpose of the system.
- An A/D converter board and a CRT control board are connected with the CPU board through the VMEbus.
- On the Theta operating system, three concurrent tasks are defined. These are the dough monitor task, the expert system task, and the man-machine interface task.
- In the expert system task, there are four agent models of ESPARON. These are the "Watch-man", the "Measure-man", the "Dough-master", and the "Reporter".
- Through the multi-window display, user may monitor the process status and instruct order to the system using a mouse or a keyboard operation.

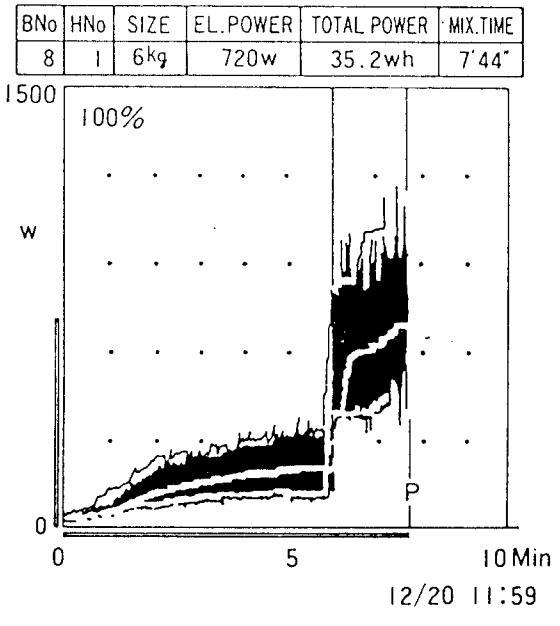

Figure 4. Graphic image for process monitoring.

4.5 Behavior of the system and its evaluation.

When the mixer is started, the dough monitor task makes the graphic image for process monitoring as shown in Figure 4. The dough monitor task detects and sends data to the expert system task. The expert system task diagnoses the process status. The Watch-man agent monitors the mixing process data and decides whether abnormal status happens or not, and sends a message to the other agent to get opinion from expert knowledge. The Measure-man agent detects abnormal measuring precisely. And, the Dough-master agent makes diagnostic inference and presents guidance to recover from an abnormal status. The Reporter agent makes some presentation to explain opinions from other agents. The man-machine interface task drives the CRT controller through VMEbus, and manipulates the multi windows display for the purpose to communicate with a human operator interactively.

This example shows that we can implement such an integrated multiple tasks application system successfully on single CPU board supported by Theta. The dough monitor task scans the mixing process data every 100 milliseconds. The expert system task makes inferences every 3 seconds to make advice and control the process. The operator can communicate with the system through the man-machine interface task whenever he wants. The system was demonstrated at an exhibition, POWDERTEC JAPAN 90 in September 1990.

5. Conclusion

The design objectives of **Icon** were twofold. First, we proposed a design method to make an embedded intelligent controller without degrading the performance compared with a commercial workstation. Second, we implemented a small but open operating system which supports Common Lisp, Prolog and real-time processing.

Experiments on the performance measurements measured on **Icon** and the commercial workstations show that **Icon** is comparable to the off-the-shelf workstations.

We chose a commercial VMEbus based computer as Icon. However, a large amount of RAM and EPROM of it are effectively used to increase the performance of the machine.

Conventional operating systems do not support such a machine. In this case.Theta provides a major contribution.

References

[1] T. Yuasa and M. Hagiya, "Kyoto Common Lisp Report," Teikoku Insatsu Inc. , 1985.

[2] W. F. Clocksin and C. S. Mellish, "Programming in Prolog," Second edition, Springer-Verlag, 1984.

[3] T. Matsui, "EZ Manual," Electrotechnical Laboratory, Research Memorandum, ETL-RM-21J, 1985. (in Japanese)

[4] M. Accetta, et al., "Mach: A New Kernel Foundation for UNIX Development," Proceedings of the Summer 1986 USENIX Conference, pp. 93-112, 1986.

[5] H. Shirakawa, H. Ogawa and M. Fujiwara, "A Dedicated Small Computer for Artificial Intelligence," Proceedings of the 1990 ACM SIGSMALL/PC Symposium on Small Systems, pp.191-198, 1990.

[6] "FORCE SYS68k/CPU-30 User's Manual," FORCE COMPUTERS Inc./Gmbh, 1990.

[7] A. Birrell, "An Introduction to Programming with Threads," Research Report 35, Digital Equipment Corporation Systems Research Center, 1989.

[8] A. Birrell, J. Guttag, J. Horning and R. Levin, "Synchronization Primitives for Multiprocessor: Formal Specification," Proceedings of the 11th Symposium on Operating Systems Principles, pp.94-102, 1987.

[9] E. C. Cooper, "C Threads," Technical Report CMU-CS-88-154, Computer Science Department, Carnegie Mellon University, 1988.

[10] J. E. Stoy and C. Strachey, "OS6 - an Experimental Operating System for a Small Computer," Computer Journal, Vol. 15, No. 2 and 3, May and August, 1972.

[11] D. M. Ritchie, "A Stream Input Output System," AT&T Bell Laboratories Technical Journal, Vol. 63, No. 3, Part 2, pp. 1897-1910, Oct., 1984.