The Furniture of Ubiquitous Computing

Sho Hashimoto
Keio University
5322 Endo
Fujisawa, Kanagawa 252-8520
Japan
hashimoto@shokai.org

Toshiyuki Masui
Keio University
5322 Endo
Fujisawa, Kanagawa 252-8520
Japan
masui@pitecan.com

UbiComp'13 Adjunct, September 8–12, 2013, Zurich, Switzerland.

Abstract
Although various ubicomp technologies have been proposed for home environments, few people are enjoying such technologies in their daily life, due to the lack of powerful software framework for building flexible applications for home. We are developing simple and powerful ubicomp frameworks which can be used for building furniture-embedded networked devices which fit to home environments. Using our frameworks, many devices can communicate with each other by exchanging data shared on the Web server using standard HTTP. In this paper, we describe the concepts and implementations of the frameworks, and show how sparsely-connected devices can cooperatively be used for various tasks needed in home environments.

Author Keywords
Ubiquitous Computing, Real-World GUI, GoldFish, Linda

ACM Classification Keywords
H.5.m [Information interfaces and presentation (e.g., HCI)]: Miscellaneous.

General Terms
Human Factors

Introduction

Although many interesting research projects have been going on in the ubicomp research community, many of them are not suited for home environment where special sensing devices do not fit. Having visible desktop computers and sensors in the home environment is not favored by people, because most of such devices are either ugly or alien to living room atmosphere. Some sensors can make people nervous because people know that cameras and sensing devices can easily cause privacy problems.

A good solution to this problem is to embed computers and sensors in everyday objects like furniture and lighting equipments so that people cannot notice the existence of special devices and computers. Many researchers have done experiments to use home furniture like wardrobes[8], drawers[4], photo frames[3], trash boxes[9], curtains[2], etc. Computing on tables and desks have long been a hot research topic and many systems like DigitalDesk[12], metaDESK[10], etc. have been researched. Using computing resources in the kitchen has also been an active research topic[3], and varioius augmented kitchen appliances like microwaves[11] have been proposed. Various intelligent light bulbs[1][2][3][4] have been proposed as commercial products. Researchers have been trying to build a house like TRON House[5], AwareHome[3], OchaHouse[6], and KsiHome[7] for the experiment of ubiquitous computing in the real home environment.

[1] http://www.kickstarter.com/projects/limemouse/lifx-the-light-bulb-reinvented

[2] https://www.facebook.com/lifxlabs

[3] http://www.insteon.net/bulb.html

[4] http://www.engadget.com/2012/08/19/bluetooth-bulb-lets-you-switch-on-time-dim-and-color/

[5] http://monotsukuri.net/mirai98/tron/tron.htm

[6] http://ochahouse.com/

[7] http://www.kyoto-su.ac.jp/liaison/kenkyu/message56.html

Although furniture-based ubicomp systems are the right approach, most of the existing research systems are just experimental and few systems are continuously used even by the authors, because of the following problems.

1. Aesthetics problem Most of the experimental systems do not have good appearances and they are not as beautifully designed as other furniture. If the devices look ugly, family members would refuse to continue using them at home.

2. Setup problem Many of the research systems require special hardware/software settings before usage. If a user has to register network parameters for all the devices before he can use the system, he would not set the parameters again when the network configuration is changed.

3. Programming problem Parallel and distributed programming under unreliable communication channel is required for handling many devices at home, but such programming is difficult without a flexible and reliable parallel programming model.

4. Customization problem When a system is made up of tightly-coupled modules, changing behaviors and adding features is not easy. Most of the experimental systems are not designed to be flexible enough, and the whole system should be redesigned when additional devices are required or the configuration should be modified.

Problem 1 can be solved if devices are hidden in existing furniture, and problems 2, 3, 4 can be solved if we use a powerful and flexible software architecture for the furniture-based systems.

Frameworks for Home Environment

We have set up the following goals for the design of our frameworks for furniture-based ubicomp systems:

- It should run on various devices, from one-board computers to desktop PCs.
- Devices can be placed at any location.
- Programs should be written in multiple languages.
- The programming model should be simple and flexible.

For these goals, we decided to use a Web server and HTTP for all the communications between devices. Since HTTP is an well-established protocol and supported by many commercial products, we believe using HTTP is better than using special protocols between devices. Communication between a Web server and a browser had to be initiated by the browser in the past, but using modern browsers and Web servers which support WebSocket[8] and Comet[9], more symmetric communication between them are possible.

We introduce two frameworks, *GoldFish* and *LindaBase*, for furniture-based ubicomp systems in the following subsections. With our frameworks, we can use PCs, iPhones/iPads, Android devices, and one-board computers like Arduino[10] and Raspberry Pi[11] for constructing the home ubicomp environment.

[8] http://en.wikipedia.org/wiki/WebSocket
[9] http://en.wikipedia.org/wiki/Comet_(programming)
[10] http://en.wikipedia.org/wiki/Arduino
[11] http://en.wikipedia.org/wiki/Raspberry_pi

GoldFish

GoldFish[5] is a JavaScript framework for developing "*Real-World GUI* (RWGUI)"[6] using an Android phone equipped with an NFC reader. Using the NFC reader and motion sensors at the same time, users can control various parameters by moving the phone on NFC tags, just like using a mouse for controlling menus and scroll bars on a PC. All the control programs are written in JavaScript and put on the Web so that users do not have to install different Android programs for different tags.

GoldFish is based on the same idea as the FieldMouse system[7]. We put NFC tags where we want to control data by GUI actions. If a user wants to control the sound volume of a TV from the dining table, he puts an NFC tag on the table, and when he puts his Android phone close to the tag, a special application on Android is automatically launched and recognizes his subsequent actions to control the TV; e.g. the application can detect the rotation of the phone and wirelessly tell the TV to modify the volume based on the angle.

We have created the GoldFish framework to enable such GUI operations just by writing a simple JavaScript code on the Web. When an Android phone recognizes an NFC tag, the GoldFish application on Android is launched and tries to display an Web page http://ubif.org/*(NFC-ID)* on the browser. The user can then use arbitrary JavaScript programs to read the sensor data of Android and perform GUI actions based on the data.

Figure 1 shows how we can copy/paste data between a PC and an Android phone. An NFC tag is put on the edge of the PC screen, and when a user puts his Android phone on the tag, a goldfish icon appears on the Android screen. If he twists the phone clockwise, the data on the

PC is copied to the phone, and the data on the phone is pasted to the PC if he twists the phone counterclockwise.

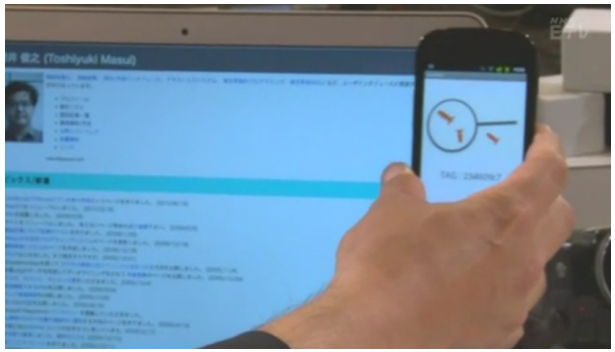

Figure 1: Real-world copy/paste.

Figure 2 shows a RWGUI program which uses the value given from the `goldfish` object.

```
var angle = 0;
setInterval(function (){
    var gyro = goldfish.gyroscope();
    angle += gyro.z * 3;
    // do actions based on the angle value
  }, 50);
```

Figure 2: A JavaScript code for detecting rotation.

LindaBase
For simple and powerful communication between devices at home, we developed a framework called *LindaBase*, which is an Web-based implementation of Linda[1], a "coordination language" for parallel programming. Linda is made up of several primitives operating on data objects called "tuples" placed in the "tuple space" shared by multiple processes. A tuple is represented as an array of objects and handled from programming languages like C, JavaScript, Ruby, etc.

The original Linda model requires four operations for handling tuples and the tuple space:

- `in`: atomically reads and removes a tuple from tuplespace
- `rd`: non-destructively reads a tuplespace
- `out`: produces a tuple, writing it into tuplespace
- `eval`: creates new processes to evaluate tuples, writing the result into tuplespace

Based on this model, LindaBase is implemented on an nginx[12] Web server, and the tuple space is implemented on Sinatra[13]. Various programs and devices can connect to the LindaBase server via HTTP and communicate with each other by using tuples using Linda operations.

Implementation of LindaBase
Requests from the client to the Web server is initiated by standard HTTP GET/POST, and request from the server to the client is performed by Comet or WebSocket based on whether HTML5 is supported. LindaBase is implemented using the RocketIO[14], a Ruby library which resembles Socket.IO[15], for realtime communication between the Web server and clients.

Using the LindaBase server, all the Linda operations like `in`, `out` can be performed only using HTTP, and wide

[12] http://en.wikipedia.org/wiki/Nginx
[13] http://en.wikipedia.org/wiki/Sinatra_(software)
[14] http://shokai.org/blog/archives/7180
[15] http://en.wikipedia.org/wiki/Socket.IO

variety of devices can freely join or leave the network. Tuples are represented in the JSON[16] format so that various systems can handle tuples easily.

For example, an Arduino in the living room can sense the status of the light bulb of the room, and the data is sent to a PC and then put to the tuple space using the following Ruby program:

```
ts = Linda::Client.new.tuplespace["myhouse"]
loop do
  light = arduino.analog_read 0
  ts.out ["sensor", "light", light]
  sleep 1
end
```

A JavaScript program running on a browser on a different machine can read the tuple and see whether the light of the room is on or off.

```
var ts = new Linda().TupleSpace("myhouse");
ts.rd(["sensor", "light"],
   function(tuple){  alert(tuple);  });
```

A separate program running on another PC can continuously monitor the same data and check if the light is turned on or off, and tweet the information on Twitter.

Here, each program does not have to know what other programs are doing; each program only communicates with the tuple space and takes care of the tuples of their interest. With such loose connection between processes sharing the same tuple space, simple but powerful cooperation becomes possible. Processes can be added, removed, and modified without affecting other processes.

[16] http://en.wikipedia.org/wiki/JavaScript_Object_Notation

Examples

We can use the combination of GoldFish and LindaBase for controlling various objects in home environment.

Opening a door

We can open the door with an Android phone using GoldFish and LindaBase. When a user touches the NFC tag on the wall with an Android phone equipped with GoldFish, an image of a thumb-turn lock appears, and the user can rotate the phone to unlock it. (Figure 3)

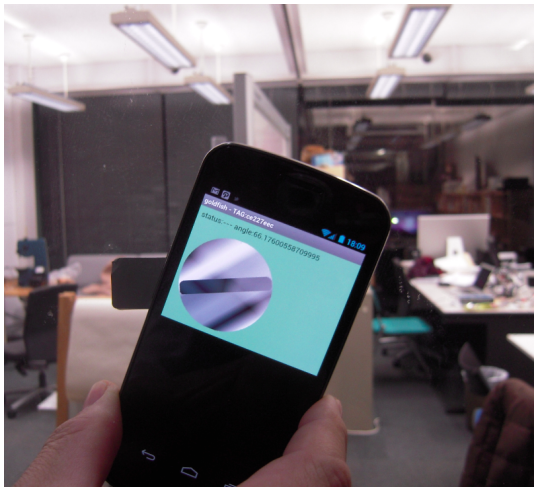

Figure 3: Using an Android phone for unlocking the door.

A tuple "`["door", "open"]`" is put into the tuple space from the phone when the user rotates the Android phone. When a PC at the door detects the tuple, it controls the Phidgets[17] servo motor to rotate the thumb-turn lock. (Figure 4)

[17] http://www.phidgets.com/

Figure 5: Coffee bag with a coffee clip.

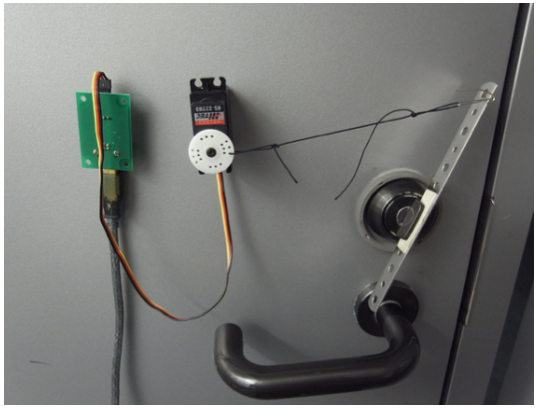

Figure 4: Unlocking mechanism.

If we want to use a different strategy for opening the door, we can just use a new device which puts the same tuple into the tuple space, and no modification to existing systems is necessary.

Home inventory management
At the supermarket, we often want to know if we have coffee in stock at home. It might be possible to install a special coffee sensor in the kitchen to report the status on the Web, but having such sensors for everything is not practical. Better approach would be using a "coffee clip" shown in Figure 5 with an RFID tag, where unused clip means that coffee is out of stock.

When the clip is put on a tray with an RFID reader, it puts a tuple "`["coffeestock", false]`" in the tuple space so that a user can check it at the supermarket using an application which accesses the tuple space on the Web.

Getting data from the Web
We can use existing Web services to get data in the real world and share the data in the tuple space. Figure 6 shows a portion of the Web page of Enoshima marina[18], where current wind speed of the shore is displayed. We are running a daemon program which periodically "scrapes" the HTML data and puts the wind data into the tuple space, so that a pinwheel system running on Arduino can use the data to physically display current wind speed at home.

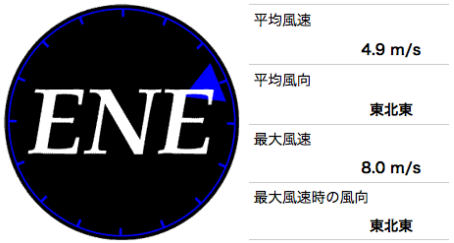

Figure 6: Wind strength at Enoshima marina.

Conclusion

We have introduced two frameworks for quickly and easily building furniture-based ubicomp systems for the home. Using our frameworks, we can flexibly integrate PCs, mobile phones, and off-the-shelf I/O devices for building useful home applications. Based on our frameworks, we are building a wide range of furniture-based ubicomp environment which is really usable in ordinary households.

References

[1] Carriero, N., and Gelernter, D. Linda in context. *Communications of the ACM 32*, 4 (Apr. 1989), 444–458.

[2] Handa, T., Kambara, K., Tsukada, K., and Siio, I. SmoothCurtain: privacy controlling video communication device. In *Supplemental Proceedings*

[18] http://enoshima-yacht-harbor.jp/index4.htm

of the 11th Ubicomp 2009 (2009), 186–187.

[3] Kientz, J. A., Patel, S. N., Jones, B., Price, E., Mynatt, E. D., and Abowd, G. D. The Georgia Tech Aware Home. In *CHI '08 Extended Abstracts on Human Factors in Computing Systems* (2008), 3675–3680.

[4] Komatsuzaki, M., Tsukada, K., and Siio, I. Drawerfinder: finding items in storage boxes using pictures and visual markers. In *Proceedings of the 16th international conference on Intelligent user interfaces*, IUI '11 (2011), 363–366.

[5] Masui, T., and Hashimoto, S. GoldFish: real-world GUI framework for Android. In *SIGGRAPH Asia 2012 Symposium on Apps*, SA '12 (2012), 3:1–3:1.

[6] Masui, T., and Siio, I. Real-world graphical user interfaces. In *Proceedings of the 2nd international symposium on Handheld and Ubiquitous Computing*, HUC '00, Springer-Verlag (London, UK, UK, 2000), 72–84.

[7] Siio, I., Masui, T., and Fukuchi, K. Real-world interaction using the fieldmouse. In *Proceedings of the 12th annual ACM symposium on User interface software and technology*, UIST '99 (1999), 113–119.

[8] Siio, I., Rowan, J., and Mynatt, E. Finding objects in "strata drawer". In *CHI '03 Extended Abstracts on Human Factors in Computing Systems* (2003), 982–983.

[9] Tsujita, H., Tsukada, K., and Siio, I. SyncDecor: communication appliances for virtual cohabitation. In *Proceedings of the working conference on Advanced visual interfaces*, AVI '08 (2008), 449–453.

[10] Ullmer, B., and Ishii, H. The metaDESK: models and prototypes for tangible user interfaces. In *Proceedings of the 10th annual ACM symposium on User interface software and technology*, UIST '97, ACM (New York, NY, USA, 1997), 223–232.

[11] Watanabe, K., Matsuda, S., Yasumura, M., Inami, M., and Igarashi, T. Castoven: a microwave oven with just-in-time video clips. In *Proceedings of the 12th ACM international conference adjunct papers on Ubiquitous computing - Adjunct*, Ubicomp '10 Adjunct (2010), 385–386.

[12] Wellner, P. Interacting with paper on the DigitalDesk. *Communications of the ACM 36*, 7 (July 1993), 87–96.