ユビキタス環境を情報空間化する組込みデータベース

Materializing Ubiquitous Computing Environments: A Technical Report

倉光 君郎 [1)]
Kimio KURAMITSU

1) 東京大学大学院情報学環　（〒113-0033 東京都文京区本郷 7-3-1 情報学環ビル 2 階
E-mail: kuramitsu @ iii.u-tokyo.ac.jp ）

ABSTRACT. A key challenge in ubiquitous computing is how to manage ambient information about the environment where a variety of devices appear. Traditional capability descriptions are inadequate for capturing dynamically changing situations. We propose the Toibox DVMS system, a multi-embedded database system for integrating a collection of database views – which each embedded database self-describes. The integrated views are materialized; updates over the view provoke relevant operations over selected devices. The materialized views are decentralized and incrementally maintained by data dissemination. This paper will present the design of the Toibox system, based on our initial prototype implementations.

1 Introduction

The continuous evolution of micro-electronics technologies makes computers small, and places them everywhere in our daily environments. This way of computing is called *ubiquitous computing* [17], or *pervasive computing*, which is characterized by *smart* applications; as an example, we can consider a Bluetooth washing machine that interact with low-cost Bluetooth chips embedded in garments carrying laundry instructions such as suggested water temperatures, detergent requirements, and cycle settings [6]. In the traditional literature of ubiquitous computing, the focus is mainly on the hardware technologies such as Bluetooth and RFID [15], but data management and its interoperability is essentially crucial to achieve such smart applications in the real-world context.

This paper presents our initial attempt to exploring data management in ubiquitous computing environments. The Toibox Data View Management System (DVMS) has been developed, based on a collection of multiple embedded databases. Figure 1 illustrates the Toibox system. The idea underlying our attempt is simple; first, we put a very small DVMS system to each device in ubiquitous computing environments. The DVMS provides a database view that represents the specs, the status and the functionality of the device, interacting with sensors and actuators. (We use DVMS instead of DBMS because the data view includes not only recorded values but also sensed values.) Second, we aggregate such device views into an integrated one as a collection of information sources that represents the environment. The integrated views are maintained as *materialized views*, where updates over a materialized view provoke changes over the corresponding devices.

Materialized view is a well-known approach to efficient querying for different data sources. In the relational database world, several view maintenance techniques has been intensively studied [8]. However, it is also known that the refreshment is too costly in distributed database environments. More recently, data warehouses or database wrappers have been developed to integrate database sources across the Internet, but the materialization is difficult [1]. Furthermore, in mobile and ubiquitous computing environments, the sources dynamically appear or disappear. But, the volume of source data is supposed to be relative small. This motivated us to develop a new view maintenance for embedded DVMS, incorporating with data dissemination [2].

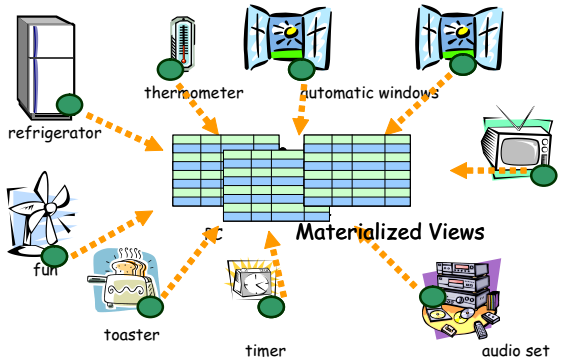

図 1: Materializing Ubiquitous Computing Environments Through Device DVMS

This paper presents the Toibox DVMS with our prototyped implementations. The rest of the paper proceeds as follows. In the Section 2, we state the problem motivating us. In Section 3, we sketch the Toibox DVMS architecture. In Section 4, we show our initial implementations. In Section 5, we study performance evaluation using the cost model. In Section 6, we conclude the paper.

2 Problem Statement

A *smart* application needs various kinds of information about devices, persons, and other physical objects in ubiquitous computing environments. The information would be obtained from a variety of methods, such as sensor networks, statistical analysis, and an inferencing system. Among them, we focus on the *snapshot* data view that devices can capture. In this paper, we introduce a scenario that states the problem that we highlight in the Toibox DVMS system.

2.1 Scenario

A *ubiquitous computing environment* is characterized by a variety of multitude physical-objects (e.g., digital appliances and sensors) that have a (small) computer with communication capability. The objects are functionally irregular, and furthermore some of these objects frequently appear or disappear in the environment. In order for a client device in such an environment to interact with other devices, it is of importance to know ambient information, including what kinds of devices exist and what are going on those devices. To start, we consider the following scenario.

Scenario. The entertainment system was belting out the Beatles' "We Can Work It Out" when the mobile phone rang. When Pete answered, his phone turned the sound down by sending a message to all the other local devices that had a volume control.

The above scenario, originally described in the introduction of the Semantic Web vision paper [3], brings us a good discussion stage for ubiquitous computing. Apparently, the phone needs to know ambient information to control the "right" devices in the given context.

2.2 Pervious Works

To manage ambient information for ubiquitous computing, we consider three possible approaches: *service discovery*, *stream database*, and *embedded web server* – which are originally developed for different purposes. To make the problem clear, we start by examining these approaches.

Service Discovery. A typical example of enabling ubiquitous computing is Sun Microsystems's Jini and Microsoft Universal Plug and Play [11, 16]. These systems focus chiefly on the control of the devices, especially with automatic and *ad hoc* interactions. Each of the local devices is described as a set of services (called *capability description*). The descriptions are registered in a lookup service and then published to service clients. Thus, a service client (say, a mobile phone in the scenario) can locate the demanded service (the volume control service) and then invocate the services (to turn the sound down). To date, several significant improvements for the service discovery have been studied in terms of security, scalability, and multimedia [7, 9, 14]. It is however important to note that the services are mostly described in XML. As a result, the discovery is static and not reflected by dynamic situations. Suppose one who listens to music with a headphone. It is unnecessary for the phone to mute the music.

Stream Data Management. Stream database techniques, chiefly discussed in contexts of sensor networks [4, 12, 13], can be though of as another candidate to capture fresh information about changing environments. The changed values are notified in real-time through data streams. In the scenario, we can suppose a data tuple (device, volume, speaker), carrying the amount of the volume and whether or not the user uses a headphone. However, the stream database is by nature read-only and we are not able to update data values to control the environment with a certain intention. That is, it is meaningless if the phone modifies the volume value in the data stream.

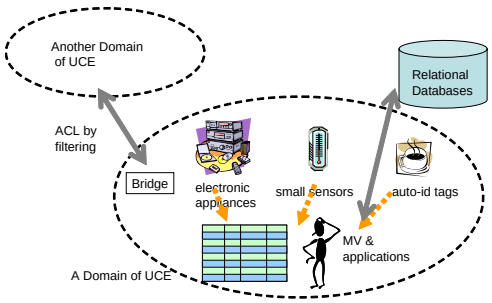

図 2: Architecture of Toibox

Embedded Web Server. The WWW architecture has been broadly accepted in the embedded systems for providing the end users with a means to set parameters to the systems, as well as acquire the "current" status. As shown in the Cooltown project [5], the embedded WWW architecture is a promising candidate for information-rich ubiquitous computing. However, the current WWW interfaces, such as HTML/CGI, are strictly limited to a collaborative interoperation across the servers. Although the phone user would be able to manually turn off the music through an embedded web browser, he or she has to send many times "sound off" messages from different pages that individual devices provide.

To sum up, the service discovery mechanism lacks the freshness of ambient information; the stream database lacks the update (control) operation; the embedded web server lacks the collaborative view among different devices. The goal of the Toibox is to cover all these aspects of ambient information.

3 Toibox DVMS Architecture

In this section, we present the architecture of the Toibox Data View Management System, a new framework to manage ambient information in ubiquitous computing environments.

3.1 Overview

There are a variety of entities appearing in ubiquitous computing environments. The Toibox architecture broadly classifies them into two kinds of nodes. One is a small and embedded DBMS, called *Toi*, providing an information source about itself, and another is a *Toibox* that aggregate the sources to create integrated views. Figure 2 shows the overview of the Toibox architecture.

Basically, the Toi systems are embedded into electronic appliances or sensors. Each Toi has a unique global identifier (TID, or Toibox ID) for the consistent identification. The Toi supports only low level input/output database interfaces that are designed to

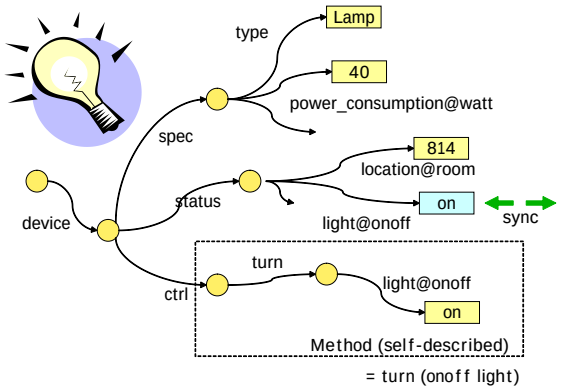

図 3: Data Representation of an Electronic Lamp

communicate the Toibox systems. (The interactions are based on Toibox Data Model, which will be detailed later.) This enables us to implement Tois in small footprint systems. Supposing the correspondence between Auto-ID [15] and TID, we can incorporate information about physical (non-electronic) objects, which relational databases store remotely.

The Toibox system, on the other hand, supports a fully-featured query processing over views that are aggregated from distributed Toi sources. The Toibox views are synchronized with individual source views; changes on the one hand affect the corresponding values on the other hand. (This refresh mechanism will be detailed later.) We allow the Toibox system to play a role in the source provider as a Toi. As a result, multiple Toibox views are maintained in a decentralized manner.

Currently, we do not introduce any access control systems to the inter-connection between the Toi systems and the Toibox systems. We consider the unit of a ubiquitous computing environment to be a room in the real-world; that is, if ones are permitted to enter a room, they also are permitted to access any devices with equipped on the room. We begin to examine a bridge system that interconnects with multiple ubiquitous computing environments. Although we suppose that the bridge will help filter unauthorized queries, the access control and privacy concerns largely remain as an open question.

3.2 Toibox Data Model

The design of database schema in the Toibox is fairly difficult, because we cannot expect what kind of attributes will come from a diversity of Toi sources. To better deal with the diversity, we have chosen semi-structured data model [1] for the Toi sources. Thus, each device is represented by a labeled graph data. Figure 3 shows an example of the data view of an electronic lamp; nodes are depicted by circles (non-terminal node) and rectangle (literal nodes). Each node is assigned with a local identifier, like $\&1, \&2, ..\&n$. Since the root (&0) is associated with TID, we can identify all nodes globally.

We assume a basic terminology to ensure the minimum interoperability between different types of devices; all devices are described by data trees that are directed by the edge labeled device from the root. The device node has tree sub-edges, labeled specs, status, and ctrl, respectively representing the specification of the device (constant values), the current status (variables), and the functional capability (methods). In addition, we associate rich datatypes (`@watt` and `@onoff`) to literal nodes – which carries database values. This prevents the users from misunderstanding queries for type-less representations. (For example, let us compare a case temparature with `temperature@Celsius` and `temperature@Fahrenheit`.)

As with the ordinary (semi-structured) database, the user can update database values. (For simplicity, we ignore restricting tree structures.) It is important to note the difference between records in a disk and the real-world status information; the "real" status does not always follow update operations that the users demand. To control the status with the consistency, we restrain direct value manipulations for the status tree. Instead, the Toibox system supports the remote method invocation mechanism by incorporating it into the data model. The method interfaces are described under the cntl tree.

For example, suppose the user formulates the following update query (`$d` is a variable bounded to devices):

```
update $d.ctrl.turn.light@onoff = 'on'.
comit  $d.ctrl.turn
```

These operations are interpreted in the Toibox query processor, as invocating the following method:

```
$d.turn('on');
```

Each Toi has to update database values on its view in accordance with the results of the processed method. The above method invocation for Figure 3 results in `status.light@onoff = 'on'`.

3.3 Distributed Architecture

The way of coupling distributed databases in Toibox differs from those in traditional multi-database systems. The major differences lies in that the number of Toi sources is dynamically changing whenever devices appear or disappear in the environment. To better deal with such a dynamic environment, we adopt *data dissemination* techniques [2], developed in contexts of mobile databases.

The Toibox system supports an incremental maintenance of the materialized view. Here, we describe the maintenance mechanisms in three cases: *appearance*, *autonomous updates*, and *updates by query*.

Appearance. When a Toi newly appears in the environment, the Toi broadcasts hello message to notify the appearance. The Toibox receiving the message requests the *bulkcopy* to the Toi. The bulk is packed by a collection of all edges and literal nodes (such as edge(&0, device, &1), edge(&1, spec, &2), node(&5, Lamp). The copied sources are constructed as a tree data over the Toibox view. (See Figure 4).

Autonomous Updates. Each Toi autonomously maintains database values by reflecting the real-world status (e.g., sensed data). These changes are asynchronously notified to all the Toibox views through a broadcast channel. The notified message is predicated by sync. For example, if somebody turn off the light depicted in Figure 3, the Toi broadcast sync(TID, &8, 'off'). The Toibox recieving the sync message has to update views to keep the freshness of Toi's value. *Updates by Query.* The users control Tois by updating the Toibox views.

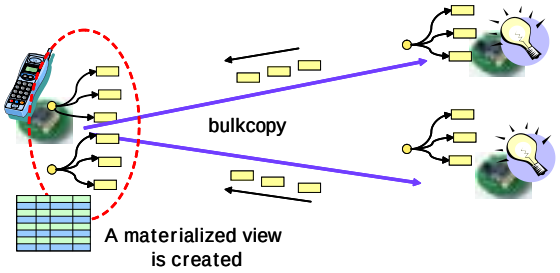

図 4: Creating the Toibox view by bulkcopies from *Tois*

```
foreach $o device
  where status.location@floor = 1
     or status.location@floor = 2
{
   update ctrl.trun.light@onoff = 'off' ;
   commit ctrl.turn ;
}
results (
  status.localtion@floor, status.location@room,
  status.light@onoff
);
```

executing on distributed Tois

図 5: An Example of the Toibox Query Language

The Toibox query language is based on the query languages for semi-structured data [1]. To deal with update operations, we design the FLWOR (foreach-let-where-operation-result) syntax [*1], instead of the traditional SFW (select-from-where) syntax. Figure 5 shows an example of a query for the database in Figure 3. The parts FLWR in the query can be executable over the Toibox view, while the part O has to be forwarded to the selected Toi systems for the execution. The update operations are compiled to update and commit commands that are executable on the Toi system. For example, the Toibox system sends to all Tois (bounded $d) commands update (TID, &11, 'on') and commit(TID, &10).

Figure 6 summarizes distributed query processing in the Toibox architecture. (1) Queries without any updates can be executed only over the Toibox view. In the presence case of update queries, the query processor first selects Tois that are going to be manipulated, and then (2) forwards only update operations through a unicast session to each Toi system. (Note that the Toibox doesn't refresh the view at this stage.) The Toi receiving the update executes the operation to (3) control its status, and then (4) synchronizes updated status through sync to all Toibox views in the environment. (5) The Toibox when receiving the sync refresh their views.

4 Prototyped Implementation

We have implemented several pilot systems along with an unexplored software development program conducted by IPA (Information technologies Promotion Agency, Japan). In the section, we describe the pilot implementations for querying and updating ubiquitous computing environments.

[*1] FLWR is originated in XQuery. We add update capability.

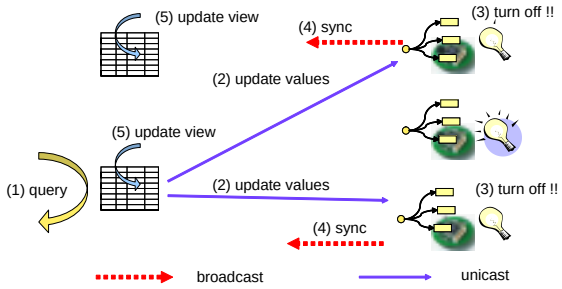

図 6: Inter-Connections between Toiboxes and Tois

4.1 Toi systems

We developed three kinds of *Toi* systems, which are intended to respectively evaluate a small footprint implementation, the descriptive power of Toibox data model, and the inter-connections in ubiquitous computing environments.

Small Footprint Toi can be implemented as a simple wrapper for the existing embedded system, when the status and the capability of the embedded system are statically mapped to the Toibox data. We have been developed several small Tois on embedded computing devices, such as an electronic lamp and a temperature sensor. In our estimation, we can developed the core of Toi with at most 3000 lines in C.

Descriptive Power The Toibox data model is based on semi-structured data in order to better represent the diversity of information structure that each device has differently. To show the descriptive power of the Toibox data model, we materialize the functionality of the latest DVD/HDD Recorder (Toshiba RD-XS40). We can query over the Toibox view for setting timer-controlled recording, as well as for playing the movies on the HDD.

Inter-Connection We have developed an experimental environment of ubiquitous computing, based on a collection of multiple Embedded Linux PDAs (SHARP/Zaurus SL-B500) (Figure 7. These PDAs are connected together through wireless LAN (IEEE801.b), and play roles in digital appliances. (The appliances emulated are running on Personal Java.) The Toi systems build on top of these emulators can provide information sources for the Toibox views.

4.2 Toibox System

ToiboxExplorer (Figure 8) is a gui-based tool that allows the user to manually manipulate the ubiquitous computing environments by querying over the Toibox view. A query entered in the upper text field is executed over the view that the ToiboxExplorer maintain. The result of the query is displayed on the bottom table with a relational formatting. As we described in Section 3.3, all selection and formatting can be executed only over the Toibox view.

Using the ToiboxExplorer, we can query (or update) all our implemented Toi systems. More interestingly, queries executed in the Toibox view behave as a sort of continuous queries for a stream database, since the

図 7: An Emulator for Ubiquitous Computing Environments built over Mobile Linux PDAs

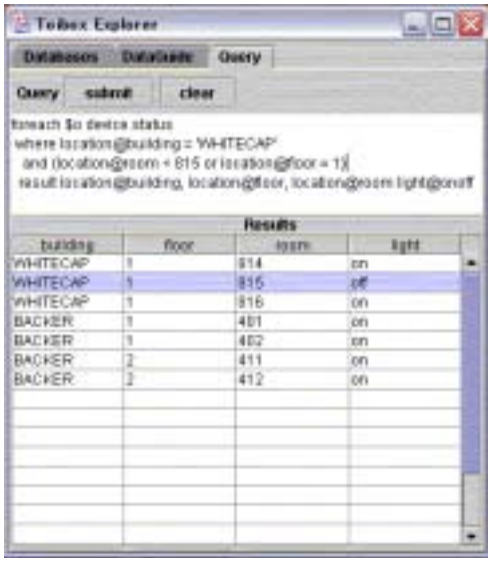

図 8: ToiboxExplorer

view maintenance is viewed as the data streams of sync messages sent from Toi systems. Thus, we can display changing situations in the table according to the refreshment of the Toibox views.

5 Discussion

In this section, we discuss the pros and cons of the performance issues in the Toibox architecture.

5.1 Traffic Cost Model

A ubiquitous computing environment usually swarms with a variety of computing devices. For simplicity in the cost model environment, we only distinguish two kinds of devices: N_s Toi sources and N_c clients (with Toibox views). The variety of the source is characterized by α, the source selectivity coefficient ($0 \leq \alpha \leq 1$).

Appearance. We start by the cost of the view malignance when a device newly appears in the environment. First, the messages hello are exchanged among the devices through a broadcast channel. The size can be estimated by $(N_s + N_c) \times M_{\mathsf{hello}}$, where M_{hello} is the size of the message hello. Then, the source is incorporated into the Toibox views. Let M_{lview} be the

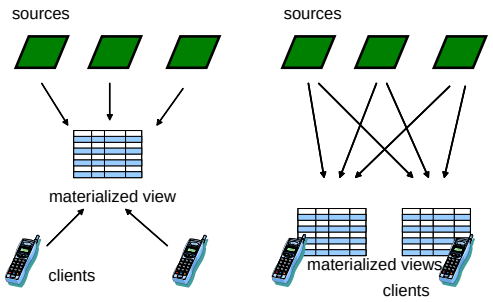

図 9: A Centralized View vs. Decentralized Views (Toibox)

average size of the source. In the case of Toi appearance (M_1), the source is copied to each Toibox, while in the case of Toibox appearance (M_2), the Toibox has to create the materialized view from all Toi systems in the environment. (For simplicity, we assume this process is done by bulkcopies from the exisiting Toibox.)

$$M_1 = (N_s + N_c) \times M_{\mathsf{hello}} + N_c \times M_{lview}$$
$$M_2 = (N_s + N_c) \times M_{\mathsf{hello}} + N_s \times M_{lview}$$

Refreshment. Next, we turn to a case in which the source views in a group of *Toi*s are changed at the same time. The sync messages are disseminated through a broadcast channel. Let R_u be updates/second, and M_{sync} be the average size of updated values for the synchronization. The traffic can be estimated:

$$M_3 = R_u \times N_s \times M_{sync}$$

Updates by Queries. Finally, we focus on the traffic in query processing. If the query includes no update operations, the processing can be executed only within the Toibox view. On the other hand, in the presence of update queries it is necessary to send update messages M_{update} to the selected sources, which cause updates in each Toi. The traffic in the update query is:

$$M_4 = \alpha N_s \times (M_{update} + M_{sync})$$

5.2 Comparative Study

The Toibox DVMS architecture is unique in that (1) the integrated views are materialized in a decentralized manner and (2) the views are refreshed in the data dissemination manner. Here we discuss the merits of the uniqueness.

In the Toibox architecture, we assume each client maintains the Toibox view in a decentralized manner, although the traditional service discovery mechanisms are largely employed in a centralized view. (Figure 9 sketch the difference).

Using the same cost model, we can estimate the traffic in a centralized case by the following formulas (M'_i):

$$M'_1 = N_s \times M_{\mathsf{hello}} + M_{lview} \ (M_1 > M'_1)$$
$$M'_2 = 0 \ (M_2 > M'_2)$$
$$M'_3 = R_u \times N_s \times M_{sync} \ (M_3 = M'_3)$$
$$M'_4 = N_s \times (M_{query} + M_{sync})$$
$$(M_4 < M'_4, \ M_{update} < M_{query})$$

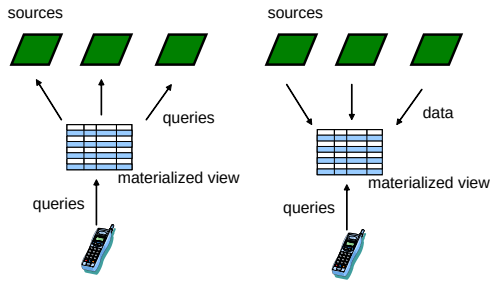

図 10: A Mediator vs. A Data Warehouse (Toibox)

Apparently, the Toibox system is inefficient in the view maintenance of device appearance. On the other hand, the Toibox system is said to be rather efficient in distributed query processing. The broadcast sync messages reduce the traffic as much as the unicast message.

We will turn to the effectiveness of distributed query processing. Although the Toibox system is based on data warehouse architecture (by incorporating data dissemination), the mediator (or the wrapper) architecture is broadly employed, especially in global information systems. (Figure 10 sketch the difference).

To compare the traffic cost, we introduce R_q, queries/second in each client. Thus, the overall query traffic (updates and refreshments) can be estimated:

$$M_T = N_c \times R_q \times (\alpha N_s \times M_{update}) + N_s \times (\alpha R_q + R_u) \times M_{sync}.$$

In the mediator architecture, the source selectivity is regarded as $\alpha = 1$. Then, the total traffic can be estimated:

$$M_W = N_c \times R_q \times (N_s \times (M_{query} + M_{result})).$$

For simplicity, we assume $M_{update} = M_{query}$ and $M_{sync} = M_{result}$. The average sizes of these messages obtained from our prototype implementation are: $M_{update} = 64$ bytes and $M_{sync} = 16$ bytes.

Now, we can say the Toibox is efficient if $M_T/M_W < 1$.

$$\frac{M_T}{M_W} = \frac{\alpha M_{update} + \frac{\alpha R_q + R_u}{R_q} \frac{N_s}{N_c} M_{sync}}{M_{update} + M_{sync}} \quad (1)$$

It is fairly difficult to evaluate the effectiveness of the above formula in the generic ubiquitous computing environment, but throughout our experience the Toibox system is said to be efficient if $\alpha < 0.2$. Note that $\alpha = 0$ means no devices are updated by the query.

6 Conclusion and Future Works

A key challenge in ubiquitous computing is how to manage ambient information about the environment where a variety of devices appear. Traditional capability descriptions are inadequate for capturing dynamically changing situations. We propose the Toibox DVMS system, a multi-embedded database system for integrating a collection of database views – which each embedded database self-describes. The integrated views are materialized; updates over the view provoke relevant operations over selected devices. The originality is that the materialized views are decentralized and incrementally maintained with data dissemination techniques. We showed good performance results by an analysis of our initial prototype experience.

The Toibox DVMS in this work provides a snapshot of ambient information. Future directions we will intensively investigate focused on more sophisticated managements, including the abstraction (or the reasoning) of the snapshot information and the management of privacy concerns. We hope that we will work the Toibox DVMS together with the electronic appliances industry for further refinements in practices.

7 Acknowledgements

The author thanks Prof. Hideyuki Tokuda (Keio University) for several comments.

参考文献

[1] S. Abiteboul, P. Buneman, and D. Suciu. *Data on the Web: From Relations to Semistructured Data and XML*. Morgan Kaufmann, 1999.

[2] D. Barbara. Mobile Computing and Databases – A Survey. *IEEE Transaction on Knowledge and Data Engineering*, 11(1), 1999.

[3] Tim Berners-Lee and James Hendler and Ora Lassila, The Semantic Web, *Scientific American*, May 2001.

[4] P. Bonnet, J. E. Gehrke, and P. Seshadri. Querying the Physical World. *IEEE Personal Communications*, 7(5), pp. 10-15, 2000.

[5] Hewlett-Packard Company. CoolTown Home. Available at http://cooltown.hp.com/cooltownhome/index.asp.

[6] F. P. Coyle. *Wireless Web: A Manager's Guide*. Addison-Wesley, 2001.

[7] S. E. Czerwinski, B. Y. Zhao, T. D. Hodes, A. D. Joseph, and R. H. Katz. An Architecture for a Secure Service Discovery Service, In *Proc. of Mobile Computing and Networking*, pp. 24-35, 1999.

[8] A. Gupta, and I. Singh Mumick: Maintenance of Materialized Views: Problems, Techniques, and Applications. *Data Engineering Bulletin*, 18(2), pp. 3-18, 1995.

[9] S. D. Gribble, M. Welsh, J. R. Behren, E. A. Brewer, D. E. Culler, N. Borisov, S. E. Czerwinski, R. Gummadi, J. R. Hill, A. D. Joseph, R. H. Katz, Z. M. Mao, S. Ross, and B. Y. Zhao", The Ninja architecture for robust Internet-scale systems and services, *Computer Networks*, 35(4), pp. 473-497, 2001.

[10] J. Heflin, R. Volz, and J. Dale, eds. Requirements for a Web Ontology Language. *W3C Working Draft*, 08 July 2002.

[11] Sun Microsystems. Jini Network Technology. Available at `http://wwws.sun.com/software/jini/.`

[12] Samuel R. Madden, Robert Szewczyk, Michael J. Franklin and David Culler. Supporting Aggregate Queries Over Ad-Hoc Wireless Sensor Networks. Workshop on Mobile Computing and Systems Applications, 2002

[13] R. Motwani, J. Widom, A. Arasu, B. Babcock, S.

Babu, M. Datar, G. Manku, C. Olston, J. Rosenstein, and R. Varma. Query Processing, Resource Management, and Approximation in a Data Stream Management System. In Proc. of the 2003 Conference on Innovative Data Systems Research (CIDR), January 2003

[14] J. Nakazawa, H. Tokuda. A Pluggable Service-to-Service Communication Mechanism for Home Multimedia Networks, In Proc. of *ACM Multimedia 2002*, December, 2002.

[15] Sanjay Sarma, David Brock, and Daniel Engels. Radio Frequency Identification and the Electronic Product Code. *IEEE MICRO*, pp. 50-54, Nov/Dec 2001.

[16] Universal Plug and Play Forum. Avaiable at http://www.upnp.org/.

[17] M. Weiser. The Computer for the Twenty-First Century, *Scientific American*, pp. 94-10, September 1991.