A Scheme of Service Discovery and Control on Ubiquitous Devices

Mitsutaka WATANABE
Mitsutaka.Watanabe@unisys.co.jp

Ken-ichi TAKAYA
Kenichi.TakayaKE04@unisys.co.jp

Akishi SEO
Akishi.Seo@unisys.co.jp

Nihon Unisys, Ltd.
1-1-1 Toyosu, Koto-Ku, Tokyo, Japan
+81-3-5546-5721

Masatomo HASHIMOTO
masatomo@carc.aist.go.jp

Tomonori IZUMIDA
tizmd@carc.aist.go.jp

Akira MORI
amori@carc.aist.go.jp

National Institute of Advanced Industrial Science and Technology (AIST)
2-41-6 Aomi, Koto-ku, Tokyo, Japan
+81-3-5546-5721

ABSTRACT
In this paper, we describe the discovery of service and controlling them in ubiquitous devices.

Categories and Subject Descriptors
H.5.2 [**User Interface**]:

General Terms
Design

Keywords
Ubiquitous computing, Service discovery, Ad-hoc network, Peer to Peer

1. INTRODUCTION
The ubiquitous computing has long been in vogue. When we can build-in intelligence into familiar devices nearby us, user-interface will evolve into a natural one to us so that devices could automatically execute what we expect them to do, or they could be controlled by voice, gesture and so on. To realize these ubiquitous computing environments, we must establish schemes that can control network-connected computers embedded in everyday-use devices.

We have developed a set of hardware and software components to realize such ubiquitous computing environments, based on two keywords, "simple" (easy to implement) and "open"(adopt widely publicized specifications). And this set has been resulted into UBKit (Ubiquity Building Toolkit)[1]. An ultra small (Compact Flush card-sized) and power-saving MPU module [2], one of the UBKit modules, can run on Linux and work as a micro-server with a wireless communication module. By attaching the micro-server to the existing consumer electronics appliances which have no network controller, these appliances can be connected with network and be accessed from outside. (Plug and Play).

WWW 2004, May 17–22, 2004, New York, New York, USA.
ACM 1-58113-912-8/04/0005.

In this paper we propose a scheme that can connect these micro-server enabled appliances to network without complex setup and easily discover and control them.

2. ZERO-CONFIGURED NETWORK PARTICIPATION
Every device should be able to connect to each other in Ad-Hoc wireless network anytime anywhere. As Ad-Hoc networks generally do not always contain servers that are essential in IP-networks like DHCP or DNS, a scheme is needed that each device could communicate to others without servers. In IP-address assignment and service publish and discovery, UBKit adopts Apple's Rendezvous[3] scheme. Rendezvous consists of the following three open specifications, Zeroconf[4] assigning IPv4 address by itself, Multicast DNS[5] enabling name resolution without DNS server, and DNS-SD[6][7] which can publish and search services provided by hosts. However, we can find only an IP-address and a port number of the specific device in terms of service name and protocol name in this scheme. Also types of services are limited to the range of Well-Known Port Numbers described in IANA [8]. Furthermore, users of a particular service must know the specification of the service beforehand.

We have defined an access interface to devices offering functions (service) in UBKit and we have devised an scheme in which devices can proactively discover services and describe contents of services by extending DNS-SD service description specification.

3. ACCESS INTERFACE TO DEVICES
When we map service requests to the actual operations of devices, such as "pushing buttons" or "rotating a knob", these operations are independent of others and do not always require the "session" idea. In other words, a device requires just a kind of RPC-like interfaces that return results of behavior when the device receives service request from other devices. Many RPC-protocol specifications exist today, such as CORBA or SOAP but these specifications are so complex for seeking high functionality that it is difficult to implement on the memory size and processing-power limited micro-server. Therefore, UBKit has adopted HTTP and/or CGI requests as service request interface and returns a single kind of data with MIME type. In other words, UBKit assigns services (functions) of a device to a unique URL and

control the device by requesting the URL. In this environment, a device requesting services are only required to provide a HTTP-client function (such as Web browsing).

Every device in UBKit also provides a graphical operation interface using Web pages when human directly accesses it. As the actual execution of services is implemented as CGI, Web pages are written as a simple front-end interface. For example, when a TV device is accessed directly via a Web browser, it returns a GUI interface which enable TV operations, looking like a kind of TV remote console.

4. EXTENSION OF SERVICE DESCRIPTION

DNS-SD describes services in DNS records and resolves service-instance name by referring IANA-defined protocol names in PTR record. Also real host name, a port number and additional information can be obtained from service-instance name by referring SRV and TXT records.

As devices in UBKit are controlled by HTTP requests and/or CGI, the service type is "_http._tcp". But in this notation all devices have the same information and no information is available about each device's services. In UBKit-supported devices application name is added as a sub-type so as to allow more specific definition on a service-type (see Table 1). The sub-types consist of "_ubkit" sub-type and also an application-uniquely defined sub-type. In TXT record field "_a" containing the sub-type must be mandatory, so that network-connected devices can select the devices containing "_ubkit._http._tcp" and scan the '_a' fields and get information about services available on the network.

Table 1. Definition of service description in UBKit

key	type	value
_http._tcp.	PTR	[device name]._http._tcp.
_ubkit._http._tcp.	PTR	[device name]._http._tcp.
_[service name]._ubkit._http._tcp	PTR	[device name]._http._tcp.
[device name]._http._tcp.	SRV	[hostname]:[port] …
[device name]._http._tcp.	TXT	_a=[service name], …

5. EXAMPLE

As a sample application using our proposed scheme, we have developed a service icon browser that displays a device's icon when the device participates in the network.

The devices offering this browser services contains the subtype "_face" and describe as additional information: the device's graphic image path like "image=/path/to/image.jpg" and the path to GUI "path=/path/to/page.cgi". By pre-defining just this information, any device icon can be displayed automatically on the browser when devices connected to the network (see Figure 1).

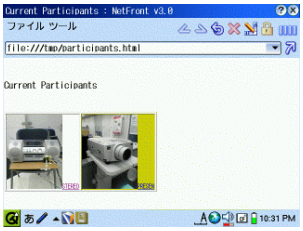

Figure 1. Screenshot of Service Icon Browser

When a device having a "_face._ubkit._http._tcp" participates in the network, other devices get the newly joined device's attributes "image" and "path" and generate a HTML document including the image with a link to the device's GUI (see Figure 2).

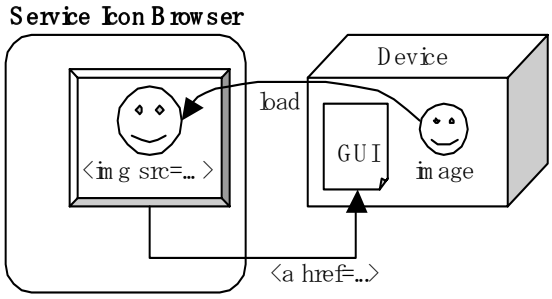

Figure 2. Relation between the Browser and Devices

Since device's images are displayed as a source of <img> tag, a browser can always load the latest image. Therefore, devices generate an image dynamically reflecting the current running status, just like live-camera or any other real time monitoring device.

6. SUMMARY AND CONCLUSIONS

We have developed a scheme in which relatively low-processing devices can be discovered and accessed for their services with zero configuration setup in the no server environment. Some devices are running and implemented with this scheme and automatically controlling each other coupled with sensor-modules in our test-beds.

We are planning to extend this scheme to other device control architecture.

7. REFERENCES

[1] UBKit – Open Platform Lab. http://opl.carc.jp/

[2] Mitsubishi Electric, Co. Press Release, October 2003 http://www.mitsubishielectric.co.jp/news-data/2003/pdf/1008.pdf

[3] http://www.zeroconf.org/Rendezvous/

[4] http://www.zeroconf.org/

[5] http://www.multicastdns.org/

[6] http://www.dns-sd.org/

[7] RFC2782, http://www.ietf.org/rfc/rfc2782.txt

[8] Port Numbers, IANA. http://www.iana.org/assignments/port-numbers