
Ken Sakamura
Interfaculty Initiative in Information Studies, Graduate School of the University of Tokyo
7-3-1 Hongo, Bunkyo-ku, Tokyo, JAPAN
+81-3-5841-2484
ken@sakamura-lab.org

ABSTRACT
Ubiquitous Computing poses new challenges for the software engineering community. The T-Engine platform consisting of standard real-time kernel, T-Kernel, running on the standard hardware with networking facility creates broad application opportunities based on the collaboration of cutting edge microelectronics, software and embedded system technologies. However, to realize the true potential of such a system in a ubiquitous computing environment, we need to overcome software engineering issues among many hurdles we encounter. We describe such issues and the future challenges inherent in the ubiquitous computing based on our experience of using Ubiquitous Communicator terminal that is based on T-Engine.

Categories and Subject Descriptors
A.0 [General]: Conference proceedings
C.2.4 [Computer-Communication Networks]: Distributed Systems – Distributed applications.
K 6.3 [Management of Computing and Information Systems]: Software Management – Software development, software maintenance, and software process.
1.7 [Computers in other systems]: command and control, consumer products, industrial control, process control, and real time.


Keywords: ubiquitous computing, context-awareness, TRON, T-Engine, T-Kernel, realtime embedded systems, uID, eTRON, and distributed systems.

1. INTRODUCTION
Ubiquitous Computing has become very important in the computer industry and beyond [2]. It has attracted much attention from wide audience lately. Ubiquitous Computing is often referred to under different monikers:
- Invisible Computing
- Pervasive Computing
- Quite Computing
to name a few.
The essence of ubiquitous computing is “Context-Awareness”. By recognizing the conditions of its surroundings including the humans present there, computer systems can optimize operations and offer better service. This is the basic tenet of ubiquitous computing.
The reason Ubiquitous Computing has attracted much attention is several fold, but we list two major reasons here.
- Its wide applicability can affect many people.
“Context-Awareness” has become technically feasible at affordable cost thanks to the development of small RFID (Radio Frequency IDentification) tags, radio and infrared markers and other devices. This advent of microelectronics and other hardware technology has made Ubiquitous Computing practical. Ubiquitous Computing technology allows us to link virtual and real world and thus offer many possibilities beyond today's virtual world only paradigms.
We will list interesting applications made possible by Ubiquitous Computing paradigm under the section 2.2 “Application of uID Architecture” in the following section.
In this paper, we explain the interesting ubiquitous computing applications, and how we go about developing the necessary software and application for such environment. We use standard hardware and software environment called T-Engine and T-Kernel. At the end, we pose software engineering issues we will face in the coming years.

2. THE FUTURE OF UBIQUITOUS COMPUTING
“Context-Awareness” is the most important thing in the field of Ubiquitous Computing or Pervasive Computing. What exactly does it mean? It means the knowledge of one's surrounding. From

Author also serves as director at the following office.
YRP Ubiquitous Networking Laboratory
Dai 28 Kowa building 7F, 2-20-1
Nishigotanda, Shinagawa-ku, Tokyo, JAPAN
+81-3-5433-2700

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
ICSE’06, May 20–28, 2006, Shanghai, China.
Copyright 2006 ACM 1-59593-085-X/06/0005...$5.00.
the viewpoint of computer science, recognizing an object is essentially identifying something or differentiating it from something else. Status of things, people and space can be temperature, color, wind direction, fragrance, luminosity, location, or to which direction something is oriented, etc. To recognize these, sensor technology is important. We should bear in mind that identifying or differentiating something from others is fundamental to ubiquitous computing.

We also have to admit that “identifying” something can be a deep subject touching on semantics and philosophical issues. We can't enter such deep discussions here. However, from a practical viewpoint, simply being able to recognize that something is different from others goes a long way to identification, and context-awareness. If we can't recognize such a difference, then we are doomed. At least, the recent advancement of tag technology has made it possible to recognize such a distinction if it exists. Our uID architecture, described in the next section, hinges on the capability to recognize the difference of digital values.

2.1 uID architecture for Identification

We have designed uID (Ubiquitous ID) architecture. uID architecture assigns unique identifier called ucode to objects, and locations in order to differentiate unique objects (and locations). This facilitates “Context-Awareness”.

ucode is a 128bit long unique id number. The structure of ucode space is shown in Figure 1. 128 bit number can be as large as $2^{128}$, which is approximately $3.4 \times 10^{38}$. This is quite large. The number is practically limitless.

Unlike conventional product code, its value doesn't signify a special meaning or classification scheme. The use of allocated space is entirely up to the user. Many existing product code systems assign meaning to the various fields of code value such as company field, product code field, and serial number field. uID space doesn't impose such a priori structure and users utilize all the available bits according to their needs.

Since ucode is to identify objects, we should not use duplicated ucode numbers. Thus we have built a management framework to publish a unique ucode number on request. This management framework, embodied in uID center [18], also offers search or resolution function. It can resolve the inquiry request to find out where the information related to a particular ucode is located. Here again, uID center doesn't impose any specific requirement on the arrangement of data. The contents of data made available from the information server are up to the user. uID center doesn't care about it any more than it does about the way bits of uID value is used by the user.

In the uID architecture, we store ucode values in tags attached to objects and locations. What we expect of the tags is simply the capability to store the uID value for an extended period of time. This is the only requirement. RFID tags are certainly useful and the major targets of our current research. However, we can use any type of tags, be it RFID-type, or optical type such as barcode and 2-dimensional optical code such as QR code. For that matter, we can use different frequencies such as 13.56MHz, 900MHz, or 2.45GHz, and different protocols depending on the application needs.

Figure 1. ucode and ucode map.

In this sense, uID architecture is tag-agnostic. For the user convenience, we have produced a guideline to pick up suitable tags based on the physical characteristics and security needs of application.

In many feasibility studies today, we use Ubiquitous Communicator (UC for short, Figure 2) to read ucode values from RFID tags attached to objects (or location). Then ucode is sent to uID center and ucode resolution performed there. When ucode is resolved, we obtain the address of the information server where the information about the object (or place) identified by ucode attached to it is stored. Then UC retrieves the relevant information and displays it on its screen (or plays it if the information is a sound clip) (Figure 3).

Figure 2. Ubiquitous Communicator (UC).

This resolution process, i.e., obtaining information based on ucode tag value, is the basic mechanism of uID architecture.
2.2 Application of uID Architecture

We can build many interesting applications using the simple and straightforward mechanism. Integration of related pieces of information and analysis permits us to build the following applications.

- **Food Traceability System**: There are variations, but basic idea is to tag food products at the origin of produce and then making it possible to track the distribution from the farmers association, through distribution channel, market stores, and to consumers. In the end, the consumers can learn when, where and how the food product was made, and how it was delivered to the store where they bought it. “How” tells the consumers what type of pesticide or fertilizer was used during the growth of the vegetable, for instance (Figure 4).

- **Drug Traceability System**: uID can be applied to the tracing of drugs just like other manufactured goods. However, we have additional application in mind. The dispensing of incorrect drugs at the pharmacies and hospitals, and taking of incorrect pills by individuals are a major health risks.

  We have been working on to produce systems to help the medical professionals in reducing the mistake of dispensing incorrect drugs. The demo system in Figure 5 can warn the incorrect dispensing by sound and display of warning messages.

- **Free Mobility Assistance Project [3] [7] [23]**: Free Mobility Assistance Project is to aid handicapped, the aged, and visitors unfamiliar to a place to navigate the neighborhood easily.

  For example, a few of the experiments conducted so far includes the following features.

  Placement of road tiles with passive tags inside (Figure 6(a)): Visually handicapped people walk with white canes with built-in antennae to pick up location information such as where the road leads, which way to go to the station, etc.. See Figure 6(b).

  We use non-RFID tags also. Infrared markers and optical barcode marks are used to offer location information in the Free Mobility Assistance Project where appropriate. See Figure 6(c).

  Free Mobility Assistance Project requires a good geographical information system also. For example, we need the detailed road map of the neighborhood so that we can offer the shortest or most smooth (i.e., without bumps) route to a wheelchair-bound traveler (Figure 6(d)).

  For geographical information systems, we must be able to mark locations with tags (Figure 6(e)). In Figure 1, we have ucode sub-space for locations (time space ID) for this purpose.

Figure 3. Ubiquitous ID Architecture.

Figure 4. Food Produce Traceability System.

Figure 5. Intelligent Drug Checker.

Figure 6(a). Bumped tiles with passive tags underneath.
The same infrastructure to aid the handicapped can be used to help the aged. Our approach is to offer universal service or universal design for "everybody" (including the handicapped, the aged and ordinary people.).
- Sight-Seeing Guide: The uID architecture infrastructure used in Free Mobility Assistance Project can be used to offer travel guide. We have already conducted experiments in a few towns and cities, Tsuwano, Kobe, Ueno, Akihabara, Shinjuku, and Aomori to name a few [16], [19], [20].

- Supply Chain Management (SCM): We have worked with distributors of goods to keep track of the distribution and inventory. See [21], for example, for men's suit application.

- Logistics: Logistics is where RFID was first proposed as valid application. We have teamed with a few partners to experiment active and passive tags in warehouse environment [22].

We have built a house of the future using the ubiquitous computing technology. The house is called PAPI and was built near the Aichi World Expo venue (Figure 8).

These are just a few application possibilities. We have performed many of these feasibility study experiments together with T-Engine Forum and uID center within it [12].

These experiments are very important in the sense that we can learn the problems associated with the large-scale deployment of uID architecture in the real world before it is used widely and for life-critical services. We have gained insight into the following through these experiments.

- how we should physically attach tags to objects.
- legal and business problems when we introduce such new systems.
- security and privacy concerns.
- maintenance issues.
- vandalism and mislabeled objects.

3. IMPLEMENTATION

When we want to introduce new technology into society, we must consider many factors. Here we discuss three major factors.

There are legal/political/social issues when we introduce a new technology as part of the social infrastructure. We don't go into details here. (See [9] for an extended discussion.) Suffice it to say that openness of the technology (specification, architecture, etc.) is very important for various reasons. So we put openness at the base of our approach.

Second factor is the system's being realtime systems. Since we need to model and respond to the real world around us, it is natural that many components have to respond to external stimuli in a realtime manner. If a system doesn't respond fast enough, it would be unusable.

The third factor is security. Ubiquitous computing makes extensive use of network. The existing legacy networks often lack good security. So we have decided to put an end-to-end security layer on top of the legacy networks so that we can protect data if such protection is necessary.

We need to pay attention to the above three factors when we build the ubiquitous computing environment. Looking at the uID architecture depicted in Figure 3, the readers will realize building such environment boils down to the implementation of user terminals and various nodes in the ubiquitous computing environment in large numbers. In a nutshell, we need to let many developers write realtime embedded applications for a large number of nodes and terminals in a short time.

3.1 T-Engine Project

Based on these three observations, we started T-Engine Project. What we wanted to build is an open realtime embedded devices with security in mind. We figured that it would be advantageous to have a common standardized platform. This is T-Engine. T-Engine is the name of a family of computer boards for embedded systems and we offer T-Kernel, an open source realtime kernel ([5], [10]), as the embedded OS of choice on T-Engine boards.

T-Engine architecture is meant to be a standard common development platform for ubiquitous computing. We have formed
T-Engine Forum (TEF for short), a non-profit organization so that specifications concerning T-Engine are made available in an open manner over the long term [12].

TEF promotes standard specification documents for T-Engine and uID architecture. TEF was formed by 22 members in June, 2002. As of February, 2006, it has close to five hundred members from government, manufacturing, user organizations, research labs and other sectors from around the world. It is the largest among organizations active in the field of ubiquitous computing. Ubiquitous ID center is part of TEF and its main aim is to publish unique ucode and offers the ucode resolution service [18].

Again, openness is the motto of its operation. During the design stage, only the members of TEF can access the interim documents, but once they go past the final release stage, the documentation is made available to the general public. The source code of T-Kernel realtime OS is also made available (more on this later).

3.2 T-Engine Architecture
The architecture of T-Engine is centered on the realtime system kernel, namely T-Kernel. The architecture is such that middleware packages can be sold or distributed in CPU-agnostic manner. CPU-agnostic manner means that middleware packages are supposed to be portable by mere re-Compilation on various standard T-Engine CPU boards even if CPUs differ. That is our goal.

3.2.1 T-Kernel and T-License
We make the source code of T-Kernel realtime OS kernel available for free under special license called T-License. Portability issues are often raised in realtime embedded system community. The subtle difference of OS APIs is often the cause of such issues. To solve this problem, we offer T-Kernel realtime OS kernel as single source OS. A carefully maintained single repository of T-Kernel source code can be accessed and its contents can be downloaded for free by signing T-License Agreement. The binary produced by using the source code can be put into the final user product. By offering the same OS, i.e., T-Kernel with its standard API for system calls, and standard driver structure, we try to ensure that middleware can be used on different CPUs by a mere recompilation.

T-License is different from other license for free software such as GPL [1] in the following sense that you do NOT have to make the change to the source code available to a third party. Actually such diversification is discouraged because we want to make sure that APIs of T-Kernel OS version in products is unique and do not differ across implementations. The major reason why T-Kernel source code is made available is to offer a unique interface for systems to improve portability and increase productivity of software development. This is important to make the time-to-market very short.

In embedded market, the custom hardware used in the whole system and the software driver to use such hardware contain the technical know-how of a commercial concern and it can't disclose the source code to outside world at all because of this competitive advantage such proprietary information offers. So T-License allows the linking of proprietary software components with it and doesn't require such software to be made available in source code form either.

In this manner, T-License has been written to be very friendly to the needs of embedded system developers. You can read T-License at leisure. See [13].

3.2.2 T-Engine Software
T-Kernel is just a kernel of a realtime OS for embedded systems. It doesn't come with niceties such as debugging support and boot ROM, etc.. We expect our partners in TEF to supply such add-ons. Already, such service is available. (See TEADEC Website [11], [14] and [15] for example.)

Also, T-Kernel, being a mere realtime kernel, doesn't offer the upper layer such as file system, process management and network support which is the foundation for large-scale advanced embedded systems. We are in the process of finalizing the definition and releasing an implementation of SE (Standard Extension) that offers such upper-level functions. With SE and low-level support such as bootloader, an advanced embedded system can be built with ease (Figure 9).

3.2.3 T-Engine Hardware
From the stand point of T-Kernel and application that runs on top of it, the underlying hardware must offer a clearly defined interface so that middleware packages can access hardware features. We have defined such hardware layer for T-Engine, a family of embedded CPU boards, to run T-Kernel and the middleware packages.

T-Engine family motherboards support major CPUs for embedded systems. CPUs include MIPS, ARM, Renesas SH, Fujitsu FR, Infineon TriCore and others. There are different boards with varying degree of performance and different features such as low power consumption vs. sheer CPU MIPS value.

3.2.4 Developing Software for UC Terminal
Ubiquitous Communicator explained in section 2 is built using a T-Engine board and T-Kernel at its core. Current version of Ubiquitous Communicator terminal (Figure 2) grew out of various experimental versions. Its hardware is derived from T-Engine architecture. But UC terminal uses a motherboard with a different form factor, etc. because UC is a delivery platform while standard T-Engine CPU board is basically a development platform that can also be used as delivery platform sometimes. It is natural that delivery platforms are customized to meet the size and form factor requirements of applications. UC terminal is no exception.
UC terminal comes with
- RFID tag (μ-chip) reader,
- IEEE 802.11b wireless LAN,
- VGA capable LCD touch screen display,
- CCD camera (for image and barcode scan), and
- built-in speaker, and microphone (for IP-phone).

It runs open source T-Kernel. Depending on the complexity of the software applications that run on it, we use middleware such as T-Shell [4] to support advanced GUI (Graphical User Interface), TCP/IP driver, and full-featured web browsers. (There are a few web browsers that run on T-Engine). Since the standard T-Engine and UC terminal share the same architecture, an application developer can port software from the standard T-Engine to UC terminal with relative ease. This is a direct result of our design goal and is a definite plus for software development.

4. eTRON

![Figure 10. eTRON SIM chip socket on T-Engine Boards.](image)

One notable hardware feature of T-Engine Board is a socket for a SIM chip called eTRON (Figure 10). eTRON is a name given to network security architecture to send and receive valuable data such as electronic money or ticket across insecure network using cryptography [6]. As we mentioned in the earlier section, the legacy network system often lacks good security. We try to offer end-to-end cryptography using eTRON architecture.

eTRON architecture relies on a physically tamper-proof hardware chip to store secret data such as cryptographic key. By using an IC SIM chip based on eTRON architecture, a computer node based on T-Engine can use service offered by a SIM chip that is based eTRON architecture.

Driver to access functions of an eTRON chip is part of T-Engine architecture. Middleware to access cryptographic functions offered by eTRON architecture has been built. Through the feasibility study experiments, we have gained insight to design the next generation of eTRON chips.

5. SOFTWARE ENGINEERING ISSUES

Ubiquitous computing environment is a very large complex distributed software environment. We wanted to make it easy to develop software for such environment, and thus we have designed T-Engine as an open standard development platform.

We have made this and other efforts to make development easy. Development has become easier thanks to such efforts. However, there are still fundamental issues that must be addressed. Problem with the creation of software applications of ubiquitous computing is very severe.

5.1 Large Realtime System

First of all, ubiquitous computing environment is a large realtime distributed system. There is no way to test the whole system in controlled repeatable conditions. How do we test and make sure such systems work?

5.2 Reliability

No single organization can produce many systems used in ubiquitous computing. This means that there are many servers and terminal nodes in ubiquitous computing developed by many entities. How can we increase the reliability of such an environment? How do we track down and debug the cause of problems then have it fixed?

The system as a whole has a large impact on people's lives since the environment will be part of social infrastructure. As some of the applications explained in section 2 implies, there will be services in which many people put trust. To improve the quality and reliability of the components and system as a whole, we need the expertise of software engineering community in the next several years.

5.3 Legacy System

For today's IT professional, we have a pressing issue such as how we incorporate the existing legacy applications in the framework offered by ubiquitous computing in the future. uID architecture can assist production database applications by embedding the product code in uID code space. But will this be enough? We will find out in the next several years.

Of course, when we built the software server systems for our feasibility experiments, we used the current state of the art technology such as object-oriented paradigm, web server/client, Java applet, or whatever were deemed appropriate for the job at hand.

5.4 Other Issues

From theoretical viewpoint of computer science, since context-awareness is the key to ubiquitous computing, we need to describe the environment conditions under which particular actions are triggered in applications. Thus, we need to model the environment in a formal manner using model language or modeling system. We are carrying out research in this direction. This will be an interesting topic for the next few years.

For secure eTRON chip, the formal verification of protocols is an important issue. But so is verifying the implementation of protocol in a very resource-poor environment.

These chips, which often have powerful co-processors for public key cryptography calculations don't have more than just a few KB of RAM. In this space, we need to squeeze the input/output buffer, stack, and global variables, and do PKI protocol processing, and correctly at that.
This poor resource situation is seen across almost all small hardware components of ubiquitous computing. Ubiquitous Computing is not a place for developers who have written bloated code and quipped that throwing hardware will take care of the rest. Formal verifiers to guard for stack overflow and such will be very valuable. We feel that software engineering community, as well as hardware community, may have failed to provide good software tools to help the file and rank of embedded system developers who need to deal with such resource-poor, yet cutting edge devices.

6. CONCLUDING REMARKS

Ubiquitous computing has gone through very fast development phase and it will be fully established in 10 years. This is our prediction. We are trying make this prediction come true by proposing uID architecture and implementing many systems. When Ubiquitous Computing becomes a part of social infrastructure, we need to view it as a very large scale realtime distributed system. Development of such a distributed system is very difficult. We have tried to ease the difficulty by offering an open standard development platform, namely T-Engine. However, fundamental difficulties still remain. We would like to draw the attention of software engineering community to address such tough issues. We explained the nature of possible ubiquitous computing applications to give you some ideas about such problems.

We would like to stress that the “OPENNESS” that allows many people to participate and contribute to ubiquitous computing infrastructure is very important to make a large-scale social-infrastructure using RFID tags and ubiquitous computing technology such as uID architecture in general.

Software engineering community has grown out of the early pioneers who needed to build large-scale time sharing systems, online transaction systems and such.

But now we face a very different paradigm, i.e., ubiquitous computing in which many resource-poor nodes interact with each other and remote big servers. This is an interesting paradigm and we believe that software engineering community can offer its insight gained over the years to solve problems before we adopt Ubiquitous Computing fully.

7. REFERENCES

[18] Ubiquitous ID Center: www.uidcenter.org
[21] uID Center, Start of feasibility study experiments with the largest specialty men’s clothing shop, Aoyama Trading, aiming at the full-scale implementation of uID technology., http://www.uidcenter.org/english/press/TEP051201-u01e.pdf
[22] uID Center, IC tag demonstration experiment - jointly conducted by Ubiquitous ID Center and NYK Line in Ohi Logistics Center – succeeds ~ Cargo number management and highly accurate 3D cargo location management in warehouse, achieved ~., http://www.uidcenter.org/english/press/TEP040414e.pdf