Advances in the ITRON specifications - supporting multiprocessor and distributed systems

Hiroaki Takada Ken Sakamura

Department of Information Science,
Faculty of Science, University of Tokyo.
7-3-1, Hongo, Bunkyo-ku, Tokyo 113, Japan

Abstract

The design policies and overviews of the extended ITRON specifications supporting distributed systems and multiprocessor systems under investigations are described in this paper. Wide applicability and high run-time performance are primary goals of these extended specifications, which are realized by inheriting the design policy of the original specification that excessive virtualization of hardware should be avoided. In this paper, we review the design policies of the ITRON specifications, and present how the policies are incorporated in the extended specifications. The extensions expand the application areas of ITRON and make important steps towards the realization of HFDS, which is the final goal of the TRON Project.

1 Introduction

The ultimate goal of the TRON Project is to realize HFDS (Highly Functional Distributed System)[1]. In HFDS environment, all the equipment, furnishings, and other objects around us will come to have computer chips embedded in them. These objects, moreover, will be capable of interacting with each other via world-wide computer networks, enabling them to coordinate their actions, in order to provide better environment to human living. These computer-embedded, networked objects are called intelligent objects.

Since various kinds of intelligent objects cooperate with each other in HFDS environment, it is advantageous that each of them is implemented using a same kind of operating system. Improving the productivity of software is also among the important issues, because many intelligent objects must be developed.

Since 1984, we have designed, implemented, and evaluated the ITRON specifications, which are a series of standard kernel specifications for embedded computer systems, to meet these requirements [2].

To facilitate the realization of HFDS, the specifications which are necessary to develop intelligent objects should be freely usable for anyone. Therefore, it is one of the most important policies of the TRON Project that the specifications are opened to the public. Following this policy, we have designed and published the specifications of ITRON1, ITRON2, μITRON, and ITRON/FILE. Operating systems based on these specifications have been developed on various kinds of processors, and are utilized in many products.

The ITRON specifications should be applicable to any kind of embedded systems in HFDS environment. To achieve broader applicability, we continue to investigate on various extensions of the ITRON specifications. In this paper, the extensions supporting distributed systems and multiprocessor systems are described. Supporting distributed systems connected via computer networks is very important by the nature of HFDS.

2 Design policy of the ITRON specifications

The common design policies of the ITRON specification series are presented in this section. The policies characterize the ITRON specifications and should be kept in mind in investigating the extended specifications.

Basic requirements in designing the ITRON specifications are as the followings.

1. Maximum performance of hardware can be obtained using the specifications.

2. Real-time properties are kept using the specifications. Here, real-time properties mean fast response time and predictable execution time.

3. The specifications are applicable to various hardware.
4. The specifications serve for improving software productivity.

Improving software productivity often contradicts with high run-time performance. For example, one method to obtain high run-time performance is to allow implementors to select optimal kernel specification for each hardware by leaving some parts of the specification as implementation dependent. However, this approach impedes the portability of application programs, and degrades software productivity consequently. Operating systems designed in the TRON Project are characterized with the balancing point of this trade-off. Among them, ITRON gives most importance to run-time performance. Some critical parts for run-time performance are left undefined in the specifications as implementation dependent.

We adopt the following approaches to meet the requirements listed above [3, 4].

1. Excessive virtualization of hardware should be avoided in order to obtain maximum performance of the hardware and to keep real-time properties.

A virtual machine approach is one of approaches to design a standard operating system specification, in which a virtual architecture model of underlying hardware is assumed and the kernel specification is defined based on it. Though this approach has an advantage in software portability, the gap between the architecture model and the actual hardware architecture increase the kernel overhead, leading to low system performance. It also makes difficult for the programmers to grasp the real-time nature of the system.

In the ITRON specifications, which parts of the specification have large influence on run-time performance are investigated, and these parts are left as implementation dependent. Maximum run-time performance of the system can be obtained through this approach.

2. The specification should have the adaptability to an application.

The adaptability to an application is the ability that a kernel based on the specification can be tuned according to the functions and performance requested by the application for the improvement of the total system performance. In case of an embedded system, object code of its kernel is generated for each application, the adaptability to an application is effective for improving run-time performance and saving memory space.

3. The specification should have the adaptability to a hardware.

The adaptability to a hardware is the ability that a kernel based on the specification can be tuned according to the performance and characteristics of the hardware architecture for the improvement of the total system performance.

4. The training of programmers should be easy.

The easiness of the training is paid attention in designing the specifications. Terminologies in the specifications and naming of system calls are determined to be consistent. Wide applicability of the specifications also serves for easy training.

5. Broad applicability is realized through providing a series of specifications and introducing implementation levels in the specifications.

In order to make the ITRON specifications applicable to various kinds of hardware platforms, we provide a series of specifications for different kinds of hardware. For example, the main targets of the μITRON specification are small embedded systems using low-cost microcontrollers or 8-bit processors, and those of the ITRON2 specification are large systems using 32-bit processors. Implementation levels, which indicate the importance of each function, are also introduced in these specifications.

6. Abundant functionality should be provided.

Abundant kernel primitives with different functionalities and purposes are provided, instead of supporting limited number of primitives. By using most appropriate one for each hardware or application, improvement of run-time performance or software productivity can be expected, respectively.

3 Extensions of the ITRON specification

3.1 Multiprocessor systems and distributed systems

In extending the original ITRON specification for the systems using multiple processors, we first classify the hardware architectures of the systems, and design a series of specifications for each class of hardware. Because run-time performance is a primary concern in the ITRON specifications, the classification must reflect the run-time mechanism of each kernel primitive
on each class of hardware. Therefore, the implementation methods of kernels on various types of hardware architectures must be studied beforehand.

As the result of the investigations, the implementation methods of real-time kernels on multiple processor systems heavily depend on the existence of shared memories [5]. For example, suppose the case that a task $t_1$ executed on a processor $p_1$ wants to refer the status of a task $t_2$ on another processor $p_2$. If the TCB of $t_2$ is placed on a shared memory, its status can be read with little interference in its execution speed. Otherwise, $t_1$ needs to request $p_2$ to read necessary information from its private memory, and the execution of $t_2$ is delayed. When $p_2$ is busy or when inter-processor communication takes a long time, the waiting time of $t_1$ becomes long and cannot be ignored.

Another difference of shared-memory architecture is that migrating a task is easy if all resources used by the task are located on shared spaces. But, it is very difficult to realize it on a system without shared memories with high run-time performance.

On the other hand, some systems require that processors are physically dispersed, in which shared memories among processors are difficult to implement in hardware level.

From these considerations, we classify the hardware architecture with multiple processors to the shared-memory architecture and the distributed architecture in which processors are connected with computer networks such as LAN and have no shared memories among them. The original ITRON specification is extended for each class of architecture. In the following, we call the former class of architecture as multiprocessor systems, and the latter as distributed systems.

3.2 Object location and ID

In extending the ITRON specifications, continuity of the specifications is also an important issue. Programmers accustomed to the original specifications should easily learn the new ones, and existing programs should be easily portable.

When the ITRON specification is applied to a system with multiple processors, how to handle object locations is a major concern. For example, a task has a new attribute indicating on which processors it is executed.

In order to facilitate the migration to the new specifications, the location information is coded in the object ID number. Since each system call operating on an object (including the system calls creating an object) receives its ID as an argument in the original ITRON specification, system call interface can remain unchanged. This approach minimizes modifications from the original specification and facilitates software reuse.

4 Supporting distributed systems

As the extended specification supporting distributed systems is a basic one for building HFDS, the final goal of the TRON Project, we have decided to support distributed systems in a new version of the $\mu$ITRON specification. This new specification is named $\mu$ITRON version 3 [6].

In extending ITRON for distributed systems, two extension steps are taken. The first step supports fixed configuration systems, whose configuration are given beforehand and are not changed after the systems are started. This kind of system is usual in constructing a large embedded system using some small computers connected with LAN. Complex robotics systems and large plant-control systems are typical examples. This step is realized in $\mu$ITRON ver. 3 (Figure 1).

In contrast, the configuration of a system is dynamically changed in HFDS environment. Networks in HFDS environment are used for various purposes and are dynamically re-configured. We will support...
this kind of systems in the second step\(^1\).

Overview of the \(\mu\)ITRON ver. 3 specification is presented below.

### 4.1 Node

In the \(\mu\)ITRON ver. 3 specification, a system unit including a processor\(^7\) and connected to a network is called a *node*. A node is a self-contained system including a processor, memories, I/O devices, and a network controller. Message passing mechanism between nodes is assumed. The specification does not prescribe the physical media and the connection topologies of the network.

Each node has an ITRON kernel, and then has tasks and inter-task synchronization/communication objects on it. Objects in a node can be operated on by other nodes, and a node can operate on objects in other nodes. A node that uses another operating system than ITRON and that can only operate on objects in other ITRON nodes can be included in a system. Operating functions on objects in other nodes are called *connecting functions* (Figure 2).

In case of a distributed system using ITRON kernels, each node uses a processor suited to the role of the node, and constitutes what is called a heterogeneous distributed system. As a small node may use a processor with smaller word length than that of the processor used in a large node, difference of word length between nodes is considered in the specification.

---

\(^{1}\)We call the specification of this step as \(\mu\)ITRON.

\(^{7}\)Multiple processors can be included in a node, when used with the ITRON-MP specification presented in Section 5.

---

#### 4.2 Object ID

In designing a distributed system, it is advantageous that each node can be independently designed except for the interface with other nodes. To achieve independence between nodes, the objects that are used within a single node should be classified from the objects which can be operated on from other nodes. We call the former ones as private objects and the latter ones as shared objects.

In the \(\mu\)ITRON ver. 3 specification, shared objects and private objects are assigned ID numbers included in different ranges. In the concrete, the lower and the upper bounds of the shared object ID are determined for each node, and an ID number included in the range represents a shared object. The mapping of an ID number in the range to a shared object is unique in a system, and is fixed when the system is configured (Figure 3).

This ID scheme is also profitable for small nodes using 8-bit processors included in a large-scaled distributed system. This is because 8-bit word may be too short to specify all the objects in a large-scaled system, but be sufficient for identifying only private objects in the node and necessary shared objects in the system.
4.3 Added functions

Some new functions to help constructing a distributed system, as well as the connecting functions to operate on objects in other nodes, are introduced to the μITRON ver. 3 specification. For example, a system call to get the version information of another node and a function to copy data between nodes are added.

New error codes are also introduced to represent error status specific to a distributed system, such as calling an unsupported function of another node and communication error between nodes.

4.4 Unsupported functions

In the μITRON ver. 3 specification, kernel primitives depending on the existence of shared memories cannot be used beyond networks. Shared memory pools and mailboxes are examples of these primitives. It is possible to implement shared memory pool function in a distributed system using the technique of distributed shared memories. However, it imposes some restrictions on the underlying hardware, and its run-time performance is not so good even when it is possible. It is sufficient if equal functionality can be realized using other primitives, such as message buffers. A task that is executable on multiple nodes is not introduced in the specification for the same reason.

Some system calls that control raw hardware resources on the node cannot be used beyond networks, either. Interrupt management and timer management calls are their example. When it is necessary to call this kind of system call on another node, a server task is used on the target node and the requesting task requests the server using some communication primitive such as the rendezvous.

4.5 Other features

The μITRON ver. 3 specification also presents the standard inter-node communication protocol in order that a distributed system can be constructed using kernels from multiple vendors. In the concrete, a standard packet format for inter-node communication is defined. How to pack the defined packet into a frame of a lower layer protocol must be defined separately for each lower layer.

On the other hand, standardization of the inter-node communication protocol in incompatible with the adaptation of the protocol, that is to use optimal protocol for each network and application in order to archive high run-time performance. Accordingly, an adapted protocol can be used when connecting functions with other kernels are not necessary. It is also possible to provide both the standard protocol and an adapted one and to choose one according as the protocols supported in the target node.

5 Supporting multiprocessor systems

The main purposes of using shared-memory multiprocessor for an embedded system is to obtain high computational power and to archive fault-tolerance. The extended specification supporting shared-memory multiprocessor systems is call ITRON-MP [7].

The hardware architectures of embedded systems and special-purpose machines, main targets of the ITRON-MP specification, are designed to be optimal for each application in order to archive high cost-performance. Because multiprocessor architectures have wide diversities, such as the kind and the number of processors, processor connection topology, and the accessibility and the access cost of hardware resources from each processor, a kernel specification which are designed presuming a specific architecture cannot be a standard.

The ITRON-MP specification aims to be applicable to wide varieties of multiprocessor architectures. The following approach is adopted to meet this goal. The specification defines a sufficient set of kernel functions, from which suitable functions to each hardware architecture are chosen and implemented. In other word, the ITRON-MP specification defines the extent of the specification of ITRON-MP conforming kernels.

Moreover, in order to reduce the system development cost, an adaptive kernel implementation, which can generate a tuned kernel code from the descriptions of the architecture and the kernel, should be possible.

Overview of the ITRON-MP specification is presented below.

5.1 Object class

In a shared-memory multiprocessor system, hardware resources, such as memories and I/O devices, are classified according as their accessibility and their access efficiency from each processor. Exceptions are symmetric architectures, or uniform memory access time (UMA) architectures, which are usually inappropriate for embedded systems and special-purpose machines, and are not the main targets of the ITRON-MP specification.

Following the approach that excessive virtualization of hardware should be avoided, the ITRON-MP specification reflects the asymmetry of an architecture.
to its kernel interface. Programmers can be conscious of the raw execution mechanism, such as physical locations of kernel objects. In the concrete, kernel objects are classified according as the hardware resources on which they are implemented. Namely, objects are classified according as their accessibility and their access efficiency from each processor. A class of objects are called an object class.

Tasks, the most important kernel objects, are further classified by their executability by each processor as well as their accessibility. Figure 4 illustrates a simple example of task classes. A task belonging to the unbounded task class uses global memories and can be executed by both processor 1 and 2, and a task belonging to the \(p_1\)-bounded task class uses local memories of processor 1 and is executable only by the processor. Accessibility of the \(p_1\)-bounded task from another processor depends on the accessibility of the local memory in which its TCB resides. Programmers can know whether a task can access an object or not and its access efficiency (when it is accessible) from the class of the task and that of the accessed object.

5.2 Object ID

An object ID consists of the field indicating to which class the object belongs and the field identifying the object in the class. Consequently, programmers can easily know to which class an object belongs from its ID. This ID scheme is adopted because multiprocessor systems are usually constructed using powerful processors, and because the number of the object classes are limited (16-bit length class ID is sufficient for almost all systems).

5.3 Added functions

Inter-task synchronization mechanisms peculiar to multiprocessor systems and functions supporting fault-tolerant computing are introduced to the ITRON-MP specification.

Barrier synchronization and synchronization by spin-locking are important inter-task synchronization mechanisms in multiprocessor systems. Barrier synchronization is effective when multiple processors cooperate to process a single job. Though simple barrier synchronization can be realized using an eventflag with “and” condition in the original ITRON specification, it is not sufficient when the number of synchronizing tasks are large. It is not efficient, also. Therefore, barrier synchronization objects are introduced in addition to the eventflags.

Synchronization by spin-locking is the mechanism that occupies a processor all the while a task is in wait state instead of dispatching to another task. Because no task dispatching occurs by the synchronization, it can be provided as library calls. However, since it is convenient for programmers that all inter-task synchronization primitives should be equally available, synchronization mechanisms by spin-locking are taken into the ITRON-MP specification.

To support fault-tolerant systems, the ITRON-MP specification provides functions for getting a snapshot of a task and for recovering the task from the snapshot. Contents of memory area used by a task can be read while the task is suspended, and no kernel support is necessary. As the task status is managed by the kernel, some kernel supports are necessary to read and restore it. In the concrete, system calls to read and restore the TCB status are provided.

We also investigate on the introduction of waiting handlers, which have only the waiting facilities of tasks. Waiting handlers are useful for realizing fault-tolerant systems as well as implementing a kernel supporting distributed systems.

5.4 Unsupported functions

We distinguish shared-memory multiprocessor systems because objects managed on a shared memory can be accessed from a processor with little interference on other processors. Therefore, functions that need to be processed by requesting another processor should not be supported in the ITRON-MP specification. When such an operation is necessary, a server task is executed on the requested processor and the

---

5 In order to implement a kernel object, memory area for its control block is necessary at least.

4 It is possible to realize them as library functions.
requesting processor requests the server using the rendezvous function. Such an operation can be realized by preparing a server task on the requested processor.

5.5 Other features

The ITRON-MP specification adopts the priority-based task scheduling inherited from the original ITRON specification. However, in case of multiprocessor systems, there are some variations of "priority-based" scheduling policies, and none of them is universal. In ITRON-MP, a definite task scheduling policy is not presented, but a scheduling principle that prescribes basic rules of scheduling policies is defined. Some system calls that affect the task scheduling are also defined, such as binding a task which can be executed on some processors temporarily to a processor.

6 Conclusion

We preset the design policies and overviews of the extended ITRON specifications for distributed systems and for multiprocessor systems, which are under investigations. These extended specifications inherit the design policies of the original ITRON specification, such as achieving high run-time performance and keeping real-time properties through avoiding excessive virtualization of hardware. They extend the application area of the ITRON specifications, and are important steps towards the realization of HFDS, an ultimate goal of the TRON Project.

Further advances in the ITRON specifications will be directed to the IMTRON specification supporting dynamically re-configured systems and networks, which are usual in HFDS environment (Figure 5). We also began basic researches on hard real-time systems, in which missing a timing constraint can cause catastrophic results. It is also an important theme to standardize software development environment, which eases the training of programmers.

Acknowledgments

We wish to thank Dr. Kazunori Saitoh, Dr. Kiichiro Tamaru, and other members of the ITRON technical committee for their continuous support.

References