xtel: A Development Environment to Support Rapid Prototyping of “Ubiquitous Content”

Satoru Tokuhisa*, Takaaki Ishizawa, Yoshimasa Niwa***, Kenji Kasuya***, Atsuro Ueki*, Sho Hashimoto**, Kazuhiko Koriyama** and Masa Inakage******

* Keio University Advanced Research Center
** Keio University Graduate School of Media and Governance
*** Keio Research Institute at SFC
**** Keio University Graduate School of Media Design
S110, delta, Endo 5322, Fujisawa, Kanagawa, Japan
+81-466-49-3545

{dangkang, txi, niw, kasuya, atsurou, shokai, koriyama, inakage }@sfc.keio.ac.jp

ABSTRACT

This paper describes the "xtel" development environment for "Ubiquitous Content". Ubiquitous contents are real space applications that are embedded in day-to-day life and intended for use by consumers. This is content that is experienced through interaction with people, objects and environments that exist in real space. Xtel comprises three tools: the "moxa" MCU board that connects to sensors and actuators and is capable of short-distance wireless communications; the "Talktic" programming/runtime environment for the MCU board that contains a JavaScript parser, compiler, VM and library; and the "Entity Collaborator" P2P network library that is capable of handling continuous information such as video and audio in addition to the discrete information from sensors. Its use both accelerates development and makes development itself easier. As sample applications, this paper also contains an overview of three rapid prototypes developed for use in demonstrations at Maker Faire 2008.

Keywords

Prototyping, Ubiquitous computing, MCU, VM, P2P

INTRODUCTION

The basic technologies for ubiquitous computing are developing and advancing, and the nature of applications is changing as a result of this evolution. As wireless environments spread, users are increasingly able to connect to networks anywhere and at any time. The connection of sensor devices to networks and databases will allow a wide range of real world phenomena to be stored and shared as data. Within this environment, applications are no longer limited to conventional input devices like keyboards and mice or conventional output devices like displays and external speakers. They can incorporate functions that are on par with those in real space. In other words, it is possible to develop real space applications that sense phenomenon in the real world and, based on the results, actuate events in the real world.

Among these real space applications, we refer to applications that are embedded in day-to-day life and intended for use by consumers as "Ubiquitous Content" [1]. This is content that is experienced through interaction with people, objects and environments that exist in real space. While "Tangible Bits" [2] is based on the concept of "touching information," "Ubiquitous Content" is focusing on “entertaining experience” in the everyday life activities. People will be entertained for a very short moment such as 5 seconds when one is interacting with the household goods or environment. This snack sized entertaining experience in the daily life opens up a new landscape for smart artifacts and environment to become a ubiquitous media.

This paper provides an introduction to "xtel," a development support environment that enables the efficient creation of these ubiquitous contents. Xtel comprises the three tools described below, which are incorporated into an integrated development environment (IDE), such as Eclipse, to enable ubiquitous content development.

- Moxa: A "Micro Control Unit" (MCU) board that connects with sensors and actuators and is capable of short-distance wireless communications.
- Talktic: A programming/runtime environment for the MCU board that includes a JavaScript parser, compiler, virtual machine (VM) and library.
- Entity Collaborator: A P2P network library that uses the SIP protocol to be able to handle continuous information like video and audio in addition to discrete information from sensors.

The use of these tools enables both developers and designers to quickly and easily create ubiquitous contents. By accelerating the speed of development and lowering the hurdles to development, it will facilitate the participation in development of creative designers who have not been involved in the past due to the technical barriers. Giving engineers and designers the ability to develop entertainment-oriented ubiquitous content will enrich everyday life.

RELATED RESEARCHES

This chapter examines research related to the three xtel tools and highlights problem points.

Sensor Devices

Two of the best-known devices using wireless communications functions to facilitate the development of ubiquitous computing applications are "MOTE" [3] and "SmartIts" [4]. MOTE is a wireless sensor terminal capable of creating ad hoc multi-hop networks. It uses 2.4 GHz IEEE 802.15.4 as its wireless standard. It also contains an operating system known as "TinyOS," and for development uses "nesC," an extension of the C language. SmartIts is a sensor network device with a wide variety of sensing functions. It achieves wireless communications with either Bluetooth or RFM communication functions. One of the problems with these devices is that they require the sensor configuration and layout on the board to be fixed; if the layout required by developers differs from the layout of the sensors on the board, developers are forced to create the sensor environment from the hardware level up.

Conversely, "Phidget" [5] and "Teleo" [6] are leading examples of devices that can be easily connected to sensors and actuators by developers themselves but do not have wireless communications functions. Instead, they create implementation environments that can be directly controlled from a computer using USB connections. Libraries are also provided to enable development in a range of development environments, including Flash and Max/MSP. The execution environment, however, is the PC, which makes this technology difficult to utilize in applications that are required to be compact and in distributed applications that use short-distance wireless to share environmental information and operate collaboratively.

Development Environment for MCU Boards

"Wiring" [7] and "Arduino" [8] are among the leading integrated development environments that extend to MCU boards. Wiring is an integrated development environment that comprises an i/o board and an open source development environment (Java). It can be programmed in the Java-based Proce55ing [9] language. Arduino is an integrated development environment that comprises an i/o board and an open source development environment, the same as Wiring. It uses an original language that it derived from Wiring and has a C/C++-like syntax. In both cases, coordination with the MCU board is hidden by the development environment. Their merits are that they do not require that hardware or software be designed and implemented from scratch and they use simple programming languages that ultimately reduce the barriers to development. However, there are also some shortcomings, including the inability to coordinate multiple MCU boards and lack of object-oriented concepts.

If one focuses on the development environment itself, it is possible to use a Java virtual machine as the execution environment in order to achieve a truly object-oriented environment. Sun SPOT [10] is a wireless sensor network device that incorporates a Java runtime environment. As a Java-based programmable device, it can be programmed using integrated development environments like NetBeans. It also includes 2.4 GHz IEEE 802.15.4 wireless and a battery. NanoVM [11] is a VM for the Atmel AVR ATmega8 CPU rather than for a specific MCU board. It can be programmed in the Java language using the Sun JDK. It has also been successfully miniaturized, enabling Java programs to utilize 75% of the 1 kByte RAM.

A Framework for Ubiquitous Computing

There has been research into frameworks and middleware to support applications development based on ubiquitous computing concepts [12], but most of the existing research focuses on the distribution of sensor information, context information and other event information, which makes it difficult to use video or audio. To address these issues, research has examined the potential of using SIP [13] or XMPP [14]. SIP is a protocol used for IP telephone services, video conferencing and Instant Messenger (IM) services, and its application to information appliances and mobile telephones has also been studied. XMPP is a XML-based protocol that is used in, for example, Google Talk. It is suited for IM-based message exchange and also for notification of presence information, for example, login status. While XMPP is a standard currently applied only to IM, SIP can be effectively used as the backend for communications infrastructure.

Examples of research into the use of SIP as a framework include CINEMA [15] and SPLAT [16]. CINEMA is a software foundation for multimedia collaboration and comprises multiple SIP servers. The server cluster is made up of SIP proxies, presence servers and registrars, which are controlled by "Call Processing Language" (CPL). It is also possible to develop original applications using the C and C++ libraries provided. SPLAT is a platform to coordinate SIP-based multimedia conferencing, games and other similar services with existing applications. It is comprised of high-level APIs to provide the SIP functions, client-side SIP services to drive them and "network infrastructure blocks" that define original SIP-based services. Low-level APIs are also provided to enable the fine-tuning of individual SIP flows. While the APIs are useful for SIP applications development, client/server architecture is used in all cases, giving this approach little

of the scalability and ad hoc adaptability that would make it suitable for home networks or offices. In addition, the APIs provided are geared for engineers well-versed in SIP, making it difficult for others to learn them and prototype applications using them.

XTEL

This chapter examines the three tools that comprise xtel, outlining their features and structures. These three tools share the common purpose of making it faster and easier to build ubiquitous contents.

Moxa

Features

After reviewing the relevant literature, we adopted the following four features as requirements for the MCU board we developed. The first two are designed to accelerate the trial and error cycle for the developer. The second two are to realize the operations in different spaces that are required for the advancement of ubiquitous computing research.

1. **Simplification of sensor connection and use**

   Use of sensors requires no knowledge of hardware. It is possible to obtain sensor information just from attaching/detaching connectors and providing code descriptions.

2. **General-purpose orientation -- ability to connect to a wide range of sensors**

   Use of a common connector standard for a wide range of sensors to be mounted, thus enabling free configuration of sensors.

3. **Distributed environment**

   Equipment with short-distance wireless communications functions to enable communication of sensor information among devices, achieving a device mounting volume that is less than that of the PC.

4. **Extended range of spatial application**

   Achievement of communications between devices and mobile telephones and between devices and PCs.

Structure

Moxa developed by our project is a computer board that uses an ATMEL controller as its MCU (see Figures 1 and 2). Near the MCU, 32 Kbyte SRAM and an IC that incorporates the primary functions are mounted, including a wireless transceiver that conforms to the 2.4 GHz IEEE 802.15.4 standard (Zigbee lower layers). Wireless communications are capable of maximum speeds of 100 kbps at a clear sight distance of approximately 100 m, and it is possible to specify channels.[1] Proto01 has an interface that is useful for development and debugging, consisting of a JTAG interface, in circuit programming connector, RS232C connector and check terminal. It can be connected to a sensor expansion board that can connect standardized sensors. Proto02 is a more compact version that eliminates these interfaces and employs a USB interface.[2] Both devices can be driven by a 4-10 V battery.

Figure1. Proto01

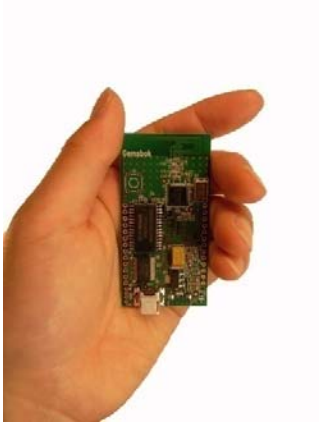

Figure2. Proto02

Application programs can be written in the C language in the development environment. Developers can easily make use of hardware functions with the software library, which consists of a device control library, wireless communications library, serial communications library and resource management library. The software library is statically linked to application programs written by users. Nonetheless, development in the C language represents significant hurdles for designers, so the unit is equipped with Talktic, which allows use of development languages based on ECMA Script, a familiar tool in web development.

[1] It is possible to mount Xbee [17] on Arduino to achieve wireless functions, but channels cannot be designated.

[2] Note that Proto02 does not provide Features 1 and 2.

Talktic

Features

After reviewing the relevant literature, we adopted the following four features when designing the platform for MCU board control.

1. **Virtual machine-based execution environment**

   The adoption of a runtime environment that is driven by virtual machine-based bytecode eliminates the dependence on a specific MCU, which was one of the problems in using MCU boards in the past.

2. **Lightweight language-based programming environment**

   The system utilizes a lightweight language that enables object-oriented coding and reduces coding load. Rather than employ an original language, we adopted a language that is already in widespread use.

3. **Concealment of element technologies by library**

   Libraries are furnished to abstract and conceal communications with other devices and the handling of sensors and actuators.

4. **Clearly identified execution constraints**

   Implementation constraints are clearly documented. It is possible to implement specific classes or methods as part of the programming environment.

Structure

The Talktic platform consists of the following implementation components (Figure 3). Combinations of implementation components achieve Talktic platform functions.

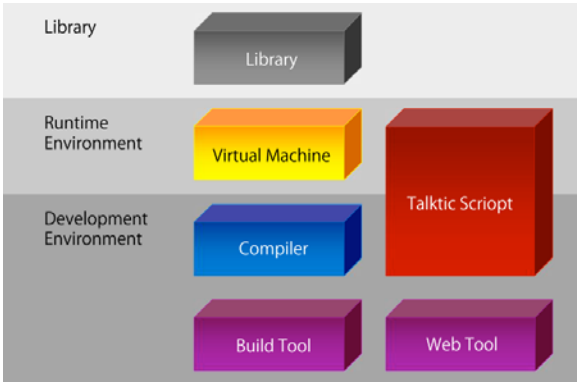

Figure3. Structure of the Talktic platform

The Talktic virtual machine is based on the NJS JavaScript Interpreter. This program furnishes a JavaScript compiler, assembler, virtual machine, native methods and classes under an LGPL license [18]. The Talktic virtual machine is based on NJS JavaScript Interpreter 0.2.5, which has been modified and revised so that it is suited for an MCU runtime environment. It is written entirely in the C language. This Talktic virtual machine provides single-thread operation, uses garbage collection for dynamic memory management, and supports floating-point operations and virtual interrupts.

At the core of the Talktic platform is Talktic Script. Talktic Script conforms almost fully to ECMA Script Version 3, which is described in the international standard "Standard ECMA-262 3rd Edition" [19]. Similar languages include ActionScript and JavaScript. General features of Talktic Script as an ECMA script include prototype-based object orientation, anonymous functions, anonymous objects and standard control syntax. However, the current language specification for Talktic Script does not implement scope chain, closure or native class prototype modification. This is all primarily dependent on the implementation of the base NJS JavaScript Interpreter.

There are two aspects of Talktic libraries. The first is that they serve as the drivers for the basic input/output and communications functions provided by each MCU board and provide native API libraries for manipulation through Talktic Script. More specifically, they contain digital IO (pinMode, digitalWrite, digitalRead), analog IO (analogWrite, analogRead, soundWrite), wireless communications (radioconnect, radioSend, onRadioReceive, radioClose), serial communications (serialInit, serialAvailable, serialRead, onSerialReceive, serialSend) and other similar services. Second, there is an intermediate library to enable implementation of Talktic Script's object-oriented design patterns. The native APIs are simple and can be used without modification, or they can be wrapped as commonly-used design patterns for smoother implementation.

Entity Collaborator

Features

After reviewing the relevant literature, we adopted the following two features in the design of the P2P framework.

1. **Integrated use of discrete and continuous information**

   "Discrete information" refers to events and context generated from sensor information as well as text-based information like IM; "continuous information," to IP-based audio telephony, video chat videos and other streaming information. SIP is used to provide integrated handling of this information. Learning costs are reduced by using event-driven APIs, concealing session management and abstracting methods.

2. **Ad hoc adaptation and scalability**

   These features are achieved with the use of distributed hash tables (DHT) in node management. DHT is a technology to create hash tables within a distributed environment, thereby providing information distribution and data search functions. Learning is easy because it only involves management of hash tables. Among DHT's features is the fact that it is pure P2P that does not depend on servers, it has a high degree of scalability and ad hoc adaptability, and it

avoids single points of failure, thus being able to discover objects on the network with a high degree of probability.

Structure

EntityCollaborator(EC) is a framework with the two features described in the preceding subsection. Users need merely to use the libraries provided to develop SIP applications capable of integrated handling of discrete and continuous information. For implementation, Java (JDK 5.0) is used.

The EC system is comprised of five modules: 1) "Entity" that is a component created by users; 2) "EntityContainer" that is a container for Entity; 3) the "SipCore" that is a processing module for SIP messages; 4) the "Chord" that is the module providing DHT functions; and 5) "EntityCollaborator," which is the system front end. These modules are explained in more detail below.

The "entity" is the central element in application development and is defined as an interface. Users can develop an entity with arbitrary functions by inheriting from AbstractEntity, which is an abstract class, and entities can be combined to create an application. The entity conceals SIP communications and defines abstracted methods to enable event-driven programming.

Entities delegate processing to SipCore to transmit SIP messages and receive SIP messages from the outside, calling event handlers corresponding to entities designated in the SIP URI. Because of this, their functions are similar to that of an SIP proxy.

As a DHT-based search method, we developed an approach whereby arbitrary keywords are attached to an entity and XML containing meta-information on entities corresponding to keywords can be used as DHT entries. The XML used to describe meta-information is "Entity Information Markup Language" (EIML), which describes the entity's SIP-URI and parameters and is automatically generated by the system. Chord is a class that implements these algorithms. Listing 1 shows sample code.

EntityCollaborator is a class that functions as the system front end and contains methods for system startup and entity search. System startup basically consists of initialization of SipCore, initialization of Chord, designation of bootstrap node, participation in DHT network from automated discovery and initialization and addition of entities, in that order. Listing 2 shows the system startup code.

```
public class CameraEntity extends AbstractEntity {
    public CameraEntity() {
        //Addition of keywords, initialization of Web camera (omitted)
        addKeyword("camera");
        init();
    }
    public void receiveMessage(EntityEvent e) {
        // Console output of received messages
        System.out.println(e.getMessage());
    }
    public SessionDescription receiveOffer(EntityEvent e) {
        //Obtain SDP from offer source,
        //and generate answer SDP (omitted)
        SessionDescription sdp =e.getSessionDescription();
        SessionDescription ret = getResponseSDP(sdp);
        //Start streaming (omitted)
        startStreaming(sdp);
        return ret;
    }
}
```

List 1. Example of keyword addition

```
public class Main {
    public static void main(String[] args) throws Exception {
        EntityCollaborator ec = EntityCollaborator.getInstance();
        // Initialization of SIP stuck
        ec.initiateSipCore();
        // Initialization of Chord and automated discovery of
        // peer
        ec.findPeer();
        // Addition of entities
        ec.addEntity(new CameraEntity());
        ec.addEntity(new SearchEntity());
    }
}
```

List 2. Startup of EntityCollaborator

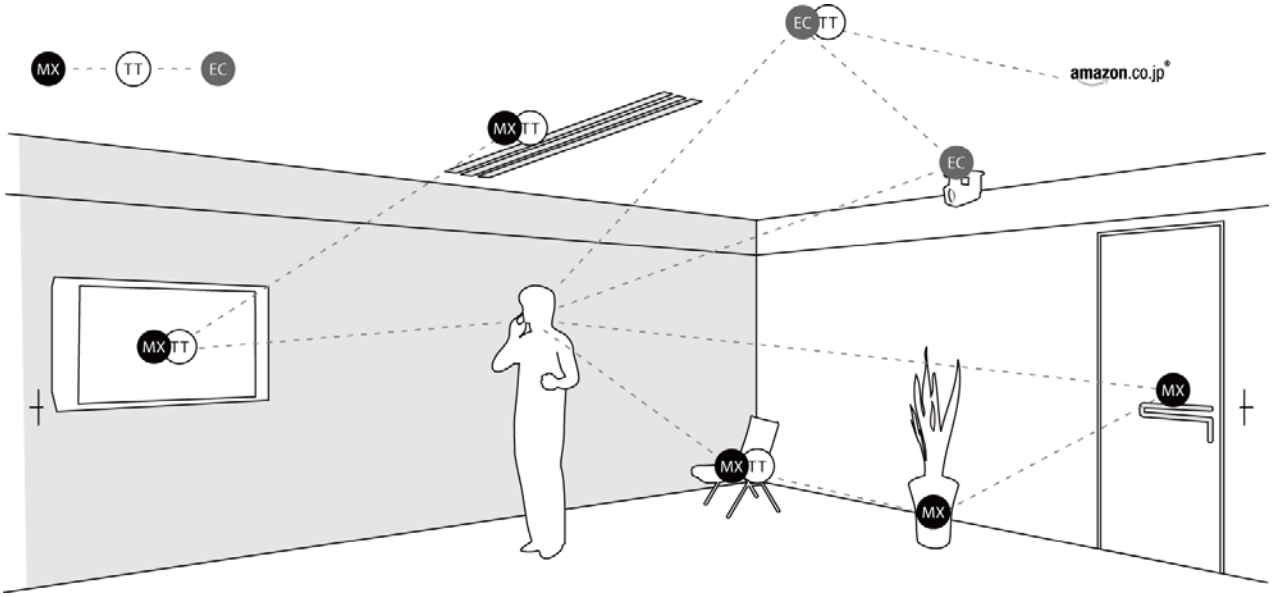

Figure3. Image of xtel-based ubiquitous contents

Cordination

Xtel is an integrated development environment comprising these three tools that creates new, personal-level ubiquitous content by making it easy to combine information of virtually any type from virtually any source, including everything from ordinary home appliances and digital equipment to the Internet itself (Figure 4). For example, it is possible to build an application in which the act of approaching a bookshelf is shared over the network and linked with a Web service like Amazon to enable the user to purchase a book directly. It is also possible to quickly achieve and test a series of actions based on that information and reflect it in behavior in the physical world, for example, turning on a reading lamp.

SAMPLE XTEL-BASED UBIQUITOUS CONTENTS

We used xtel to build several real space applications as demonstrations for Maker Faire 2008 Bay Area[20]. In this paper, we provide an overview of three ubiquitous contents implementations: "Fluttering," that extends a lamp; "Lovely Wife," that extends a photo frame; and "Shoulder Massager," that extends a massager.

Fluttering is a lamp that changes its brightness depending upon the value from a 3-axis acceleration sensor embedded in a wireless controller. The changes in brightness mimic the flickering of candles. This application uses two moxa boards. One of the boards embeds 3-axis acceleration, using wireless to transmit acceleration on the X, Y and Z axes. The other board receives these values and controls three high-luminescence LEDs (14 lumens/100 mA).

Lovely Wife is a photo frame that plays the voice of a wife or significant other when the user approaches. The envisioned users of the application are people involved in long-distance romances or stationed in remote locations without their families. For example, a man returns home from work. As he approaches a photo frame containing a photograph of his wife or girlfriend, he hears her voice, which helps him to dispel the fatigue of the day. This application uses one moxa board and a PC. The photo frame has a built-in LED and proximity sensor. Moxa processes the user proximity information from the sensor and sends a trigger to the PC when the user is within 5 cm. The PC uses a Flash application to play back a random sound based on the trigger information. This specification uses pre-recorded sound information, but it would also be possible to use Entity Collaborator and SIP to read audio data from the wife or girlfriend that exists in a remote location.

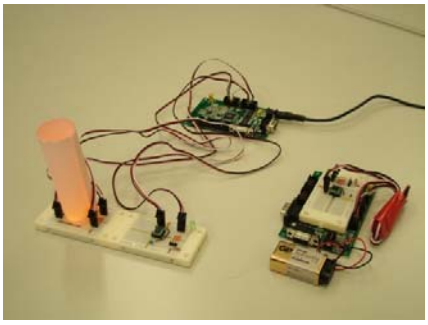

Figure4. Wireless controller (right) and lamp (left)

Figure5. Lovely wife

Shoulder Massager is a system that allows a remote user to control the operations of a low-frequency massager attached to another user. This application uses two moxa boards. User A wears a glove with a built-in pressure sensor that is connected to the moxa. He massages his own shoulder etc. At this time, the value from the pressure sensor is sent to the other moxa via wireless. A low-frequency pad attached to User B is controlled on the basis of information received from the moxa.

Figure6. Glove with embedded sensor (right) and low-frequency pad (left)

The three ubiquitous contents described above are all rapid prototypes. Designers familiar with the ActionScript development environment built each of the applications in about one hour based on ideas from a brainstorming session. All of them are written in about 50 lines of JavaScript. The key point at the rapid prototyping stage is how quickly an application can be built. The first step in implementation is to build a stage at which interaction with the user is possible, so as to verify the need to add or remove functions depending upon the experience to be provided. The three xtel tools were designed to accelerate the development cycle and provide extremely powerful tools for rapid prototyping.

EVALUATION

We held a workshop in Japan for some students and researchers in order to familiarize them with. In this workshop, we introduced it about moxa and Talktic with some sample codes to operate moxa itself, to communicate between moxa and a PC, and to communicate between moxa and moxa. We also explained Entity Collaborator with sample codes on P2P network about how to send a message, how to add a node, and how to use a camera. At the end of the workshop, we asked the 26 participants to answer some questionnaires. Table 1 shows positive and negative opinions from 19 participants who have experience to developing applications and table 2 shows opinions from 7 participants who have no experience.

Table1. Some opinions from expert group

positive opinions	negative opinions
Cooperation with the web is fun! It's very easy to use wireless communication. This is more user-friendly than other existing technologies. It's not too complicated for beginners. Designer can reflect his/her thinking because there is no bar that we have to overcome.	It is necessary to cooperate with Flash or a similar program in the case of developing graphical expression. Including this, we will do a lot of things on xtel. I think that the users who are not familiar with programming cannot use it, because it is difficult to develop visual expression such as Processing. Some libraries for Flash are happy.

Table2. Some opinions from non expert group

positive opinions	negative opinions
It is easy to understand the operation of wireless communication. Easiness of programming makes me happy. These systems for beginners are easy to adopt.	I hope more samples would be prepared.

These opinions indicates us that xtel got a certain amount of evaluation about the user-friendliness of the development environment. However, we think to strengthen cooperation functions with Flash and Processing in order to realize graphical expression easily. In addition, we got a lot of opinions that it would be difficult for most of the non-experts who have no experience to write programs in JAVA or to develop network applications, to use Entity Collaborator. We will solve this problem by offering some classes which can run on Flash or Processing.

CONCLUSION

This paper describes the "xtel" development environment for ubiquitous contents. Xtel's objective is both to accelerate the development of these contents and to lower the hurdles to development. Xtel comprises three tools with the features required to achieve their individual purposes. Moxa provides general utility, simplifying sensor connections and allowing connection to a wide range of sensors. Wireless communications functions enable the

specification to achieve wide-area distributed environments. Talktic adopts ECMAScript, which is very familiar to designers. Entity Collaborator uses a P2P network to handle both discrete information and continuous information, significantly reducing learning costs.

We will continue to improve these three tools in the future furthering our efforts to simplify the development process including cooperation with graphics function. First, the current development environment requires the use of Eclipse for development in either JavaScript or Java. However, some strongly feel that Java-based development still presents high hurdles to designers. In the future, we look forward to providing function-specific classes that can be used in Flash, Processing, Max/MSP and others, concealing the programming portion from designers in order to attract more users. These classes enable the users to cooperate with the graphics functions under these development environments. In addition, we will release each source code of moxa, talktic and Entity Collaborator under LGPL licenses on our website [21].

ACKNOWLEDGEMENT

This Project is granted by CREST, JST.

REFERENCES

1. Inakage, Masa, et al. "Designing Ubiquitous Content for Daily Lifestyle." *International Journal of Cognitive Informatics and Natural Intelligence (IJCINI)* (in print) (2008).
2. Ishii, Hiroshi, and Brygg Ullmer. "Tangible Bits: Towards Seamless Interfaces between People, Bits and Atoms." *Conference on Human Factors in Computing Systems* Atlanta, Georgia, 1997. 234 - 41.
3. Polastre, Joseph, et al. "The Mote Revolution: Low Power Wireless Sensor Network Devices." *Proceedings of Hot Chips 16: A Symposium on High Performance Chips.*, 2004.
4. Beigl, Michael, and Hans Gellersen. "Smart-Its: An Embedded Platform for Smart Objects." *Smart Objects Conference (SOC 2003)*. Grenoble, France, 2003.
5. Greenberg, Saul, and Chester Fitchett. "Phidgets: Easy Development of Physical Interfaces through Physical Widgets." *Symposium on User Interface Software and Technology(UIST)*. Orlando, Florida, 2001. 209 - 18.
6. Making Things. "Teleo". 2000. 30 June 2008. <http://www.makingthings.com/teleo/>.
7. Barragán, Hernando. "Wiring". 2006. 30 June 2008. <http://barraganstudio.com/>.
8. Arduino. "Arduino". 2005. 30 June 2008. <http://www.arduino.cc/>.
9. Reas, Casey, and Benjamin Fry. "Processing". 2001. 30 June 2008. <http://processing.org/>.
10. Simon, Doug, et al. "Java™ on the Bare Metal of Wireless Sensor Devices: The Squawk Java Virtual Machine." *ACM/Usenix International Conference On Virtual Execution Environments*. Ottawa, Ontario, Canada, 2006. 78 - 88.
11. Harbaum, Till. "Nanovm". 2005. 30 June 2008. <http://www.harbaum.org/till/nanovm/index.shtml>.
12. Endres, Christoph, Andreas Butz, and Asa MacWilliams. "A Survey of Software Infrastructures and Frameworks for Ubiquitous Computing." *Mobile Information System* 1.1 (2005): 41 - 80.
13. J. Rosenberg, et al. "Sip: Session Initiation Protocol". 2002. <http://www.ietf.org/rfc/rfc3261.txt>.
14. Jabber Software Foundation. Extensible Messaging and Presence Protocol (Xmpp): Core, 2004.
15. Berger, Stefan, et al. "Ubiquitous Computing Using Sip." *International Workshop on Network and Operating System Support for Digital Audio and Video*. Monterey, CA, USA, 2003. 82 - 89.
16. Singh, Aameek, et al. "Splat: A Unified Sip Services Platform for Voip Applications: Research Articles." *International Journal of Communication Systems* 19.4 (2006): 425 - 44.
17. MaxStream. "Xbee/Xbee-Pro". 30 June 2008. <ww.digi.com/products/wireless/zigbee-mesh/>.
18. Software, New Generation. "Njs Javascript Interpreter." 1998. 30 June 2008. <http://www.njs-javascript.org/>.
19. International, ECMA. "Standard Ecma-262, 3rd Edition." 1999. 30 June 2008. <http://www.ecma-international.org/publications/standards/Ecma-262.htm>.
20. Make. "Maker Faire Bay Area 2008". 2008. 22 October 2008. <http://makerfaire.com/bayarea/2008/>.
21. Keio CREST. " Xtel: Ubiquitous Content Platform ". 2008. 24 October 2008. < http://xtel.sfc.keio.ac.jp/en/ >.