The ITRON Project: Overview and Recent Results

Hiroaki Takada
Dept. of Information and Computer Sciences
Toyohashi Univ. of Technology
1–1 Hibarigaoka, Tempaku-cho
Toyohashi 441–8580, Japan

Yukikazu Nakamoto
Software Design Lab.
NEC Corporation
2–11–5 Shibaura, Minato-ku
Tokyo 108-0023, Japan

Kiichiro Tamaru
System ULSI Engineering Lab.
TOSHIBA Corporation
580–1 Horikawa-cho, Saiwai-ku
Kawasaki 210–8520, Japan

Abstract

The ITRON Project is to standardize real-time operating system and related specifications for embedded systems. We have defined and published the series of ITRON real-time kernel specifications. Of these, the μITRON real-time kernel specification, which was designed for consumer electronic applications and other small-scale embedded systems, has been implemented for many kind of processors and adopted in numerous end products, making it an industry standard in this field. Based on the achievements, we have broadened the scope of our standardization efforts beyond the kernel specifications to related aspects.

This paper briefly outlines the history and the current status of the ITRON project and introduces its recent results, including the ITRON TCP/IP API specification, an application program interface (API) specification for TCP/IP protocol stacks suitable for embedded systems, and the JTRON2.0 specification, an interface definition for hybrid environments of Java Runtime Environment and ITRON-specification kernel. We also describe the current standardization activities in the project, focusing on μITRON4.0, the next generation μITRON real-time kernel specification.

1 Introduction

Advances in microprocessor technology continue to open up new application fields for embedded systems, and today nearly almost all the electrical and electronic products around us are controlled by embedded systems. At the same time, the equipment controlled by embedded systems has become more sophisticated, often incorporating many functions in one product. As a result, embedded system software has grown in scale and complexity. Moreover, as products increasingly adopt digital technology, advanced microprocessors have enabled more of the processing to be implemented in software, making embedded system software more important.

Real-time operating system is among the key components in designing embedded systems. Currently, the real-time operating system market is so fragmented that we can say that there exists no international standard in this field. The main reason is that embedded software tended to be developed with a company's propriety software technology. With the increasing complexity of embedded system software, however, introducing some software components from outside software vendors becomes unavoidable, thus the standardization of real-time operating system becomes very important.

The ITRON Project, which is one of the subprojects of the TRON Project [1, 2], is to standardize real-time operating system and related specifications for embedded systems. The series of ITRON real-time kernel specifications have been defined and published so far [3]. Real-time kernels conformant to the specifications have been implemented for many kind of processors and adopted in numerous embedded system designs. Now, it can safely be said that the ITRON specifications are an industry standard in this field in Japan. In this first stage of the ITRON Project, we focused our standardization efforts on the real-time kernel specifications. The reason is that many embedded systems (especially, small-scale ones) use only kernel functions.

As embedded systems grow larger and more complex, however, the relative importance of real-time kernels is decreasing. The need has increased for standardization efforts that take into account software components, development tools, and other specifications related to embedded system software and hardware. Among them, we have determined to put an emphasis on software component-related standardization at first, and started several standardization activities described in the following sections. With these activities, the second stage of the ITRON Project has started [4].

In order to promote the circulation and use of software components, two directions of standardization efforts are

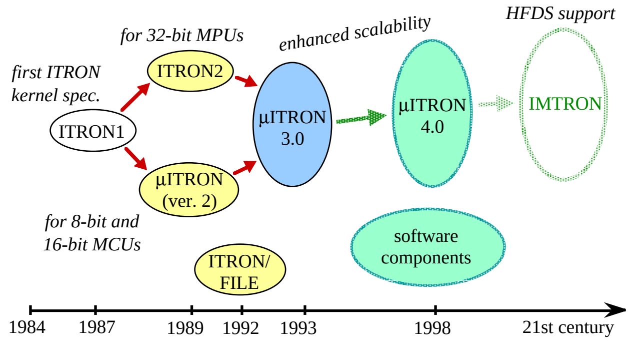

Figure 1: History of the ITRON specifications

necessary.

At first, it is necessary to satisfy the preconditions for their circulation and use. In more specific, the software components should be portable to different platforms. In addition, because some software components have hard timing constraints, the framework to guarantee the timing constraints of each component within a system is necessary. In other words, what is needed is a framework that allows coexistence of those software components with applications while satisfying their real-time constraints, and enabling use of multiple software components each with their own real-time needs.

The second direction is to standardize the application program interface (API) of software components, which must be done for each kind of software components.

This paper first presents the overview of the current ITRON real-time kernel specifications, and then introduces two recently published specifications of software components. We also describe the current standardization activities in the project, focusing on μITRON4.0, the next generation μITRON real-time kernel specification.

2 ITRON Real-Time Kernel Specifications

The overview of the current ITRON real-time kernel specifications, the results of the first stage of the ITRON Project, is presented in this section.

2.1 History

Since the ITRON Project started in 1984, we have studied standard real-time OS specifications for embedded systems, and have developed and made available the series of ITRON real-time kernel specifications as a result (Figure 1).

The first ITRON real-time kernel specification was issued in 1987 as the ITRON1 specification [5]. Several kernel products were developed based on this specification and applied to systems, mainly by way of proving the applicability of the specification. In 1989, two revised specifications were released. The μITRON specification (Ver. 2.0) was developed with a smaller set of functions targeted to 8-bit and 16-bit MCUs, and the ITRON2 specification for 32-bit processors. Of these, the μITRON specification met the industry's demands very well, and has been adopted in a number of embedded system designs.

Thereafter, the expanding use of MCUs resulted in the μITRON specification being implemented for 32-bit processors, which was not anticipated when the specification was designed. We therefore decided to revise the specification approach by drawing up a scalable specification, able to be used with MCUs ranging from 8-bit to 32-bit processors. The result of this work was issued in 1993 as the μITRON3.0 specification.

2.2 Design Principles

The ITRON specifications are designed so that the following requirements on an operating system standard for embedded systems are satisfied [3].

- Being able to derive maximum performance from hardware
  Given the severe hardware resource limitations of a typical MCU-based system, the ability to derive maximum performance from the available hardware is a prerequisite for real-time OS adoption.
- Helping to improve software productivity
  Especially important is standardization from a training standpoint, such as adopting consistent concepts and terminology, and standardizing design methods.
- Being uniformly applicable to various processor scales and types
  The hardware used in an embedded system is normally designed optimally for its application. The processor scale, moreover, may vary widely from 8-bit to 32-bit processors depending on the kind of equipment to be controlled.

In addition to the above requirements, another very important issue is whether the specifications are truly open. This means not only that the specification documents can be obtained, but also that everyone is free to implement and sell products based on those specifications.

The ITRON real-time kernel specifications were designed according to the following principles.

- Allow for adaptation to hardware, avoiding excessive hardware virtualization
  In order for a real-time kernel to take maximum advantage of the performance built into the MCU and deliver excellent real-time response, the specifications must avoid excessive virtualization of hardware features. Adaptation to hardware means changing the real-time

kernel functions and internal implementation methods as necessary based on the hardware architecture and performance, raising the overall system performance.

- Allow for adaptation to the application

  Adaptation to the application means changing the kernel functions and internal implementation methods based on the functions and performance required by the application, in order to raise the overall system performance. In the case of an embedded system, the kernel object code is generated separately for each application, so adaptation to the application works especially well.

- Emphasize software engineer training ease

  The ITRON specifications employ standardization as a way of making it easier for software developers to acquire the necessary skills. Consistency in use of terminology, system call naming and the like help ensure that once something is learned, it will have wide applicability thereafter. Another way training is emphasized is by making available educational text materials.

- Specification series organization and/or division into levels

  To enable adaptation to a wide diversity of hardware, the specifications are organized into a series and/or divided into levels based on the degree of need for each function. When the kernel is implemented, the level can be chosen based on the kinds of applications aimed for and their required functions.

- Provide a wealth of functions

  The primitives that the kernel provides are not limited to a small number but cover a wide range of different functions. By making use of the primitives that match the type of application and hardware, system implementers should be able to achieve high runtime performance and write programs more easily.

A basic approach common to several of these design principles is “loose standardization.” This refers to the approach of leaving room for hardware-specific and application-specific features rather than trying to apply strict standards to the extent that runtime performance would be harmed. Loose standardization makes it possible to derive the maximum performance benefits from a diversity of hardware for a diversity of application.

2.3 Current Status

The ITRON real-time kernel specifications, esp. μITRON, have been implemented for many kind of processors and adopted in numerous end products, making it an industry standard in this field in Japan.

Audio/Visual Equipment, Home Appliance	TVs, VCRs, digital cameras, settop box, audio components, microwave ovens, rice cookers, air-conditioners, washing machines
Personal Information Appliance, Entertainment	PDAs (Personal Digital Assistants), personal organizers, car navigation systems, game gear, electronic musical instruments
PC Peripheral, Office Equipment	printers, scanners, disk drives, CD-ROM drives, copiers, FAX, word processors
Communication Equipment	answer phones, ISDN telephones, cellular phones, PCS terminals, ATM switches, broadcasting equipment, wireless systems, satellites
Transportation, Industrial Control, etc.	automobiles, plant control, industrial robots, elevators, vending machines, medical equipment

Table 1: ITRON-specification kernel applications

Currently, more than 40 real-time kernel products implemented for around 30 different processors are registered with the ITRON Technical Committee, and 10 to 20 non-registered products are known. Major Japanese semiconductor makers have implemented ITRON-specification real-time kernels on their own processors, and several software vendors have implemented for widely used processors. Support for the ITRON-specification kernel is starting to come from U.S. software vendors as well. Moreover, because the μITRON-specification kernel is small in size and relatively easy to develop, many companies have built kernels for their own in-house use in addition to the products mentioned above. There are also several μITRON implementations available as free software.

Obviously, with so many ITRON-specification kernels having been implemented, they are being used in many different application fields. Table 1 gives some examples of the huge number of applications making use of an ITRON-specification kernel. The survey conducted by the ITRON Technical Committee from late 1997 through early 1998 in Japan shows that the ITRON specifications are in especially wide use in consumer products fields, where they are a de-facto industry standard (Figure 2). Among the cases where an ITRON-specification kernel is used, many of these use an in-house implementation of the kernel, attesting to the true openness of this standard specification.

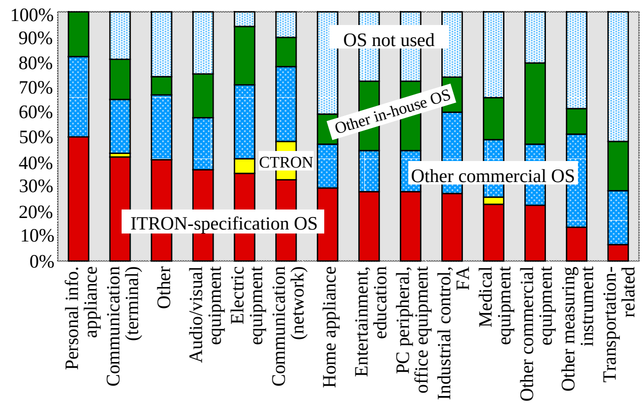

Figure 2: Real-time OS use in embedded systems
(TRON Association Survey, 1997-1998, in Japan)

3 ITRON TCP/IP API Specification

3.1 Motivations

A TCP/IP protocol stack is one of the most important software components for embedded systems, and its importance is ever increasing. We determined to start the efforts to standardize software component APIs with this important field. As a result, the first version of the ITRON TCP/IP API specification was published in May, 1998 [6].

The most widely used API for the TCP/IP protocol stack today is the socket interface originally designed for BSD UNIX operating system. The socket interface, however, has the problems listed below when it is applied to embedded systems.

- The socket interface is designed to be protocol-independent, thus has some overhead.
- A TCP/IP protocol stack with the socket interface must depend on dynamic memory management facility. When memory space is running short, for example, the protocol stack silently discards packets. This is not suitable for many embedded systems.
- The number of data copies becomes larger with the socket interface.
- The socket interface is designed to be compatible with the UNIX process model, which is quite different from the ITRON task model.

3.2 Overview

The ITRON TCP/IP API specification defines a standard API for the TCP protocol and the UDP protocol over IPv4 (version 4 of the Internet Protocol). The other APIs necessary for a TCP/IP protocol stack product (for example, APIs for managing the IP routing table and for managing the ARP table) are out of the scope in the current version.

Considering that many existing internet software are based on the socket interface and that many software engineers are familiar with it, the ITRON TCP/IP API is based on the socket interface, and the problems listed above are remedied. It is also possible to implement a library implementing the socket interface on top of the API.

Some of the important differences with the socket interface are listed below.

- The API for TCP and that for UDP are separately specified. In other words, the ITRON TCP/IP API is protocol-dependent, while the socket interface is protocol-independent. In addition, the current version is focused on IPv4.
- In stead of adopting the socket abstraction, end points for communications are directly handled. Moreover, the end point to wait for TCP connection requests (which corresponds to a socket on which `listen` is called) and the end point for a TCP connection are managed as different objects, while a socket is an abstraction of both objects. In the specification, the former one is named a TCP reception point (abbreviated as “rep”) and the latter one is named a TCP communication end point (abbreviated as “cep”).
- A set of TCP service calls with which the number of data copy can be reduced is defined in addition to the usual `read`/`write` style APIs. Specifically, `tcp_rcv_buf` returns the start address and the length of the buffer in which the received data is stored. After processing the data within the buffer, the application program should call `tcp_rel_buf` to release the buffer space. The service calls for sending data is defined similarly.
- Non-blocking calls and callbacks are supported for asynchronous handling of the protocols, which has a different semantics with that of the socket interface of UNIX. The infamous `select` call is not supported, because it can be emulated with the callbacks and the eventflag function of the ITRON real-time kernel specifications.
- When an UDP packet is received, a callback routine is called instead of storing the packet within a buffer managed by the protocol stack. Within the callback routine, the application program should allocate a buffer space for the UDP packet and copy the packet to the buffer using `udp_rcv_dat`. Otherwise, the UDP packet is discarded. With this approach, the application can know when memory space is running short.

Currently, several companies are developing TCP/IP protocol stacks based on the ITRON TCP/IP specification. After the implementations are completed, we plan to review the specification again and revise it.

4 JTRON2.0 Specification

4.1 Motivations

Java technology is drawing interest also in the field of embedded systems these days. A Java runtime environment is one in which programs downloaded from a network can be run safely, and Java lends itself readily to creating a graphical user interface (GUI). Despite obvious advantages like these, the overhead for byte code processing and the need for garbage collection make Java less than ideal for real-time systems.

A practical approach for applying Java technology to embedded real-time systems is to implement the Java Runtime Environment on top of an ITRON-specification kernel, then build an application system whereby the parts for which Java is best suited are implemented as Java programs, and the parts taking advantage of the ITRON-specification kernel strengths are implemented as ITRON tasks. With this approach, we can obtain their respective advantages.

A key issue here is the interface for communication between Java programs and ITRON tasks. This interface has to be standardized; otherwise Java's highly touted portability will be lost. The JTRON specification is to define this interface standard [7, 8].

4.2 Overview

Three types of interfaces are defined in the specification. The first is the interface by which a Java program accesses ITRON-specification kernel objects (Type 1). The second is for sharing Java objects between a Java program and ITRON tasks (Type 2), and the third is for stream communication between a Java program and ITRON tasks (Type 3) [9].

Type 1

With Type 1 interface, a Java program can access the kernel objects such as tasks and semaphores through the *attach classes*. In more specific, an attach class and its instance correspond to an object type and an object of the ITRON specification, respectively. A method of an attach class corresponds to a service call of the ITRON specification. For example, a semaphore attach class is defined corresponding to the semaphore object and has `signal` method corresponding to the `sig_sem` service call of the ITRON specification.

Type 2

With Type 2 interface, a Java program and ITRON tasks communicate through a *shared object*. A shared object is exported from the Java program by registering the object to the shared object manager. ITRON tasks can obtain the pointer to the shared object and directory access it with a structure definition generated by javah command.

In order to access a shared object exclusively, explicit locking/unlocking mechanism is provided to the shared objects. In addition, the Java program should call `unshare` method to cease the object sharing. The unsharing operation on a shared object is blocked when the shared object is locked, with which an ITRON task can access the shared object in safe while the task locks the object.

In the JTRON2.0 specification, the API for a Java program to share, unshare, lock, and unlock a shared object is defined, as well as the API for ITRON tasks to lock, unlock, and get the pointer to a shared object.

Type 3

Type 3 interface provides a simple stream-base message passing facility between a Java program and ITRON tasks. From the Java program, the stream for communicating with ITRON tasks can be handled just like the Java native stream. The API for a Java program to obtain the stream is defined in the JTRON2.0 specification.

5 Next Generation μITRON Real-Time Kernel Specification

The most important activity currently in progress is the standardization of μITRON4.0, the next generation μITRON real-time kernel specification.

Note that the specification described in the following sections is the current snapshot of the discussions. It may be changed until the μITRON4.0 specification is fixed.

5.1 Motivations

The two major motivations for designing new real-time kernel specification are to raise the portability of software components (and application software) developed on the μITRON-specification kernels and to incorporate new kernel functionalities, including the functions supporting hard real-time systems. In addition, we think that regular updates of the specification is required to follow the rapid advancement of microprocessor technology.

5.2 Standard Profile

Concept

The standard profile is a set of real-time kernel functions, defined for raising the portability of software components. The software components (or application software) which are required to be portable among different μITRON kernels are recommended to use only the functions included in the standard profile, and the real-time kernels to which the software components are requested to be portable are recommended to implement all the functions included in the standard profile.

Extensions and subsettings of the standard profile are still permitted in order to retain the advantages of the loose standardization policy of the ITRON specifications.

Function Overview

The standard profile of the μITRON4.0 specification includes almost all level S (standard level) functions of the μITRON3.0 specification. It also incorporates some extended functions of μITRON3.0 and some new functions including exception handling functions described below.

The functions supported in the standard profile of the μITRON4.0 specification are summarized in Table 2.

Task management
- Static API to create a task
- Services call for direct manipulation of a task

Task-dependent synchronization
- Service calls for task synchronizations

Task exception
- Static API to define a task exception routine
- Service calls for raising, disabling, and enabling task exception

Synchronization and communication
- Static APIs to create synchronization and communication objects
- Service calls for four task-independent synchronization and communication functions: semaphores, eventflags, mailboxes, and data queues

Interrupt management
- Static API to define an interrupt handler
- Service calls for disabling and enabling interrupts

Memory pool management
- Static API to create a memory pool
- Service calls for allocating and releasing fixed-size memory block from/to a memory pool

Time management
- Service calls for system clock setting and reference
- Service call for task delay
- Service call for provide time ticks
- Static API to create a cyclic hander
- Service calls for starting and stopping a cyclic handler

System status referencing
- Service calls for referencing the current system status

System management
- Static API to define a CPU exception handler
- Service call for referencing the kernel version

Table 2: Functions supported in the standard profile

Exception Handlings

Exception handling functions are totally defined for each implementation in the current μITRON specification. In μITRON4.0, two functions for exception handling, CPU exception handlers and task exception routines, are defined in the standard profile.

The CPU exception handlers are to handle CPU exceptions, such as zero-division or bus error. In the standard profile, though the API to define a CPU exception handler is defined, how to write a CPU exception handler is not standardized. This is because the CPU exception mechanisms of processors have great variety, thus it is difficult to standardize it with low overhead. At least, a CPU exception handler is required to be able to raise a task exception.

The task exception routines are to handle exceptional events in task contexts. Only one task exception routine can be defined for each task for making the kernel footprint small. An exceptional event are raised on a task with `ras_tex` (or `iras_tex` when invoked from an interrupt handler) service call. The kind of events is passed as a parameter to `ras_tex` and then passed to the task exception routine.

Static API

The service calls to create and delete kernel objects (`cre_tsk`, `cre_sem`, etc.), which are level E (extended level) functions in μITRON3.0, are not included in the standard profile of μITRON4.0. In most μITRON implementations without those service calls, kernel objects are created statically referring to the kernel configuration file. The syntax of the kernel configuration file is different for each implementation and is not portable.

In the μITRON4.0 specification, in order to ease the software porting, descriptions in kernel configuration files, called static API, are standardized. For example, the directive for creating a task is `CRE_TSK` (note that it is described in capital letters) and the parameters to it are basically the same with the `cre_tsk` service call. With this approach, application programmers are requested to study one API only.

5.3 Extended Functions for Hard Real-Time Systems

Two functions will be introduced to μITRON4.0 as the extended functions for hard real-time support: mutual exclusion mechanism with priority ceiling and priority inheritance support and overrun detection mechanism. The detailed specification of the functions is still under discussions.

5.4 Automotive Control Profile

With current practice, real-time kernels are difficult to apply to automotive control systems, mainly because automotive control applications generally require very short response with very limited hardware resources, and because the overhead of real-time kernels is not permissible. In the recently developed systems, however, the control systems grow larger and require more sophisticated run-time software.

We formed a committee with automotive engineers to bring together the requirements on real-time kernels used in automotive control systems and to propose a real-time kernel specification suitable for them.

One of the recommendations from the committee is (basically) a subset definition of μITRON including only necessary functions for many automotive control applications. In addition, a mechanism to share a stack space with multiple tasks is introduced. The subset definition will be incorporated to the μITRON4.0 specification, as another profile than the standard profile.

5.5 Real-Time Kernel without Wait State

The other recommendation from the committee is a real-time kernel specification without wait state. In the previous versions of the ITRON kernel specifications, wait state is mandatory. It is thought to be the prerequisite for a real-time kernel.

In recent studies, however, many application systems, especially small-scale systems, do not necessarily require wait state. Without wait state, all the tasks within a system can share one stack area, and thus removing wait state is very effective for decreasing memory consumption and reducing task dispatching overhead. Though it is still questionable if a real-time kernel without wait state can be called as a "real-time kernel," it is useful to define such real-time kernel specification as a subset of μITRON an introductory specification.

From these considerations, wait state is removed from the minimum requirements of μITRON4.0. As a result, the minimum implementation of μITRON4.0 is even smaller than that of the current μITRON specification.

6 Other Current Activities

6.1 Application Design Guidelines

Motivations

There are two motivations to define application design guidelines for real-time embedded systems. One of them is to provide a standard approach to design an embedded system using a real-time kernel for embedded application designers. For example, how to divide a system into tasks and how to assign priorities to them should be covered.

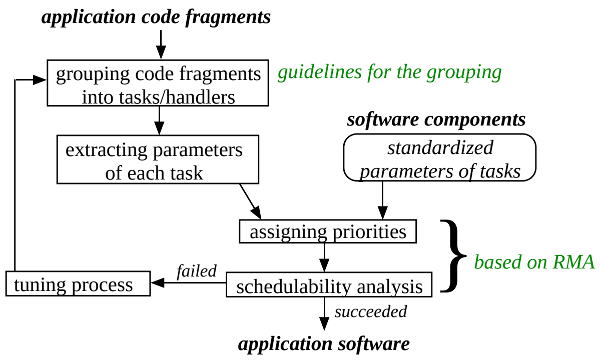
application code fragments
grouping code fragments into tasks/handlers
guidelines for the grouping
software components
extracting parameters of each task
standardized parameters of tasks
assigning priorities
based on RMA
failed
tuning process
schedulability analysis
succeeded
application software

Figure 3: Basic flow of application design guidelines

We think that it is impossible to cover all application fields of embedded systems with one set of guidelines, because of the great variety of embedded systems. The set of design guidelines being defined is an approach focusing on real-time features of embedded systems.

The other motivation is to support software components with hard real-time characteristics. In order to make software components with hard real-time characteristics coexist with applications while satisfying their real-time constraints, both of the software components and the applications should be designed following a set of rules, or design guidelines.

Overview

In order to guarantee the real-time constraints of application systems, the application design guidelines adopt the rate monotonic analysis (RMA) as the basic scheduling theory.

The guidelines are organized as follows (Figure 3). At first, the application system should be divided into processings which are basic computation units constituting the system and which correspond roughly with functions in C language or subroutines in assembler. The guidelines do not cover this step, but show how to group the processings into a task. To do that, parameters representing the real-time characteristics of the processing, including the deadline, the maximum execution time, the maximum execution frequency, and the significance should be listed up.

Then, the processings having the same (or similar) real-time constraints are built up into a task. The priority of a task is assigned according to the deadline monotonic scheduling policy. After those step, the schedulability of the system is checked using the RMA theory. If the system is found to be unschedulable, some kind of tuning process should be applied.

When a software component having real-time constraints is provided, the provided should present the real-time characteristics of the component. The user of the component can check the schedulability of the system consisting of the software component and their own application programs.

6.2 Device Driver Design Guidelines

Another current activity is to define the guidelines for designing device drivers. Because the ITRON real-time kernel specifications have no I/O functions, device drivers are implemented on a real-time kernel. Therefore, device drivers are directly used by application programs, not by an operating system kernel. In order to avoid confusion, we are using the term *device interface components* (DIC) instead of device drivers.

We are now defining a hierarchical model of DICs. The lowest level DIC has a special characteristics that it should not add functions to the hardware device and that it should not depend on real-time kernel functions. We think that the lowest level DIC should be provided by the device maker.

7 Summary and Future Plan

In this paper, we have presented the overview and the recent results of the ITRON Project. As the result of the activities in the second stage, several outcomes are to be obtained. The resulting specifications and the guidelines will be made open following the basic policy of the TRON Project.

Another important activity we are planning to start very soon is the interface standardization between real-time kernels and debugging tools such as software debuggers, in-circuit emulators (ICE), and logic analyzers. With a standard interface between them, making a debugging tool support μITRON-specification kernels becomes easier and we can expect that more software development tools will support μITRON-specification kernels.

Other topics include the standardization of C++ API of the ITRON specifications and the API standardization of software components supporting graphical user interface (GUI).

The market environments surrounding the ITRON Project are changing very rapidly. We will continue the efforts to catch up the the market requirements and to contribute for the advancement of embedded system technologies.

Acknowledgments

We would like to thank Dr. Ken Sakamura, the TRON Project Leader, and the members of the ITRON Technical Committee and other ITRON-related study groups and committees for their support and efforts for the ITRON Project.

References

[1] K. Sakamura, “The objectives of the TRON project,” in *TRON Project 1987*, pp. 3–16, Springer-Verlag, 1987.

[2] K. Sakamura, “After a decade of TRON, what comes next?,” in *Proc. 11th TRON Project Int’l Symposium*, pp. 2–16, IEEE CS Press, Dec. 1994.

[3] H. Takada and K. Sakamura, “μITRON for small-scale embedded systems,” *IEEE Micro*, vol. 15, pp. 46–54, Dec. 1995.

[4] H. Takada and K. Tamaru, “Recent results of the ITRON subproject,” in *Proc. 14th TRON Project Int’l Symposium*, pp. 31–35, TRON Association, Mar. 1998.

[5] H. Monden, “Introduction to ITRON, the industry-oriented operating system,” *IEEE Micro*, vol. 7, pp. 45–52, Apr. 1987.

[6] Embedded TCP/IP Technical Committee and ITRON Technical Committee, *ITRON TCP/IP API Specification (Ver. 1.00.01)*. May 1998. (only Japanese version is available now, “http://www.itron.gr.jp/SPEC/tcpip-e.html”).

[7] Java Technology on ITRON-specification OS Technical Committee, *JTRON2.0 Specification (Ver. 2.00.00)*. Oct. 1998. (only Japanese version is available now, “http://www.itron.gr.jp/SPEC/jtron2-e.html”).

[8] Y. Nakamoto and H. Takada, “Integration of Java and μITRON,” in *Proc. 14th TRON Project Int’l Symposium*, pp. 37–40, TRON Association, Mar. 1998.

[9] Y. Nakamoto and H. Takada, “JTRON: A hybrid architecture of Java runtime environment and real-time OS.” (in submitting).

The ITRON Technical Committee provides various information on the ITRON Project through the Internet, including the latest ITRON specifications and the ITRON Newsletter. The URL is “http://www.itron.gr.jp/”.