A Rule-based Acceleration Data Processing Engine for Small Sensor Node

Kenji KODAMA
Graduate School of Engineering, Kobe University
kodama@stu.kobe-u.ac.jp

Naotaka FUJITA
Graduate School of Engineering, Kobe University
nfujita@stu.kobe-u.ac.jp

Yutaka YANAGISAWA
NTT Communication Science Laboratories
yutaka@cslab.kecl.ntt.co.jp

Tsutomu TERADA
Graduate School of Engineering, Kobe University
tsutomu@eedept.kobe-u.ac.jp

Masahiko TSUKAMOTO
Graduate School of Engineering, Kobe University
tuka@kobe-u.ac.jp

ABSTRACT

In recent years, various small sensor nodes have been developed for recognizing real world situations and events for the development of context-aware systems. We consider that the acceleration sensor is one crucial element for recognizing various types of situations because it has rich and simple information. An application system using acceleration data requires the following features: 1) rapid processing of data without large memory since the amount of acceleration data is much greater than the amount of other sensor data, 2) a node that reduces the amount of data sent to a server, and 3) systems that can be easily configured by users at low cost. Current sensor nodes, however, do not have enough functions to satisfy these requirements. We propose a rule-based data processing engine for processing acceleration data. Our proposed engine rewrites the rules on each node with a few bytes of data. We evaluated our rule-based engine on our small sensor node called the Motion sensing and Communication Minimized Chip (MoCoMi-Chip).

Categories and Subject Descriptors

C.2.4 [Distributed applications]: Distributed Systems

General Terms

Design, Implementation, Experimentation, Performance

Keywords

Acceleration Sensor, Rule Base, Wireless Sensor Node

MidSens'08, December 1-5, 2008, Leuven, Belgium

1. INTRODUCTION

Recent technological advances have created pervasive computing systems using many small computer nodes with various types of sensors to recognize events in the real world. The nodes enable us to develop new services, based on huge amounts of information such as events, contexts, and situations.

Many nodes have several types of sensors such as temperature, proximity, acceleration, light, pressure, and magnetic for extracting beneficial real information [1]. Among the above, the acceleration sensor is one of the most significant devices for obtaining such sensor data as dynamic motions, both of humans and objects, in real time. Therefore, most sensor nodes have acceleration sensors.

Generally, the amount of acceleration data obtained in one second is much larger than other sensor data because an acceleration sensor obtains data at a high frequency. The increase in the amount of data causes a corresponding rise in communication cost between sensor nodes. Another problem that concerns users is they must frequently adjust each setting of sensor nodes to accurately process the acceleration data because these data are strongly affected by the real world. For this reason, acceleration sensor nodes require operation-specified acceleration data.

We consider four requirements: 1) simple description of program, 2) reconfigurable program, 3) promptitude, and 4) low communication cost. Currently, however, the same method is used to process all types of sensors, even though acceleration sensors have quite different features from other sensor nodes. In other words, previous sensor nodes do not have functions that satisfy the requirements of acceleration sensors because they use acceleration data as well as other sensor data sampled at low frequency. To deal with the acceleration data on sensor nodes, the data processing system on a node must introduce specified mechanisms to process the acceleration data on each sensor node.

Therefore, we propose a rule-based data processing engine for processing acceleration data on small sensor nodes. Using rule description, users describe the operations of sensor nodes simply and compactly. Furthermore, to modify these rules, users can change operations dynamically in real time. Our proposed engine processes sensor data at high frequen-

cies because it uses a simple processing method. The engine reduces communication cost because sensor nodes only send sensor data or context information when the engine detects events. Moreover, we describe an implementation of our rule-based engine on our small sensor node called the Motion sensing and Communication Minimized Chip (MoCoMi-Chip). We also experimentally evaluated the performance of our engine based on the above four system requirements.

This paper is organized as follows: we survey related works in Section 2. In Section 3, we introduce our approach and describe the design of our proposed engine in Section 4. In Section 5, we discuss an evaluation of our engine. Finally, we conclude in Section 6 with our summary and directions for future work.

2. RELATED WORK

In this section we show related systems using sensor nodes. First, we introduce systems that recognize context from the real world. Second, we introduce database systems that process sensor data on sensor networks. Finally, we introduce rule-based systems for sensor nodes.

Many researchers have attempted to obtain general human or object context information using sensor nodes. In these systems, acceleration data are crucial to obtain both context and situational information. For example, The MediaCup uses sensor nodes to obtain general context information [2]. The DigiClip uses sensor nodes attached to documents to manage them more effectively [3]. User activity recognition systems also use acceleration data for inferring activities. Kawahara et al. proposed methods to estimate a person's motion using sensor nodes [4]. These systems process the data stored in server computers without processing on the sensor nodes.

To store sensor data in a server, each node must send data via a wireless network. As the amount of sensor data obtained by sensor nodes increases, the amount of data sent to the server also increases. TinyDB can reduce the amount of sensor data by using acquisitional query processing (ACQP) in sensor networks [5]. TinyDB processes a query described in a SQL-like query language, the user indicates the minimum set of necessary sensor data to send to the server. However, this system is not suited for acceleration data for estimating motion.

In the rest of this section, we mention the reconfigurable mechanisms for sensor nodes. Over-the-air programming (OTAP) is proposed to rewrite programs stored on many sensor nodes in wireless networks [6]. For example, both the MICA MOTE and Smart-Its Particle introduced OTAP for reducing rewriting costs [7, 8]. However, since OTAP incurs a large communication cost and cannot rewrite a program in real time, this method cannot be adapted to sensor networks in certain cases.

Some pervasive services using rule-based systems have been proposed to adapt various environments for sensor nodes. A rule-based system operates small devices as an event-driven system and changes programs quickly. AhroD and DYCOM use rule-based systems to operate small sensor nodes for pervasive services [9, 10]. AhroD is a ubiquitous computing device using event-condition-action (ECA) rules whose description simply and compactly describes device operation. AhroD has operation rules that determine which applications should be executed and processes them in sequence. However, AhroD cannot process sensor data because it uses binary signals to evaluate input signals. DYCOME is a context-oriented switcher using a rule-based system to process sensor data for sensor nodes. By processing raw sensor data, DYCOME dynamically switches an active application at an event. However, the rule description in DYCOM cannot describe a motion event because it only processes raw acceleration and other sensor data.

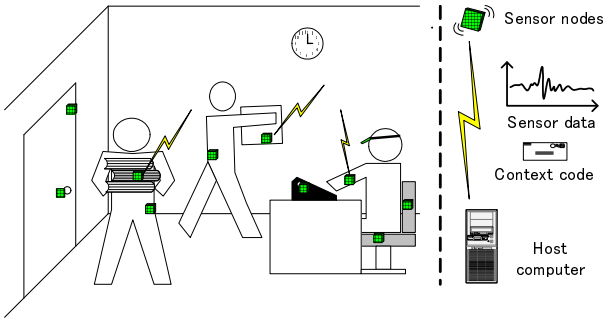

Figure 1: An application image.

3. APPROACHES

As mentioned in Section 1, we propose a method to operate small autonomous sensor nodes that process acceleration data. After describing several examples of our application systems and the framework of a sensor network, we discuss the system requirements for processing acceleration data on our sensor nodes for application systems. Finally, we show an outline of the method to satisfy these requirements.

3.1 Example Application Systems

We focus on sensor nodes attached to various indoor objects. The application systems use collected data from each sensor node to extract real-world events and context information.

Figure 1 shows an example of an application system. In the office, each person and most objects have a sensor node. Each sensor node obtains various types of sensor data, and the node extracts beneficial data from these data. After extraction, the node sends the data via a wireless network to a host computer. The host computer integrates the received sensor data to calculate complex moving situations or events that each node extracts. The host computer also sends the information to an application system. As a result, the application system provide meaningful context-aware services.

Generally, each node has many types of sensors, such as acceleration, temperature and light. Acceleration sensors are crucial in obtaining data of real-world situations since they catch detailed motions or tilts of objects. Most context-aware systems adapt acceleration sensors to extract contexts such as human activities, situations, and events. The acceleration sensor nodes in these systems must have higher data processing capability than other sensor nodes.

On the other hand, attachable sensor nodes have poor calculation capabilities and battery capacities because they must be small and low in cost. In other words, acceleration sensor nodes must have sophisticated processing capabilities with small resources. To satisfy these requirements, we must develop a processing method for reducing processing costs.

3.2 System Requirements

We discuss the requirements of this method for processing acceleration data on a small sensor node.

3.2.1 *Application System Requirements*

- Simple description
  Application systems are used by many people, some of whom do not have experience configuring sensor nodes. Therefore, sensor node programs should be simple and brief.
- Reconfigurable
  In the example application system, sensor node programs may be different for each node due to the provided services or their locations. The application system should change the method for processing sensor data of each node for each purpose. For example, if a user wants to add a new event for detection, the program for processing the data on the sensor nodes must be rewritten in a routine. The features of the sensor data are different for each attached node. Moreover, each node has different hardware features so that we must give individual settings for each node. Since sensor nodes often need reconfiguration, they require a large reconfiguration costs. To adapt the processing method to frequently changing situations, it is necessary to introduce a mechanism for rewriting programs on a node without much communication and processing cost.

3.2.2 *Acceleration Data Requirements*

- Promptitude
  Acceleration data have different features from other types of sensor data, for example, high frequency sampling, volatility of data, and independence of data. Therefore, acceleration sensor nodes need high frequency sensing and rapid response. For example, to track such human motions as walking and running, the nodes must sense the acceleration data of the person's body between 10-100 Hz. On the other hand, temperature and light are sampled up to 10 Hz. If an alert system detects such anomalous events as collisions or products falling in a factory, the system must respond rapidly until a person comes to help.
- Low communication cost
  In general, the amount of acceleration data is larger than other sensing data because the acceleration sensor must obtain data to trace a motion at a high frequency. If a node sends all obtained data to the host computer via a wireless network, the amount of data sent from a number of nodes may exceed the capacity of the wireless network because a small sensor node has only a low band-rate wireless communication device. To avoid such situations, acceleration sensor nodes must reduce their transmitted data; that is, sensor nodes must only send the necessary data. Thus, acceleration data should be processed on the sensor nodes. However, processing must be simple, because sensor nodes have a small amount of resources.

3.3 Our Approach

We propose an acceleration data processing engine that satisfies the above four requirements for small acceleration sensor nodes. We briefly show how to satisfy these requirements with this engine. For a simple description, we adapt a rule-based language to process sensor data. Our engine uses an If-Then rule that consists of conditions and actions. The condition of the rules is based on acceleration data. If a condition is implemented, a corresponding action is executed. Using rule-based language, we can rewrite the sensor node program. To add or delete rules, application systems quickly change the operation of sensor nodes and reduce the communication cost of reconfiguration. For promptitude, we adapt threshold comparisons for determining the conditions, which are described as feature values: instantaneous value, average value, sum of deviation, derivative value, and number of counters. By event-driven operating to process the rules based on acceleration data, sensor nodes reduce communication costs. Most sensor nodes only send context information as a short piece of text data to a host computer, not long raw sensor data.

4. RULE ENGINE FOR ACCELERATION DATA

4.1 Rule Engine Design

Since both sensing acceleration and processing data are elementary tasks on sensor nodes, an engine must have their tasks before every evaluation process cycle. To operate sensor nodes at accurate intervals, the engine uses a hardware timer for sensing in a cycle. In other words, since the hardware timer announces when the next cycle starts after processing all rules, the engine periodically operates the processes on a sensor node. The engine processes the stored rules in order. By having several rules, the sensor nodes operate for several situations. The engine allows combinations of conditions or actions for rule flexibility.

4.2 The Processing Flow

Figure 2 shows the essential processing flow of our designed rule engine on a sensor node.

1. The engine obtains a piece of acceleration data from the sensor device on the node before calculating the feature values of the acceleration data as a preparation for evaluating the conditions. The feature values are used to recognize the motions, tilts, and states of the sensor node.
2. The engine fetches rules one by one before comparing the condition with extracted feature values in the previous step. If the engine finds a matched condition in a rule description with a feature values, the engine checks the next rule. In another case, if the checked condition is a combined condition, the engine fetches each rule from the combined condition and evaluates every fetched condition recursively.
3. If all fetched conditions are matched with the values, the engine executes the action. The engine also checks the rules one by one from the next rules from the first one described rule to the end of the last.
4. The engine waits until there is an interruption from the hardware timer when the engine has completely evaluated the rules.

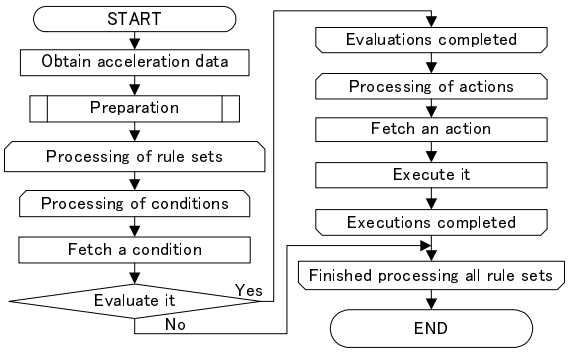

Figure 2: System flowchart.

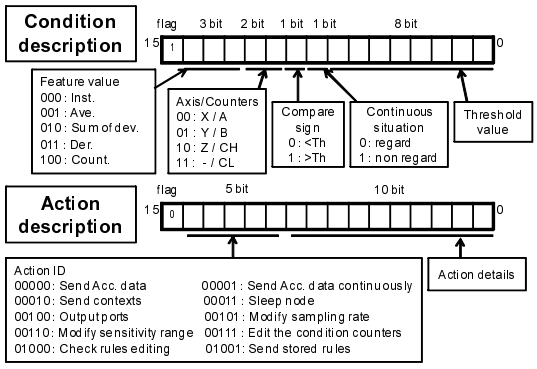

Figure 3: Format of rule description.

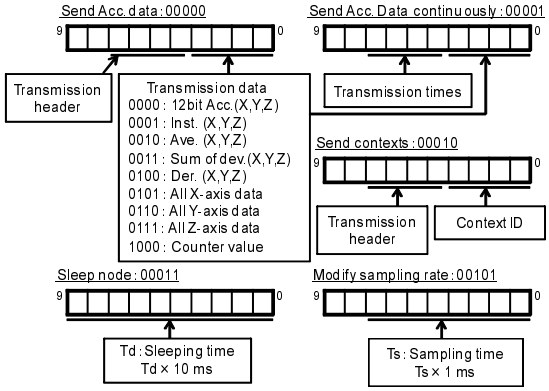

Figure 4: Part of detail formats of action description.

4.3 Conditions

Traditional systems capture motions or situations from acceleration data to calculate average, variance, deviation, FFT, and support vector machine (SVM) as feature values. However, sensor nodes cannot calculate such sophisticated processes as FFT and SVM in real time.

Our engine calculates the next three feature values of the acceleration data for high-speed evaluation of the conditions at the preparation. The engine simply calculate these feature values on the sensor nodes. The average shows tilts or smooth sensor data. Variance is the available value for detecting the presence of motion, but it cannot be calculated rapidly using a low-cost MCU. The conditions of our rules use a sum of deviation that resembles variance to detect the presence of motion. The engine calculates an average and a sum of deviation in the last ten samples. The sum of deviation is the sum of the distances between the average and each sample. To detect rapid motions such as an object hits, the engine also uses a derivative value. In addition, the engine uses four counter values (condition counters) to describe continuous situations further to the feature values for evaluation of the conditions. The engine increments and resets the condition counters by executing an action. By using the condition counters, we can describe the condition of continuous situations: if a state occurs more than once, the engine measures the number of occurrences. The engine allows descriptions of conditions that regard a continuous state as an event. Therefore, we can describe the condition of an event starting where a state is continuously detected. The engine evaluates conditions to compare the threshold and feature values by simple processing.

4.4 Actions

The engine allows the execution of the next ten actions by processing the rules for sensor nodes.

Sending acceleration data is a basic operation of sensor nodes. This action sends raw acceleration data, feature values, and condition counters values. The engine must also send acceleration data continuously when the host computer requests for streaming data. To reduce communication cost, the engine enables the context information that detects events and situations of sensor nodes to be sent. Putting the node to sleep is important to conserve the battery. In addition, the engine modify the sampling rate and sensibility range for adapting to situations by executing actions. Port outputs are used for indicator such as LED lights. The condition counters are edited by action to increment and reset for continuous situations. The engine also enables sending of stored rules on sensor nodes and checking for the request of rule editing from the host computer.

4.5 Rule Editing Mechanism

The engine runs the rule-editing mechanism to follow the next four steps. First, a sensor node checks the request of rule editing to send a code to a host computer as an action or a routine in the engine processes. Second, after receiving the code, the host computer sends the request and areas of rule editing. Third, if the rules need to be edited, the sensor node sends a request for rule forwarding. Finally, after rule forwarding, the sensor node verifies the rules by check sum and sends a finished message or a retransmission request.

4.6 Rule Description

The rules consist of combinations of conditions and actions. We use a binary rule that enable us to easily evaluate conditions to compare the threshold value with feature values. The rules are described based on the formats of the rule descriptions shown in Figures 3 and 4. The descriptions of the conditions and actions are fixed-length two-byte data. Users program multi-conditions and multi-actions using a most significant bit(MSB) flag. In fact, continuing conditions that have MSB flags are evaluated as “AND conditions.” To program an “OR conditions,” a user should describe each set of rules. Continuing actions are executed for each action.

A description of a condition consists of five components: kind of values, axis or kind of counter, sign, and regard of continuous situation and threshold values. The threshold value is an 8-bit abbreviated feature values or a value of the condition counters. The feature values are calculated based

on the acceleration data at the preparation.

A description of an action consists of an action ID and a configuration of an action detail. Except for MSB, higher 6-bits are allocated for each action, and the lower 10-bits show the configuration of an action detail. Figure 4 shows a detail part of the formats of the action description. The configuration of an action detail differs for each action. For example, for an action that sends acceleration data, users select the kind of data and sending the data to each axis.

5. EVALUATION

We experimentally evaluated the performance of our proposed mechanism from the viewpoint of the four system requirements. In the experiments, we attached the MoCoMi-Chip to humans and objects. Table 1 lists the specifications of our developed chip, and Table 2 lists the evaluation results. The engine detected events based on rule descriptions. Figure 5 shows our chip attached to a pen, and Figure 6 shows the acceleration data obtained from the chip while writing. The engine had rules that provided accurate descriptions based on exploratory experiments. The size of a transmission packet between the node and server was set to 15 bytes, and the payload was set to 8 bytes.

5.1 Simple Description

To evaluate the simplification of operation description, we compared the amount of code written with our description language with the amount of code description in C language. Because C language is the most popular language for programming MCUs, we adapted a language to compare the simplifications. Using the rule description, a rule is composed of conditions and actions described as binary code whose size is only two bytes. A set of rules is described as four bytes in hex format. When we describe an operation to detect an event, the rule description is 20% of the code described in C language. It should be noted that C language is used to describe the details of operation using calculations, control statements, and functions for sensor nodes. For sensor node operations that only have cyclical routines, however, our rule description is simpler to use than C language.

5.2 Reconfigurable

We measured the time of the rule editing routine to test a real-time reconfiguration. In this experiment, our chip had 32 rules and edited eight and 16 rules into new rules. The routine took 12.4 ms when eight rules were changed, and 14.5 ms when 16 rules were changed. This result suggests that the engine edits the rules in the sampling cycle to sense the motions of humans and objects. In the other words, the engine changes the rules dynamically without pausing operation to installed sensor nodes in the example application.

5.3 Promptitude

To evaluate the promptitude, we measured the operation time of our proposed engine. The operation times includes both obtaining the sensor data and sending messages of events as context codes to the server. To measure operation time, we used a hardware timer on MCU in our chip. In this experiment, the engine detects such events as opening and closing a door, dropping a chip, writing, and a moving human. Table 2 lists the processing time results. In most cases, the processing time was less than 2.5 ms. In other words,

Table 1: Specifications of MoCoMi-Chip.

MCU	8051 compatible microcontroller
Operation frequency	16 MHz
Program memory	4 KB
Data memory	256 byte
Radio transceiver	nRF2401
RF data rate	1 Mbps/250 Kbps
Communication distance	approx. 25 m
Technical regulations conformity certification	2.4 GHz band wide-band low-power data communications system in Japan
Interfaces	UART, SPI, Digital I/O, PWM
Acceleration full scale	±2 G/6 G selectable
Acceleration resolution	12 bit
Consumption current	Transmission : Typ.20 mA, Sleep: 4 μA
Power supply	DC 3 to 9 V
Overall size	20×20×3.9 mm
Weight	approx. 1 g

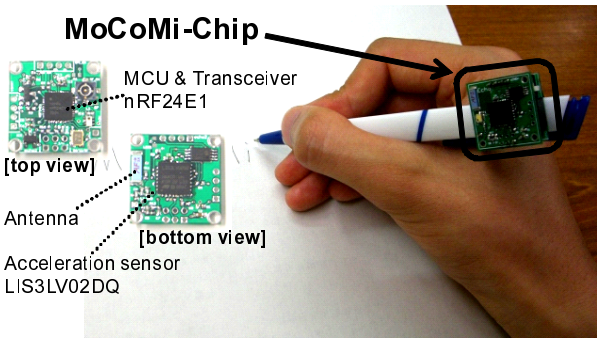

Figure 5: MoCoMi-Chip mounted on a pen

Table 2: Evaluation results

Name of detected events	Tx data [Byte]	Amount of data rate	Number of occurred events	Processing time [ms]	Number of rules	Amount of code in C language [Byte]
Transmit raw 3-axis acceleration data	75000	100	-	-	-	-
Start of free fall	150	0.2	10	2.47	7	138
Fall to floor	585	0.78	39	2.49	12	294
Spots on a dice	270	0.36	18	2.58	18	396
Opening and closing a door	180	0.24	12	2.33	6	130
Opening and closing a drawer	180	0.24	12	2.47	10	216
Start of writing	1575	2.1	105	2.30	4	84
Standing and sitting of human	435	0.58	29	2.38	8	169
Human moving	2010	2.68	134	2.11	3	59
Stop hand motions	240	0.32	16	2.30	4	88

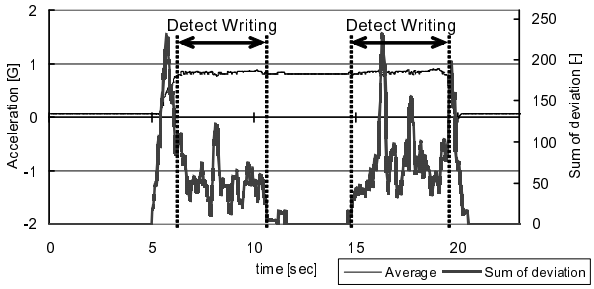

Figure 6: Acceleration data in pen operation

the engine processed events of over 400 Hz. To recognize human motions, an event detection system must generally process detection between 10-100 Hz. Thus, the results indicate that the engine has enough power to detect human events in real time. In addition, we measured the processing time of each part of the operation. The longest processing time was transmission, which took 1.0 ms. Evaluating the rule's condition required about 0.04 ms. To completely finish processing a cycle, we must strictly describe the rule and reduce the number of events found within a process cycle. This technique enables us to reduce the processing time, although it increases the number of rules described in a sensor node.

5.4 Low Communication Cost

To evaluate the communication cost, we experimentally measured the amount of data sent from a sensor node to the server computer. We measured the amount of transmission data by transmitting event context codes and raw three-axes acceleration data to evaluate the communication cost. For the settings, our chip contains rules for detecting the start of each event; an event occurs 10 times per 100 seconds. Table 2 lists the results of this experiment. The amount of transmission data was less than 2,000 bytes when our proposed engine transmitted the context codes of an event. On the other hand, for transmitting raw three-axes acceleration data within 100 seconds, the amount of transmission data was 75,000 bytes. This result indicates that our engine reduces communication cost by one hundredth of the transmission data. Therefore, we believe our engine saves more power of the sensor-node power than the conventional transmission method.

5.5 Discussion

We detected the events of humans and object motions using our proposed engine. However, it could not detect complex situations such as human activities because the engine uses feature values calculated by simple processing. To detect complex situations, we must configure the rules to transmit acceleration data when a motion occurs and calculate them on a host computer, as in traditional systems. Therefore, an system that combines our proposed engine for sensor nodes and host processing conserves battery power of sensor nodes and reduces communication costs.

We implemented our engine to our chip on an Intel 8051 compatible MCU. The program size of the engine was 4 KB. Therefore, we can implement it on most conventional sensor nodes because they have more resources than our chip.

6. CONCLUSION

In this paper, we proposed and designed a rule engine to process acceleration data on small sensor nodes. We developed an acceleration sensor node called MoCoMi-Chip and evaluated the performance of the engine with a node from the viewpoint of the four system requirements. Evaluation results show that the engine satisfies the four requirements. In the future, we plan to develop a system that easily describes and generates rules. We also plan to develop a rule engine that processes other sensor data.

7. ACKNOWLEDGMENTS

This research was partially supported by the Ministry of Education, Science, Sports and Culture, Grant-in-Aid for Scientific Research on Priority Areas, 19024056, 2007 and Scientific Research (A), 20240009, 2008.

8. REFERENCES

[1] Beigl, M., Krohn, A., Zimmer, T. and Decker, C.: Typical Sensors needed in Ubiquitous and Pervasive Computing, In Proc. of INSS 2004, pp. 153–158 (2004).

[2] Beigl, M., Gellersen, H.-W. and Schmidt, A.: Mediacups: Experience with Design and Use of Computer-Augmented Everyday Artefacts, Computer Networks, vol.35, No.4, pp.401–409 (2001).

[3] Decker, C., Beigl, M., Eames, A. and Kubach, U.: DigiClip: Activating Physical Documents, In Proc. of the Intl. Conf. on Distributed Computing Systems Workshops (ICDCSW'04), pp.388–393 (2004).

[4] Kawahara, Y., Kurasawa, H. and Morikawa, H.: Recognizing User Context Using Mobile Handsets with Acceleration Sensor, In Proc. of the IEEE Intl. Conf. on Portable Information Devices (IEEE Portable 2007) (2007).

[5] Madden, S., Franklin, M., Hellerstein, J. and Hong, W.: TinyDB: an Acquisitional Query Processing System for Sensor Networks, ACM Tran. on Database Systems, Vol.30, No.1, pp.122–173 (2005).

[6] Kulkarni, S., Wang, L.: MNP: Multihop Network Reprogramming Service for Sensor Networks, In Proc. of the IEEE Intl. Conf. on Distributed Computing Systems, pp. 7–16 (2005).

[7] Crossbow Technology, inc. http://www.xbow.com/.

[8] Holmquist, L., Mattern, F., Schiele, B., Alahuhta, P., Beigl, M. and Gellersen, H.-W.: Smart-Its Friends: A Technique for Users to Easily Establish Connections between Smart Artefacts, In Proc. of the Intl. Conf. on Ubiquitous Computing (UbiComp 2001), pp. 116–122 (2001).

[9] Terada, T., Tsukamoto, M., Hayakawa, K., Yoshihisa, T., Kishino, Y., Nishio, S. and Kashitani, A.: Ubiquitous Chip: a Rule-based I/O Control Device for Ubiquitous Computing, In Proc. of the Intl. Conf. on Pervasive Computing (Pervasive 2004), pp. 238–253 (2004).

[10] Koizumi, K., Sakakibara, H., Iwai, M. and Tokuda, H.: A Context-Oriented Application Switching Mechanism for Daily Life Supports, the Intl. Conf. on Ubiquitous Computing (UbiComp 2005), Poster Session (2005).