Challenges and Lessons Learned in Building a Practical Smart Space

Yoshihiro Kawahara[1], Masateru Minami[2], Shunsuke Saruwatari[1],
Hiroyuki Morikawa[1] and Tomonori Aoyama[1]
1) The University of Tokyo, 2) Shibaura Institute of Technology
kawahara@mlab.t.u-tokyo.ac.jp, minami@sic.shibaura-it.ac.jp,
{saru, mori, aoyama}@mlab.t.u-tokyo.ac.jp

Abstract

Ubiquitous computing as the integration of sensors, middleware, and networking technologies to form a "smart space" environment relies on the development of both software and hardware solutions. For over 3 years, our group has been developing a smart-space environment, involving the exploration of core technologies and attractive applications. In this paper, we introduce the challenges faced and lessons learned in designing and developing a smart-space environment, including device control services, locating systems, wireless sensor nodes, and middleware services.

1. Introduction

The notion of ubiquitous computing is quite distinct from the conventional computing paradigm in that it implies the provision of services via a network of invisible computers. In a ubiquitous computing environment, tiny devices such as sensors are embedded in living spaces to collect real-world information and build a model of the real world in order to provide computer-controlled services. Using this repository of user and world contexts, users can reap the benefits of smart and attentive services provided by embedded computers.

The realization of ubiquitous computing relies heavily on the miniaturization of computers and improvement of communication protocols and power supply technologies. However, it is also necessary to develop novel services to complement these new technologies and facilitate the development of new core technologies. This situation is quite similar to that surrounding the birth of the Internet. New services and applications are inspired by the use and exploration of the new technologies, and vice versa. Tim Berners-Lee, the inventor of the World Wide Web (WWW), stated that "I just had to take the hypertext idea and connect it to the TCP and DNS ideas and the World Wide Web." [1] This implies that most novel inventions are supported by existing technologies. In addition, the use of new services triggers the invention of new core technologies. A good example is the search engine algorithm, which was developed as the WWW was deployed. In the research and development of ubiquitous computing, synergistic effects can therefore be expected between the creation of services and the development of core technologies.

Since 2001, our group has been constructing a smart-space environment, which has involved the exploration of core technologies and attractive applications. Through this experience, we have learned many lessons and developed several practical systems and core technologies. Most importantly, this experience has lead us to conclude that the development of both software and hardware is indispensable for ubiquitous computing, which is realized as an integration of various components including sensors, middleware, and networking technologies.

1.1. Related works

Several other research programs are currently in progress with a shared perspective of exploring a new research agenda. These include iRoom [1], the Aware Home [3], Easy Living [4], Smart-Its [5], and SSLab[6], which are all practical approaches to realizing smart environments. Although our project shares a similar philosophy to these other programs, the underlying technologies, goals, and lessons learned are different. We believe it is meaningful for each project to report on their respective experiences and the solutions developed. In this report, we present the design considerations for our testbed room, the development of our distributed object localization system for physical-space internetworking (DOLPHIN) ultrasonic locating system, the U3 sensor node, and STONE middleware.

2. How we made things smart

In the absence of a formulaic method for building a smart environment, we began by trial and error from some basic principles. The testbed for this project was a room of about 100 m^2 in area (8.6 m × 12 m; Figure 1) with steel trellises installed on the ceiling to allow the easy attachment of various devices. The room was divided into three main spaces, a working desk environment, a collaborative meeting space, and a relaxing living space. Up to 10 researchers actually work in this room everyday.

Figure 1 STONE room

The first step in the construction of our smart environment was embedding devices into the environment logically. This involved the connection of computers, audio-video equipment, and lighting to a network. The appliances in the room were modified using an off-the-shelf multipurpose infrared remote control ("crossim"), which was connected to a computer via an RS-232C interface to allow the appliances to be controlled via the network. The infrared signal transmitter was extended in order to ensure the signal reached all areas of the room.

2.1. Smart Baton System

Although audio-visual equipment was connected to the network, it simply was not worth the effort to boot the computer and open a browser in order to turn on a television in front of the user. Moreover, when a large number of appliances are connected, selecting the appropriate device from the list can also be quite troublesome.

To overcome this problem, we developed a Smart Baton System, which provided the following features in order to overcome these inconveniences:

(1) Explicit and easy selection of appliances

(2) Provision of an appropriate user interface for each appliance

(3) Support of multi-user operation

(4) User identification and realization of differentiated services for each user

Although several middleware architectures have been developed for ubiquitous computing environments, such as Jini [7] and UPnP [8], these existing architectures do not satisfy the first requirement since their directory service-based approach does not allow users to choose appliances intuitively. Infrared remote controls are preferable in terms of intuitive manipulation. However, infrared remote controls usually provide only a poor user interface and do not allow for user customization [9], thus not satisfying the second requirement. A system such as that reported by Olson and Nielson [10] allows users to explicitly choose an appliance with a laser pointer via various user interfaces. However, as the laser spot is recognized by image processing, the system allows only one user to control appliances at a time, and as such it does not satisfy the third and fourth requirements.

In order to satisfy all requirements, we developed the Smart Baton System as a laser-pointer-based manipulation technique. The system differs from similar techniques in that it is possible for users to download an appropriate user interface to a handheld device via the network, and target appliances are able to distinguish between multiple users and provide differentiated services.

2.1.1. System

An overview of the Smart Baton System is shown in Figure 2. The system consists of smart batons, smart baton-compatible appliances, and a certificate authority (CA). A smart baton is a handheld device equipped with a laser pointer, and is used to control appliances. A smart baton-compatible appliance has a laser receiver and is connected to the network. The CA is used to authenticate and identify users and appliances. Figure 3 illustrates the function diagram for the smart baton and an appliance. The user chooses an appliance by pointing to its photo detector with the laser pointer, and the smart baton sends information by the laser beam. The appliance detects the beam and obtains the information via its laser receiver, identifies the network address of the smart baton, and establishes a network connection to the smart baton. An authentication process follows to establish the user's identity, allowing so the appliance to provide different

user interfaces and services for respective users. For example, the system can prevent children from turning on a television at night while allowing adults full control.

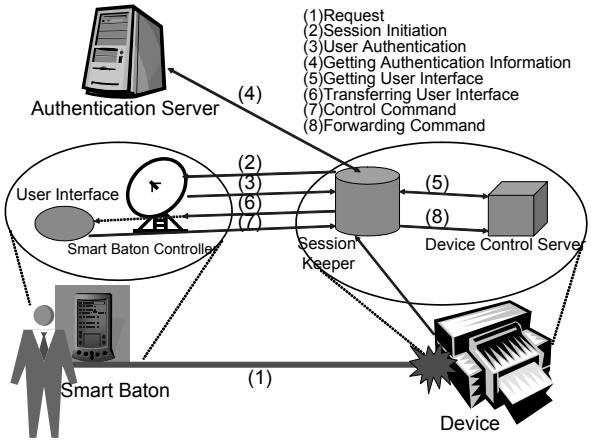


Figure 2. Smart Baton System

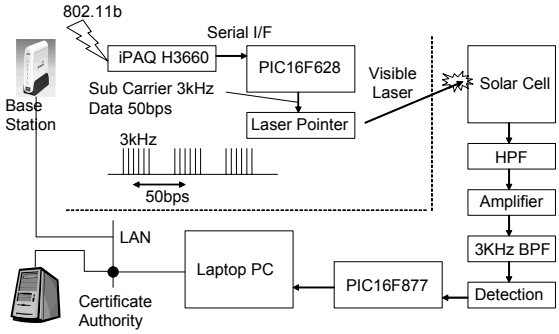
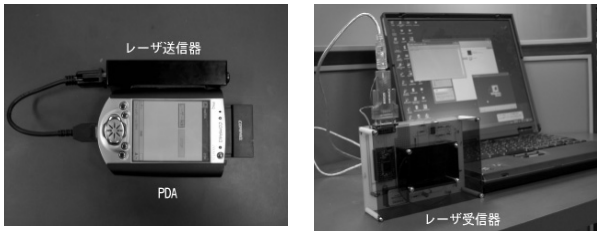


Figure 3. Hardware function diagram

2.1.2. Security Considerations

In a ubiquitous computing environment, appliances will exploit user information in order to provide flexible services, which may mean users will provide appliances with private information. In such a situation, there may be a concern about access to this private information when it is transferred to the appliances. In the Smart Baton System, there are two communication channels between users and appliances over which information is transferred; the laser beam and a network connection. The former is vulnerable to impersonating appliances, such as a fake appliance set to collect information received by a laser beam. To avoid such security problems, the laser beam conveys only unimportant information, such as the Internet protocol (IP) address of the smart baton, the TCP port number, and a randomly generated session identifier. As the TCP port number and session identifier are one-time values, the IP address of the smart baton remains the only potential target of theft, but this is not a serious issue because the IP address can already be easily revealed via the domain name server (DNS).

The network connection is vulnerable to wire-tapping, but that can be easily prevented employing an encryption technology. In our prototype implementation, all communication over the network is encrypted by secure sockets layer (SSL) encryption, making it virtually impossible for someone listening on the network to steal private user information. This measure also makes the system resistant to tampering, and the authentication of appliances prevents impersonation attacks on the network.



(a) Smart Baton (PDA) (b) Information Appliance

Figure 4. Prototype system

2.1.3. Implementation

Figure 4 shows a prototype implementation of the Smart Baton System. In this implementation, a COMPAQ iPAQ Pocket PC H3660 equipped with a custom laser transmitter is employed for the smart baton, and a laptop computer equipped with a laser receiver is adopted as a smart baton-compatible appliance. The laser transmitter consisted of visible laser device and a microcontroller (PIC16F628, Microchip Inc.), which is controlled via the iPAQ's serial interface. The laser transmitter sends data via the visible laser at 200 bps upon receiving data from the iPAQ. The laser receiver consists of a photodetector (solar cell), a signal-processing circuit, and a microcontroller (PIC16F877, Microchip Inc.), and sends received data to the laptop via a serial interface.

In this implementation, HTML and HTTP are used as the basis for the user interfaces in order to simplify the system. When an appliance serves a user, a Device Control Server sends the URL of the user interface to the smart baton, and a web server runs the appliance. On the smart baton, a web browser is invoked and accesses the URL, and the user then controls the appliance via the web browser. As the system can distinguish between users, appliances can provide

differentiated services for each user. For example, by checking the user's age, the system can provide a remote control interface that prohibits access to prohibited programs.

3. Obtaining Contexts in Real World Environments

Our next challenge was to develop a platform that included a computer model of the real-world environment. This system required the implementation of a locating system and the deployment of sensor nodes.

3.1. DOLPHIN

In a ubiquitous computing environment, the physical location of indoor objects is key information for supporting context-aware applications. Several positioning systems have been proposed for obtaining indoor location information, including Active Bat [11] and Cricket [12], which use ultrasonic time difference of arrival (TDOA) data to measure the three-dimensional position and orientation of objects in an indoor environment with high accuracy. However, such ultrasonic positioning systems require additional hardware for the transmission and reception of ultrasonic pulses, and usually require manual pre-configuration of the locations of reference beacons or sensors. The setup and management costs of such a system would be unacceptably high if the system were to be applied to a large-scale environment such as an office building. An ad-hoc localization mechanism [13] can be applied to such problem. In [13], the authors proposed a collaborative multilateral algorithm to solve the localization problem in a distributed manner, and performed a detailed simulation-based analysis of a distributed localization system. To design a practical location information infrastructure, we believe that experimental analysis is also required in order to discover practical problems with the distributed localization system.

From this point of view, we developed a distributed positioning system called the Distributed Object Localization System for Physical-space Internetworking (DOLPHIN), which determines the position of an object using only a few manually configured references. The system was constructed from off-the-shelf hardware devices, and represents a simple but practical distributed positioning algorithm.

3.1.1. Positioning Algorithm

Figure 5 shows an overview of the DOLPHIN system. The system consists of a number of DOLPHIN nodes consisting of a 2400 bps radio frequency (RF) transceiver for time synchronization and message exchange among nodes, several 40 kHz omnidirectional ultrasonic transducers, and a Hitachi H8S/2215 16 MHz microprocessor for calculating the location of the nodes.

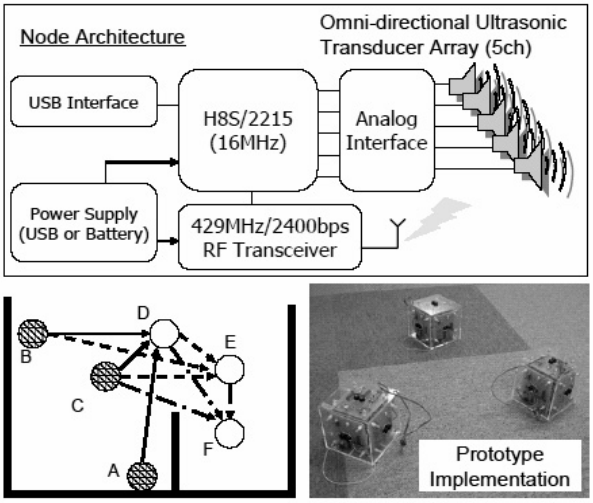


Figure 5. DOLPHIN system

The key premise of our positioning algorithm is hop-by-hop localization. For example, in the bottom left of Figure 5, node D can determine its position by receiving ultrasonic pulses from the reference nodes A, B, and C. However, nodes E and F cannot receive ultrasonic pulses from reference nodes due to physical obstacles such as a wall. Here, if the position of node D is determined, and node E can receive ultrasonic pulses from node D, node E can compute its position using the distances from nodes B, C, and D. If the locations of nodes D and E are determined, node F can compute its position using nodes C, D, and E. In this way, all nodes in the DOLPHIN system can be located. There are two main advantages to this mechanism. First, the system requires only a few (minimum three) nodes to determine the positions of all nodes. Second, nodes can determine their positions even if unable to receive ultrasonic pulses from reference nodes directly.

The positioning algorithm runs by exchanging several messages as shown in Figure 6: and identification (ID) notification message (IDMsg), measurement message (MsrmtMsg), and location notification message (LocMsg). The nodes in the system are assigned such that there is one master node, one transmitter node, and

the rest receiver nodes. Consider the example depicted in Figure 5, where nodes A, B, and C are reference nodes, and nodes D, E, and F are normal nodes (the positions of the nodes are unknown). Here, we assume that nodes A, B, and C have node lists [B, C], [A, C], and [A, B], respectively. We also assume that node E and node F are unable to receive ultrasonic pulses from node A because of an obstructing wall.

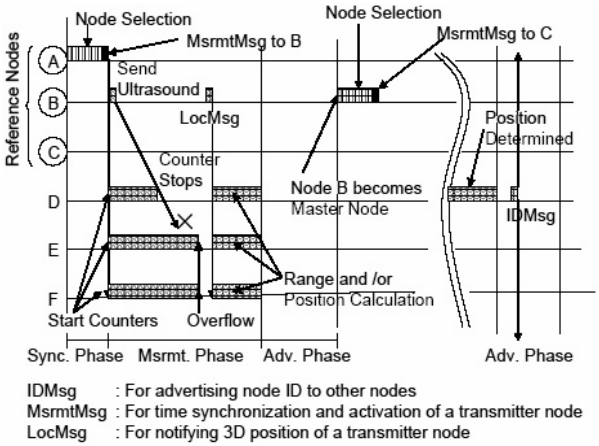


Figure 6. DOLPHIN positioning algorithm

With node A acting as the master node, Figure 6 shows the timing chart for the positioning algorithm. First, node A chooses one node randomly from its node list [B, C]. If node B is chosen, node A transmits MsrmtMsg including the ID of node B. On receiving the message, node B becomes the transmitter node and generates ultrasonic pulses. At the same time, nodes C, D, E, and F become receiver nodes and start internal counters (synchronization phase). When a receiver node detects an ultrasonic sign from node B, it stops its internal counter and calculates its distance from node B. After several milliseconds, depending on the time taken by the overflow of the internal counter, node B sends LocMsg to notify the receiver nodes of its position. The receiver nodes that are able to detect the ultrasonic signal from node B store the location of node B and their distances to node B in their position table (measurement phase). All nodes then listen for IDMsg for several milliseconds (advertisement phase). If there is a node that could determine its position based on three or more distances, it advertises its ID in this phase. This ID is added to the node list of every other node.

In the above example, because nodes D, E, F cannot determine their positions, no IDMsg is sent by those nodes in this phase. This sequence of phases defines one cycle of the positioning algorithm in the DOLPHIN system. In the next cycle, node B, which acted as a receiver node in the previous cycle, becomes the master node, and the positioning algorithm proceeds in the same manner. After three or more cycles of positioning, node D can determine its position based on measured distances from nodes A, B, and C. At which time, node D can send its IDMsg in the advertisement phase. All other nodes that received the IDMsg from node D add the ID of node D to their node lists, and node D is recognized as a candidate master node. After node D becomes master node, node E and node F can measure their distances from node D. Then, node E can determine its position and advertise its IDMsg. Finally, based on nodes C, D, and E, node F can determine its position. In this way, we can locate all nodes in the DOLPHIN system.

In the DOLPHIN system, it is necessary to prepare for node failure. To recover from node failure, each node in the system has a recovery timer and an advertisement timer. The recovery timer is set when nodes receive MsrmtMsg, and expires if there has been no MsrmtMsg for a certain period. If the recovery timer expires, a node in the system is chosen randomly to become master node, and the positioning algorithm resumes. If a candidate node does not receive MsrmtMsg from other nodes within a certain period, the advertisement timer in the node expires, meaning that the node is not recognized as a master node by the other nodes. In this case, the node retransmits IDMsg in the advertisement phase of each positioning cycle.

3.1.2. Experimental Result

As an experimental evaluation of the algorithm, we placed seven nodes as shown in Figure 7, and computed the average and variance of the measured position of each normal node (nodes D-G) over 1000 cycles. The results revealed that the system could determine the position of objects with an accuracy of around 15 cm in a real indoor environment. However, the positioning error for nodes E-G was higher than that for node D, as the positioning error at node D affects the position determination of nodes E-G, which determine their position based on node D. Although this error propagation problem is inherently unavoidable in the DOLPHIN system, we expect to minimize positioning error by placing reference nodes at appropriate locations.

Our group has obtained a large amount of data through experimentation in our laboratory. We are currently designing an improved version of the positioning algorithm that can handle practical problems such as multipath propagation and node mobility.

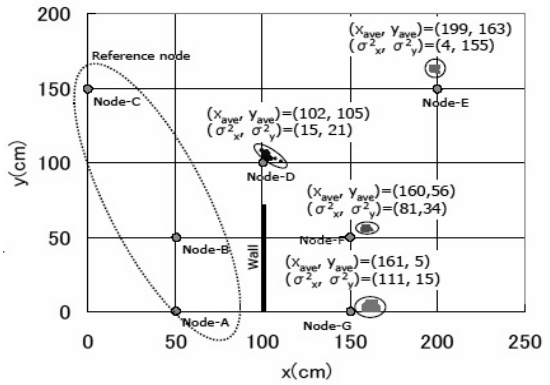


Figure 7. Experimental results

U³

In addition to location information for users and objects, real world information is helpful for context-aware applications. When designing a practical sensor network architecture for future ubiquitous computing environments, it is desirable that the requirements of future applications be made clear. However, it has been difficult to envision what such future applications will be like. To assist in the implementation and evaluation of prototype applications and to clarify technical challenges, we have developed a sensor network development testbed called U³ (U-Cube) that allows developers to flexibly implement various applications. The U³ device is a 50 mm cube containing a power module, a microprocessor module, an RF communication module, and a sensor module. The first implementation of U³ is capable of sensing several types of data such as temperature, brightness, and the presence of humans, and can send this information to other nodes and/or peripheral devices including computers and PDAs.

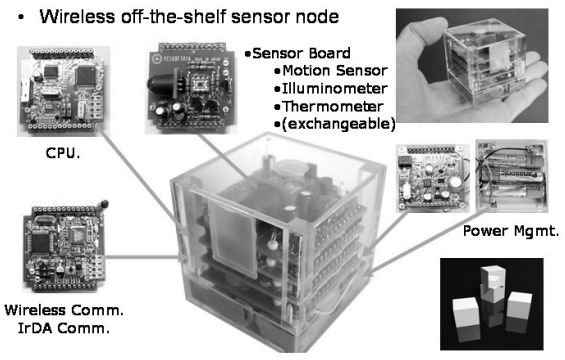


Figure 8. U³ hardware components

3.1.3. Hardware Design

There is a wide range of potential applications and protocols for sensor networks. For environmental monitoring applications, it would be useful if the wireless nodes could provide a generic sensor interface, which would allow the sensors to be easily replaced. For practical reasons, it would also be preferable to be able to provide solar panels and rechargeable batteries. Moreover, users might want to choose more appropriate computing and wireless communication devices according to the power consumption and required processing/transmission speed. To meet these kinds of requirements, the hardware components of the sensor node should be constructed as an ensemble of independent modules that can be easily replaced. However, conventional sensor network nodes [13][15][16] such as MICA Mote only allow replacement of the sensor board. Other components such as the microprocessor and wireless communication module are not replaceable.

In order to achieve this flexibility, we divided the functionality of the sensor node into four physically separated modules: a power control module, a processing module, a communication module and a sensing/actuating module. Each module is then connected to others by a bus connector to achieve extensibility.

3.1.4. Software Design

Due to the small physical size of the sensor nodes and the limited power consumption, software on the sensor node hardware must make efficient use of processor and memory while providing low-power communication. In this section, we describe task scheduling and application programming interface (API) layering in the sensor node.

1) Event and task scheduling

In general, one sensor node plays several roles in the sensor network. For example, information may be simultaneously captured by sensors, manipulated, and streamed onto a network. Alternatively, data may be received from other nodes and forwarded in multi-hop routing or bridging situations. To realize such concurrent execution using such resource-restricted hardware, we allow the interruption of tasks by events. Here, an event is defined as a process that must be executed immediately and is assumed to complete immediately, such as the arrival of wireless packets. A task is defined as a process that takes longer than an event to complete, such as the periodic capturing of environmental data. Event-based task scheduling incurs only a low overhead for state transition compared to a stack-based approach. Accordingly, this scheme reduces processor load and power consumption.

A media access control (MAC) algorithm significantly improves the system performance of the sensor nodes. Specifically, MAC ensures accurate synchronization between sending and receiving nodes, which is crucial for high-speed, reliable transfers. As application tasks can be frequently interrupted by wireless communication events, an additional microprocessor dedicated to wireless communication and application tasks is provided to reduce the load on the main processor.

2) Functional layering

The primary objective of U^3 is to provide a testbed sensor network for the development of ubiquitous computing environments. Having a testbed development environment is essential for the innovation of attractive applications. For example, if a user wants to design an appropriate communication protocol for a specific application, it would be useful if various APIs for controlling communication functionality could be provided in the development kit. While TinyOS [17] is a component-based runtime environment designed to provide support for embedded systems such as MICA Motes (UC Berkeley), it does not provide a separate communications module. To date, we have defined several APIs that provide layered abstracted communication functions (application, media access, physical layer of RF and IrDA 1.0). This layering allows developers not only to reuse the APIs but also to concentrate on the development and evaluation of the specific functions or protocols in which they are interested. Of course, it is conceivable that conventional layering may no longer be valid for sensor networking. In that case, however, developers can choose not to use the APIs and conventional layering.

Communication Module: this module consists of an RF monolithic transceiver (300 MHz band, On-Off keying, 115.2 kbps), a helical antenna, and second PIC microcontroller for processing network protocols. The current implementation realizes data transmission of up to 100 kbps with our current implementation of the carrier sense multiple access/collision avoidance (CSMA/CA) protocol. The transmission range is within a radius of 30 m. The communication module hosts a dedicated processor to drive the RF transceiver. As mentioned in the previous section, it is possible to implement various MAC and network layer protocols using this processor.

Sensor/Actuator Module: the sensor module is dedicated to obtaining information about the surrounding environment. The implementation of the sensor node only includes a motion sensor, a brightness sensor, and a thermometer, but various transducers can be connected to the board via a generic bus connector.

3) Software Implementation

Figure 9 shows the software architecture of U3. We adopted C as the programming language due to the wide availability of commercial C compilers, which provide plenty of built-in functions for handling features such interrupts.

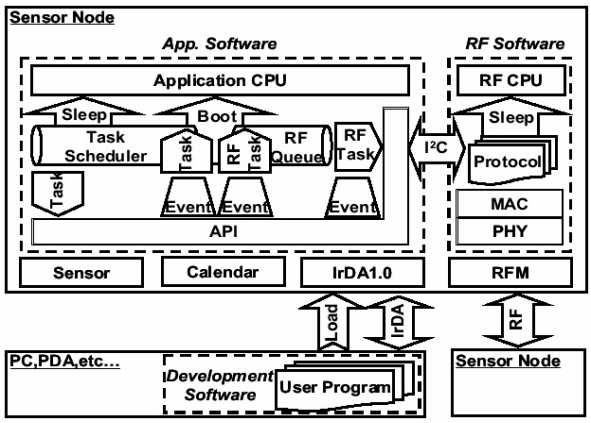


Figure 9. U^3 software architecture

The U^3 software consists of the application software, wireless communication software, and a development environment. The application software and wireless communication software are implemented in PICs on the main processor board and the wireless communication board, respectively. Messages between the software travels via the I^2C bus.

Application software controls the sensor and actuating devices, and forwards processed data to the wireless communication software. Multiple tasks are scheduled in a first in, first out (FIFO) task scheduler. The application software also provides APIs for communicating with a computer or PDA (via IrDA), as well as other nodes (via the RF module). After all tasks in the scheduler have been processed, the system sets the main processor in sleep mode to save power. Timer events generated by the calendar integrated circuit (IC) and packet arrival events generated by the IrDA chip and the wireless communication software can wake the main processor.

The wireless communication software provides APIs that bridge the application processor and the wireless RF modules. The wireless communication software includes a routing protocol and MAC protocol, which

are implemented independently such that each of the protocols can be replaced according to user requirements. In the physical layer, we provide periodic transmission and reception control for the RF communication module. The RF module transmits data by amplitude shift keying (ASK), where each bit is encoded with the Manchester Code, and encoded data is sent after the transmission of a preamble (1 Byte) and header (2 Bytes).

3.1.5. Related works

MICA Motes and U^3 share a similar motivation: to develop an off-the-shelf sensor network testbed. However, they differ on several points. Although MICA Motes and U^3 use similar RF wireless communication modules for communication between nodes, the frequencies used differ due to legal regulations (the 916 MHz used by MICA Motes is restricted to transmission over less than 1 m in some countries including Japan). In U^3, we use two PICs, as a processor and network controller. Compared to the ATMEL 90LS8535 controller installed in MICA Motes, PIC has advantages with respect to power consumption, particularly with the sleep mode feature. Due to the use of these two controllers, U^3 therefore has a potential advantage in terms of power consumption. In addition, MICA Motes is only equipped with replaceable sensor board; in U^3, both the sensor board and main processor, network and power modules can be replaced. IrDA communication with other peripheral device is also unique to U^3. Finally, TinyOS as used by MICA Motes provides event-based development environments, but does not provide layering of the network stacks due to the monolithic networking module.

4. Middleware technologies

The technologies we have introduced to make an environment "smart" represent fundamental components for realizing ubiquitous computing, but a "glue" for creating applications and services is required. This glue is implemented in the form of middleware to coordinate the ubiquitous devices on the network, and forms an indispensable part of the system. As ubiquitous computing environments can be expected to be highly dynamic, heterogeneous, and context-dependent, the functionality of applications should be able change depending on the dynamically changing user context. For example, when a user wishes to brows a portable document format (PDF) file on a PDA display, the document-browsing application running on the PDA will require an additional transcoder function to reformat the PDF file. If the user wishes to use a voice-only device such as a personal digital cellular (PDC) to obtain information in the PDF document, the application will need to locate and use a PDF-to-text function as well as a text-to-voice function. Thus, to enable ubiquitous Internet applications with adaptable functionality on the fly, a mechanism capable of finding and synthesizing the appropriate functions transparently on the Internet is necessary.

4.1. STONE

Our group is currently developing the STONE network service platform, which dynamically synthesizes a desired context-aware service from a set of resources. STONE provides service discovery, context awareness, service synthesis, and service mobility in a unified way using a naming service [18][19].

Figure 10 depicts the STONE architecture. STONE has three major components: a functional object, a service resolver, and a service graph. The functional object is the most basic element of a service, and has the mechanisms required for providing the requested service by dynamic linking to another functional object. Functional objects may be either hardware or software, and include objects such as the display, a camera, speaker, microphone, various types of transcoders and proxies, and streaming videos. A synthesized service is a string of functional objects such as the dynamic combination of functions of the source of world news, a transcoder, and a display.

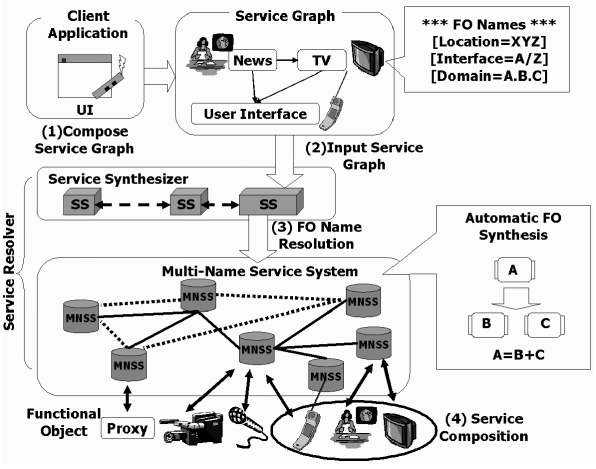


Figure 10. STONE architecture

In this system, even if a change in the environment around the functional object or user occurs, the service

can be maintained and be transparent to mobility or failure as long as the functions composing the service can be maintained (i.e., distribution transparency). Alternatively, the service provided can be tailored to the environment by modifying one function into a more appropriate function as the environment changes (i.e., context awareness).

STONE achieves distributed transparency and context awareness through appropriate naming of functional objects. Networking systems are traditionally organized using a layering model composed of applications, transport/networks, and link layers. This model is useful for clearly defining the responsibilities and restrictions of software that exists at each level. To be implemented fully, a layer requires a naming scheme, a means of resolving those names, and a means for routing communications. On the Internet, the naming types used in each layer include MAC addresses in link layers, IP addresses in network layers, and URLs and email addresses in the application layers. In STONE, we extend the model to include a new top layer.

STONE adopts location-independent naming for describing the objects being searched for by users and/or applications, not the location of those objects. Current naming types for IP addresses and URLs specify the network location of server and client machines, and as such these names are location-dependent. The advantage of location-independent naming is to be able to achieve distributed transparency with respect to access, location, failure, and replication. Location-independent naming allows nodes that provide a function to precisely describe what they provide and users to describe what they require. This makes it possible to achieve service discovery, context awareness, service synthesis, and service mobility in a unified way. The following is an example of STONE naming. Every name is represented as an attribute-value pair, and includes an interface name for describing the function of a functional object.

```
[FO Name =
  [Location=x.y.z@myhome.net],   //Physical Location
  [InterfaceName=                //Function Description
    [Output Interface = Rendered Video],
    [Input Interface = MPEG4/IP],
    [Relation = Convert Input Interface to Output Interface],
      [Ctrl Interface = Display Control/GUI]
  ]
]
[Access Pointer List= [Address=xx.xx.xx.xx:yy],  //IP+Port ]
```

Location-independent naming, however, often has a flat name space, resulting in a scalability problem, in contrast to location-dependent naming such as DNS which has a hierarchical name space. The introduction of interface names in STONE naming mitigates the scalability problem by grouping interface names to form a two-level hierarchy.

The service resolver network overlaid on the Internet is used to route a request to the appropriate locations by maintaining a mapping between interface descriptions and their network locations. The service resolver network is a logical overlay network, and finds and connects functional objects using interface names. As an IP router routes data by examining its destination IP address, the service resolver routes data by examining its interface name.

The service graph specifies the service request of a client, for example,"I would like to see the camera images for room 409 on the nearest monitor". The service graph may be created by the client, or may be downloaded from the network. It describes the interconnection between functional objects (e.g., to connect a camera output function with a monitor input function), and a context script to specify context awareness explicitly (e.g., to select the output function of the nearest monitor). When a user issues a service graph, STONE finds suitable functional objects and synthesizes the requested service by combining several functional objects dynamically in a context-aware manner.

We have implemented STONE in a testbed room and constructed several application prototypes, including mobile video conferencing, a 'connect to' service, and a 'media kitchen' service. The locations of objects and people can be determined using the indoor positioning system installed in the testbed room.

5. Conclusion

In this paper, we have introduced our approach for realizing a ubiquitous computing environment. Our smart space testbed is comprised of networked appliances, a device control system (Smart Baton), an ultrasonic location system (DOLPHIN), wireless sensor nodes (U^3), and service synthesize middleware (STONE). This testbed is unique in that research on both hardware and software are conducted using a single testbed environment with coordination between research topics.

Our next step is to develop a variety of context-aware applications using these technologies, and deploy these applications in the smart space. It is only through such

practical experimentation that the real functionality of ubiquitous computing can be realized and new research opportunities be discovered.

6. References

[1] Berners-Lee, T., "Answers for Young People," http://www.w3.org/People/Berners-Lee/Kids

[2] Johanson, B., Fox, A., and Winograd T., "The interactive workspaces project: experiences with ubiquitous computing rooms," IEEE Pervasive Computing, Volume 1 , Issue 2 (April 2002), pp. 67 – 74, 2002

[3] The Aware Home, http://www.cc.gatech.edu/fce/ahri/

[4] Brumitt B., Meyers, B., Krumm, J., Kern, A. and Shafer, S., "EasyLiving: Technologies for intelligent environments," Proc. Handheld and Ubiquitous Computing, September 2000

[5] The Smart-Its Project, http://www.smart-its.org/

[6] Smart Space Laboratory, http://www.ht.sfc.keio.ac.jp/SSLab/

[7] Sun Microsystems Inc., JiniTM Network Technology, http://wwws.sun.com/software/jini/, 2000

[8] UPnP, Universal Plug and Play Forum, http://www.upnp.org, 2002

[9] Brouwer-Janse, M.D., Bennett R.W., Endo, T., vanes F.L., Strubbe, H.J., and Gentner, D.R., "Interfaces for consumer products: 'How to camouflage the computer?" Proc. CHI'1992, pp. 287-290, 1992

[10] Olsen, D. R., and Nielsen, T., "Laser Pointer Interaction," Proc. CHI '2001, pp. 17-22, 2001

[11] Ward, A., Jones, A., and Hopper, A., "A new location technique for the active office," IEEE Personal Communications Magazine, Vol. 4, No. 5, pp 42-47, 1997

[12] Priyantha, N.B., Miu, A.K.L., Balakrishnan, H., Teller, S, "The Cricket Compass for contextaware mobile applications," Proc. MOBICOM 2001, 2001

[13] Savvides, A., Han, CC, and Strivastava, M. B., "Dynamic fine grained localization in ad-hoc sensor networks," Proc. MOBICOM2001, 2001

[14] Pottie, G. J., and Kaiser, W. J., "Wireless integrated network sensors,"Comm. ACM, Vol. 43, pp. 51–58, 2000

[15] SCADDS, Scalable Coordination Architectures for Deeply Distributed Systems, http://www.isi.edu/scadds/

[16] WEBS, Wireless Embedded Systems, http://webs.cs.berkeley.edu/

[17] TinyOS, http://webs.cs.berkeley.edu/tos/

[18] Minami, M., Sugita, K., Morikawa, H., and Aoyama T., "A design of internet application platform for ubiquitous computing environment," IEICE Trans. Comm., vol. J85-B, no. 12, pp. 2313-2330, 2002

[19] Minami, M., Morikawa, H., and Aoyama T., "The design and evaluation of an interface-based naming system for supporting service synthesis in ubiquitous computing environment," IEICE Trans. Comm., vol.J86-B, no.5, pp.777-789. 2003