A Dynamic Device Driver for Real-space Programming in Ubiquitous Computing Environment

Makoto Suzuki, Shunsuke Saruwatari, Hiroyuki Morikawa, Tomonori Aoyama
The University of Tokyo
{makoto,saru,mori,aoyama}@mlab.t-u-tokyo.ac.jp

ABSTRACT

In the ubiquitous computing environment, a service description that coordinates sensors and actuators in low power consumption will be an important element. Based on this standpoint, we are developing a new service coordination model that precludes semantics. This paper describes a dynamic loadable device driver manager designed based on this service coordination model.

1. INTRODUCTION

Recent advances of MEMS and semiconductor technology will enable us to integrate a radio frequency circuit, a processor and sensors/actuators into a single-chip. In the near future, it is expected that even clocks, doors, bells and lights have computing function and wireless communication function. For the environment like this, a service description to coordinate sensors and actuators in low power consumption will be an important element. From this viewpoint, we are developing new network architecture ANTH, which includes wireless communication protocol and user interface based on a service coordination model that precludes semantics. The following describes the Dynamic Device Driver. Dynamic Device Driver is a dynamic loadable device driver manager that hides device-dependent implementation, based on this service coordination model.

2. ANTH

We took notice of the simplicity of sensors and actuators, and introduced Bind Control Model, a service coordination model that precludes semantics like kinds of data, to ANTH. Fig.1 describes Bind Control Model. In Bind Control Model, all device functions are abstracted into *event*, *action* or *controller*. An *event* is what a sensor can detect the change of state, and an *action* is a function which an actuator can do. A *controller* is a function like a baton to bind an *action* to an *event*, and a *controller* can construct all services in ANTH.

The *event* and *action* that are bound by *controller* have same ID. The operations of *controller* are only to get ID from and to set ID to an *event* or an *action*, and it is the key of Bind Control Model that a user can coordinate *events* and *actions* by these simple operations. We designed an user interface, a computer architecture and a network architecture of ANTH based on this service coordination model. In this way, ANTH achieves a service coordination framework that is simple, flexible, low power consumption and autonomous-decentralized.

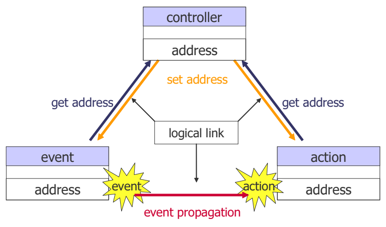

Figure 1: Bind Control Model

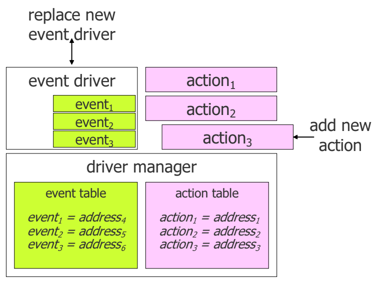

Figure 2: Dynamic Device Driver

3. DYNAMIC DEVICE DRIVER

To realize ANTH, the feature that abstracts device-dependent implementation to ID is required. And, we aim at dynamic addition of functions in ANTH from the viewpoint that users will desire to download new software to embedded devices when they have network reachability. We introduced Dynamic Device Driver, a manager of device drivers for low power consumption devices, to ANTH. By Dynamic Device Driver, a user can download new device drivers from the Internet, and add new functions to devices according to the user's preference.

3.1 Design and Implementation

Fig.2 shows a brief overview of Dynamic Device Driver. Dynamic Device Driver consists of an event driver, a driver manager, action modules. An action module is a software module that corresponds with *action* in Bind Control Model, and this is added to a device from a control terminal, which is implemented *controller* in Bind Control Model on. An event driver is a software module that has several *events* in Bind Control Model, and issues an *event* depending on the situation. A driver manager manages correspondences of each *actions*, *events* and IDs of Bind Control Model.

To implement Dynamic Device Driver, a dynamic loading framework of software modules for low power comsumption devices is required. Several means of dynamic loading of software modules for low power consumption devices are proposed for wireless sensor networks[1][2]. Though, it is difficult to adopt these frameworks to ANTH without modification. It is because wireless sensor networks work in a very specific architecture and the utilization of software framework of wireless sensor networks places a burden on developers.

In these viewpoints, we implemented a dynamic loading framework based on Bind Control Model for low power consumption devices. We utilized ANTH Modules (fig.3) devel-

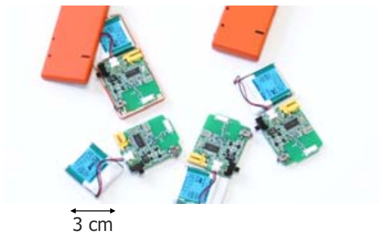

Figure 3: ANTH Module

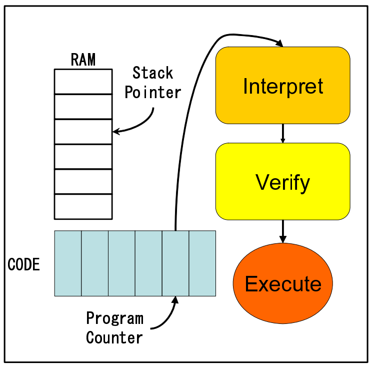

Figure 4: Virtual Machine

oped by us for implementation as hardware platform. Tab.1 shows the specification of ANTH Module. We designed that Dynamic Device Driver uses a virtual machine as an execution engine because we think that a protection feature is required when dynamic loading of software modules externally is allowed. We put verify phase in sequence of virtual machine execution as fig.4. This enables protection of memory or other hardware resource in 1-chip microcontroller without hardware protection feature.

4. PRELIMINARY EVALUATION

To evaluate Dynamic Device Driver, we prepared three modules. *Module1* is a module that consists of only empty *for* loop. *Module2* is a module that outputs random value to the general purpose I/O. *Module3* is a module that sends a message through 9,600 bps UART. For these three modules, we measured the performance and compared with the performance of the corresponding native code. Tab.2 shows the modules' sizes and the performance ratios.

4.1 Performance

Tab.2 shows the ratios of the execution performance on the virtual machine to that of corresponding native code. About *module1*, the performance ratio is about 34.7. It indicates that the performance is very slow compared with that of corresponding native code when a module consists of fine granularity virtual instructions. About *module2*, the performance ratio is about 1.28. It shows that complex virtual instructions like rand() call can reduce the overhead of interpretation of virtual instructions to a large extent relatively. About *module3*, the performance ratio is about 1.01. Because *module3* includes block instruction that transmits data through slow 9,600 bps UART, the interpretation of virtual machine does not become a bottleneck.

We evaluated *module3* in detail. We changed the logical throughput and measured the change of the effect throughput. Fig.5 shows the effect throughput of UART changing the logical throughput of UART to 9,600 bps, 19,200 bps, 57,600 bps and 115,200 bps. When the logical throughput is 115,200 bps, the effect throughput ratio is 1.27 because the interpretation of virtual machine becomes a bottleneck.

Table 1: ANTH Module Specification

CPU	Microchip PIC18LF4620 RAM : 4 KByte ROM : 64 KByte
Wireless	ChipCon CC1000
I/O	UART, GPIO, I^2C
Peripherals	Four LEDs, Real Time Clock, External EEPROM (16 KByte)

Table 2: Preliminary Evaluation

	Module Size	Performance Ratio
Module1	26 Byte	1:34.7
Module2	3 Byte	1:1.28
Module3	29 Byte	1:1.01

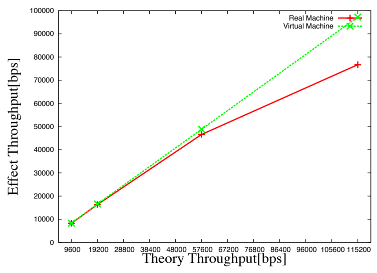

Figure 5: UART Effect Throughput

5. CONCLUSION AND FUTURE WORK

This paper has described the design, implementation and preliminary evaluation of Dynamic Device Driver, a manager of device drivers designed based on new service coordination model that can coordinate sensors and actuators in low power consumption. Dynamic Device Driver enable a user to add new functions to devices according to the user's preference. Our evaluations shows that application specific instruction set decrease the overhead of interpretation greatly and Dynamic Device Driver based on our virtual machine can execute modules almost as fast as native code. Currently, we are designing the instruction set specialized in ANTH.

6. ACKNOWLEDGEMENT

This work is supported by Ministry of Internal Affairs and Communications.

7. REFERENCES

[1] C. C. Han, R. Kumar, R. Shea, E. Kohler, and M. B. Srivastava. A Dynamic Operating System for Sensor Nodes. In *Proceedings of the 3rd International Conference on Mobile Systems, Applications, and Services (MobiSys'05)*, pages 163–176, Seattle, Washington, June 2005.

[2] P. Levis and D. Culler. Mate: A Tiny Virtual Machine for Sensor Networks. In *Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'02)*, pages 85–95, San Jose, California, October 2002.