FIAP: Facility Information Access Protocol for Data-Centric Building Automation Systems

Hideya Ochiai
The University of Tokyo
jo2lxq@hongo.wide.ad.jp

Masahiro Ishiyama
Network Systems Laboratory
R&D Center, Toshiba Corporation

Tsuyoshi Momose
Cisco Systems

Noriaki Fujiwara
Panasonic Electric Works

Kosuke Ito
Ubiteq

Hirohito Inagaki
NTT Cyber Space Laboratories

Akira Nakagawa
NTT Cyber Space Laboratories

Hiroshi Esaki
The University of Tokyo
hiroshi@wide.ad.jp

***Abstract*—Intelligent buildings are getting *data-centric* – they archive the historical records of motion detectors, power usages, HVAC statuses, weather, and any other information in order to improve their control strategies. The engineering cost of installation and maintenance of such systems should be minimized as the system owner has to operate them for several decades: i.e., the lifetime of the building. However, there are several design pitfalls that multiply such engineering costs, which make the operation heavy burden. This paper identifies those pitfalls and presents technical challenges that enable lightweight installation and maintenance. We, then, design facility information access protocol (FIAP) for data-centric building automation systems. We carried out FIAP-based system integration into a building of the University of Tokyo, and demonstrate that FIAP enables incremental installation for wide varieties of applications with small engineering costs.**

***Index Terms*—Intelligent Building, Building Automation, System Design, Implementation**

I. Introduction

The data sequences produced by sensors are historical records, which have not been primarily used for actuation controls in the traditional building automation systems. However, we are getting realized that such historical records are useful for analysing and planning control strategy to make the building more intelligent. By comparing the history of motion detection with HVAC (Heating, Ventilation, and Air Conditioning) working statuses and power usages, we can analyze how people use electricity, find wastes, and make plans for improvement. Intelligent building systems are getting ”data-centric” (Fig. 1) for this purpose.

The widely-acknowledged solution for managing such historical records is to integrate web services, databases and many other technologies into a single building management system. There are various choices in the design of architecture, interfaces and data models that implement the identified requirements and use cases. However, if they are designed in a wrong way, the maintenance of the system becomes burden for several decades: i.e., for the lifetime of the building. Some system components become out-of-order. Requirements and use cases change according to the reconfiguration of floor plans. We must be able to re-assemble the system for these types of accidents and changes with moderate engineering cost.

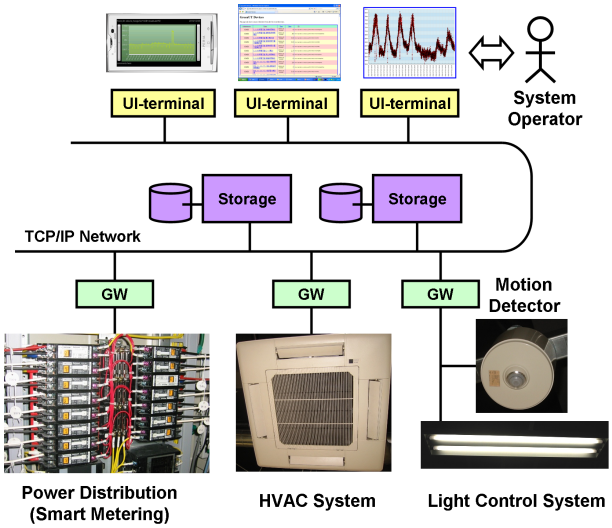

Fig. 1. A data-centric building automation system. Storage servers archive the historical records of power consumption, HVAC status, light control signals, detected motions and any other information. System operator analyzes the habitats, and plans the control strategies to make the building more intelligent.

This paper presents facility information access protocol (FIAP) and demonstrates that FIAP-based system integration avoids design pitfalls that potentially multiply such engineering cost. The design pitfalls, which we identify in this paper, are about (1) definition of data schema, (2) interface design, and (3) data exchange method.

FIAP-based system integration avoids these pitfalls by taking the following design principles: (1) use simple data structure as the common data model, and allow definition of application-specific schema at the system assembly-phase, (2) generalize interfaces into a single interface, and develop system components (e.g., gateways, storages, and user interface terminals) under the common interface, (3) allow scalable data

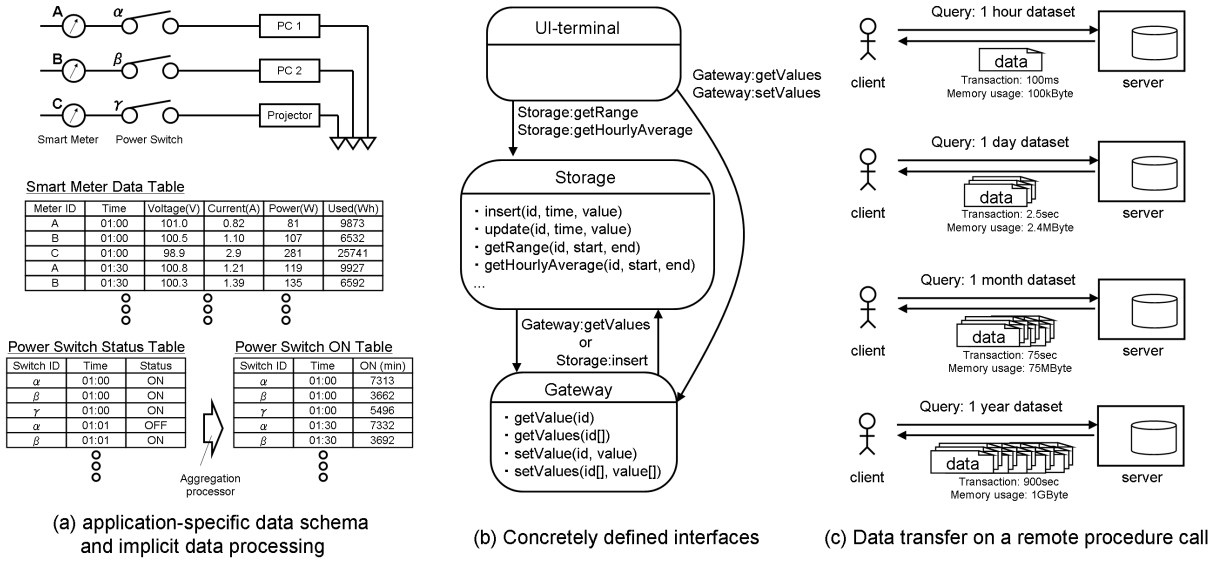

Fig. 2. Pitfalls in the design of data-centric building automation systems. (a) Application-specific data schema and (b) concretely defined interfaces multiply the engineering cost (i.e., installation and maintenance cost) of the system. (c) Data transfer on a remote procedure call causes failure at large dataset transmission.

exchange on remote procedure calls (RPC).

This paper is organized as follows. In section II, we identify the architecture, the pitfalls and the challenges on data-centric building automation systems. In section III, we present the design principles of FIAP. We show FIAP-based system integration in section IV. Section V demonstrates our application. Section VI addresses related works. We conclude this paper in section VII.

II. Data-Centric Building Automation Systems

A. Architecture

Fig. 1 shows typical system architecture for data-centric building automation systems. The gateways translate data between field-level buses and Internet-side storages or user interface(UI) terminals. The field-level buses could be ZigBee[1], Lonworks[2], BACnet[3] , 1-Wire[4] and any other sensor actuator networks. The data storages archive the history of data generated by those sensors and actuators. System operators access such storages and gateways from their UI-terminals and (1) obtain the historical records, (2) obtain the current snapshot and (3) set control schedules.

Some systems calculate statistics of historical records at gateway-side or storage-side. Daily usage of a room can be summarized at the gateway-side from the ON/OFF records of the corresponding motion detector. It can be also summarized at the storage-side.

[1]ZigBee provides tiny nodes for wireless sensor networks. http://www.zigbee.org/

[2]Lonworks: Local operating networks for building automations. http://www.lonmark.org/

[3]BACnet: a data communication protocol for building automation and control networks. http://www.bacnet.org/

[4]1-Wire. http://www.1wire.org/

Intelligent buildings must archive raw data with enough time-granularity. We sometimes need to compare the status of motion detector, door monitor and HVAC. We cannot make such analysis from aggregated trend data (e.g., daily average). This analysis basically involves very large dataset transfer mainly from storages to UI-terminals.

B. Conventional Solution

The widely-applied or conventional solution is to make use of web services and databases, and to integrate them for their identified use cases. Gateways with web service interfaces are available on the market (e.g., BACnetWS[5], oBIX[6], i.Lon SmartServer[7]). Any database management systems (DBMS) can be customized to archive historical records under their identified data schema. Some batch jobs can run behind the database to generate statistics of historical records. Any platforms can be used for developing UI-terminals if it provides the access interface to the database. Integrated and packaged products for building energy management system (BEMS) are also available: e.g., Exaquantum[8].

The problem of this solution comes from its proprietariness. A data-centric building automation system can be developed with system integrators at the first phase. However, we have to maintain and reconfigure the system for several decades after the deployment. Especially if the design falls into the following pitfalls, such maintenance becomes a huge burden for the system owner for several decades.

[5]BACnet web services. http://www.bacnet.org/

[6]oBIX: Open Building Information Exchange. http://www.obix.org/

[7]i.Lon Smart Server: Embedded Internet server provided by Echelon. http://www.echelon.com/

[8]Exaquantum: a packaged product for energy management provided by Yokogawa Corporation

C. Pitfalls

1) Application-specific data schema and implicit data processing: It seems quite natural to define application-specific data schema as a common data model when implementing the use cases. Fig.2 (a) shows the definition of schemas for power and switch monitoring applications. System operator wants to know the power usage and working statuses of the PCs and the projector. This system implicitly calculates the working time in the background, based on the status of the switches.

This system is totally customized for this type of application. We can expand the number of smart meters and switches, but we cannot include weather data, motion detector, HVAC statuses without adding new schemas. Of course, we can do this. However, this approach multiplies the engineering cost. The maintenance of the wide varieties of data schema and related software becomes a burden.

2) Concretely defined interfaces: It seems quite natural to define multiple interfaces and access methods concretely for identified use cases by system integrators as Fig.2 (b). It can surely implement the identified use cases if we could pay a number of attentions in the design, review, software development, system test and so on.

However, if we concretely define many interfaces and develop many subsystems, such engineering cost becomes multiplied. Besides, we cannot easily extend or change the implementation of the system on the concretely defined interfaces. Such burdens follow for several decades.

3) Data transfer on a remote procedure call: Many platforms support remote-procedure call (RPC) as a web service for accessing remote objects and it is getting easier to develop RPC-based communications over the Internet. Data-centric building automation systems have taken this advantage with the development of the web service technologies.

RPC-based communication performs well at small dataset transmission. However, it causes timeout failure or out-of-memory error if the amount of dataset becomes large as Fig.2 (c) illustrates.

D. Challenges

1) Application-Independent Data Model: In order to use the system at many applications, the common data model must be designed independently from application-specific schema. The least requirement on the data model is (1) that it allows managing time-series data sequence and (2) that it allows identifying the sequences. This also omits implicit data processing inside the system (if we need to process data inside, we should explicitly specify the processing scheme).

2) Interface Generalization: In order to simplify the management of interfaces in data-centric building automation systems, we must design a common interface that can interconnect system components. This interface becomes a generalized interface that can implement gateways, storages, UI-terminals and any other functionalities.

3) Scalabile Data Transmission: A data-centric building automation system must be able to transfer very large dataset from a component to another. Though data transmission on a single RPC is not scalable, we must make use of RPC-style communication to get scalability because RPC is well-supported by many platforms.

III. Facility Information Access Protocol

FIAP defines an application-independent data model for managing data sequences produced by sensors and actuators. It generalizes the interface for gateways, storages and UI-terminals. We also designed the data exchange procedure that allows scalable data transmission.

A. Management of Data Sequence by Point

A sensor produces a sequence of data. Signals for controlling an actuator also forms a sequence of data. Generally any entities work in the same manner in data-centric building automation systems. For example, a data aggregator (e.g., hourly-average temperature calculator) generates sequences of data from their sources.

We define *point* to identify those sequences in the system. A point has only a sequence of data, which meaning (e.g., information of sensing target) must not be defined here in the common model. System components such as gateways, storages and UI-terminals manage them by points in their memory space or exchange them with others on flows.

More formally, let P be a set of points managed in a component or transferred on a flow. A point $p \in P$ has a set of time-value pairs, which we denote by $V(p)$. A pair $v \in V(p)$ has time and value. The format looks like as follows.

```
<point id="p1">
 <value time="2011-05-01T00:00:00">35.5</value>
 <value time="2011-05-01T00:01:00">35.4</value>
 <value time="2011-05-01T00:02:00">35.3</value>
</point>
<point id="p2">
 <value time="2011-05-01T00:00:00">true</value>
 <value time="2011-05-01T00:01:00">true</value>
 <value time="2011-05-01T00:02:00">false</value>
</point>
```

This data model itself is designed to be generic. It can be used at power monitoring applications, weather information gathering, and management of virtual machine working statuses.

B. Generalization of Gateways, Storages and UI-terminals

FIAP defines a common interface for gateways, storages, and UI-terminals. They can be inter-connected with other components with the common interface. The differences of components come from their implemented functionalities. A gateway has a data bridge between a field-level bus and FIAP data model. A storage has a huge capacity of buffers to archive the history records of data. An UI-terminal has a data bridge between user interfaces and FIAP data model. Although these components are implemented differently, they can be interconnected by the common interface.

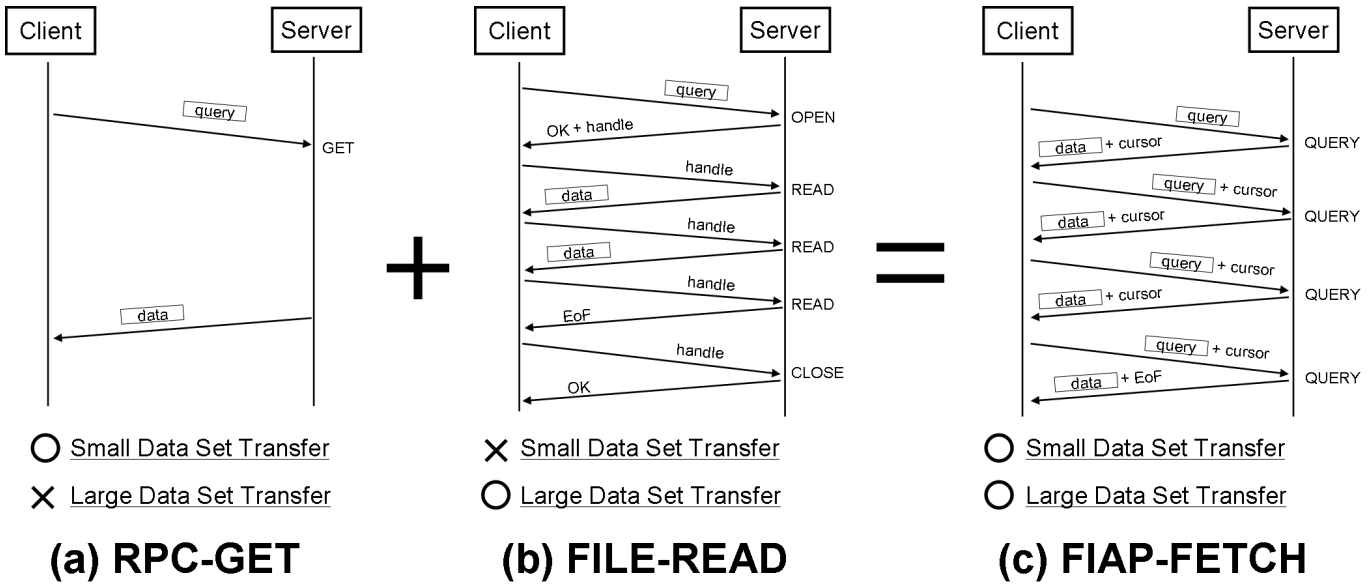

Fig. 3. FIAP-FETCH procedure takes both advantages of RPC-GET and FILE-READ. (a) RPC-GET works efficiently at small data transfer but involves heavy load at large data transfer. (b) FILE-READ enables very large data transmission but has overhead at small data transfer. (c) FIAP-FETCH works as RPC-GET when the size of dataset is small but changes to FILE-READ style when it becomes large.

C. Data Exchange Procedures

FIAP defines three procedures for data exchange among components. These procedures are (1) WRITE to send data to other components, (2) FETCH to retrieve data from other components, (3) TRAP to configure the other components to notify the change of status. FIAP defines them on RPC style communication because RPC is well-supported by many platforms.

1) WRITE: WRITE procedure initiates data transportation at the sender-side. The client (i.e., the sender) sends a message formatted as section III.A. The server (i.e., the receiver) returns "OK" if it accepts. In WRITE procedure, the sender-side can avoid huge data transmission in one procedure call.

2) FETCH: FETCH procedure initiates data transportation at the data receiver-side. The receiver-side sends a query with specifying the range of dataset, but it does not know the amount of data for the returning data. If the amount of dataset is not large, the server returns all of them at the response. However, if the amount exceeds a certain limit, the server returns only the first subset of the specified dataset with a cursor, indicating that there is the next subset. If the client receives a cursor, it requests again with the cursor, and the server returns the next subset. They repeat this procedure until all the data transfer finishes.

Fig. 3 shows that FETCH procedure takes both advantages of RPC-GET and FILE-READ. RPC-GET returns all the dataset in one procedure call. It has smaller overhead in communication but it puts heavy load at the server if the amount is large. FILE-READ is scalable for reading very large dataset, however, it has overhead for small dataset transfer. FIAP-FETCH procedure performs well for both small dataset and large dataset.

3) TRAP: FIAP defines TRAP procedure to dynamically subscribe event-like data sequences from other components. A subscriber sends a stream (or trap) query to a notifier. Then, the notifier sends updated data to the subscriber. The stream query has a lifetime, and it must be updated by the subscriber.

IV. FIAP-based System Integration

In FIAP-based system integration, we assemble individually developed components (i.e., gateways, storages and UI-terminals) into a data-centric building automation system. We configure them to properly exchange data with each other so that these components can collaboratively work. The system becomes a diagram of component and flow as Fig. 4. Thus, we call this assemble process *component-flow programming*.

Formally, let $\Psi = (C, F)$ be a component-flow system. C is a set of components and F is a set of flows between components. A component $c(\in C)$ implements functionalities such as gateways, storages and UI-terminals. A flow $f(\in F)$ specifies how and what data shall be exchanged between the components.

For example, Fig. 4 shows a data-centric building automation system. It has four components $C = \{c_1, \ldots, c_4\}$, and these components are connected by flows $F = \{f_1, \ldots, f_4\}$.

4) Programming of Components: Each component is programmed as follows.

- c_1: p_1 gives the status of motion detector by *true/false*. p_2 gives the summarized time of motion detection in second. p_3 gives the status of the lamp by *true/false*. The lamp can be controlled at p_4 by *true/false*.
- c_2: p_5 gives the summarized power usage in Wh.
- c_3: c_3 archives the data of any points written to this component, and returns the specified range of dataset.

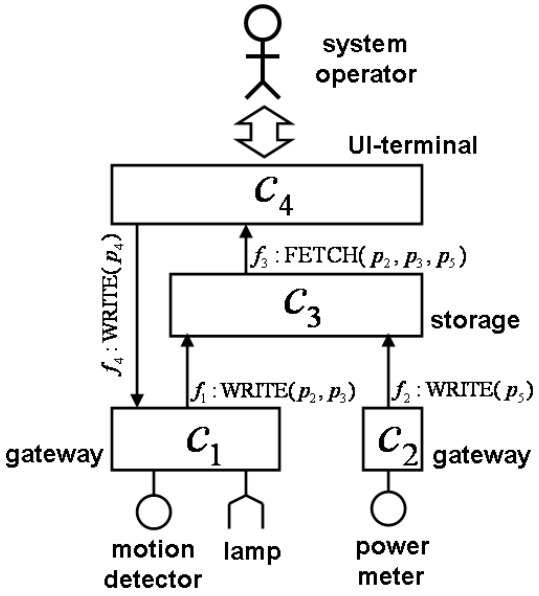

Fig. 4. FIAP-based system integration. We assemble individually developed gateways, storages, and UI-terminals into a data-centric building automation system, which can be drawn by component-flow diagram.

c_4: c_4 shows one day history of p_2, p_3 and p_5. It creates lamp control commands at p_4.

5) Programming of Flows: Each flow is programmed as follows.

f_1: c_1 sends the latest values of p_2 and p_3 periodically to c_3 by WRITE procedure.

f_2: c_2 sends the latest value of p_5 periodically to c_3 by WRITE procedure.

f_3: c_4 retrieves one day history of p_2, p_3 and p_5 from c_3 by FETCH procedure when requested by the system operator.

f_4: c_4 sends p_4 to c_1 by WRITE procedure when requested by the system operator.

In this way, the specification of $\Psi = (C, F)$ determines the total behavior of the system, which works as a data-centric building automation system.

Practically, these programming can be made by script-based configuration. We can make use of text editors or command line interfaces (CLIs) for configuring the components and flows just as the conventional Ethernet switches and IP routers. We do not have to prepare integrated development environments (e.g., Eclipse, Microsoft Visual Studio) for this purpose.

V. Application

A. Implementation and Deployment

We have been operating a FIAP system for about one year from the beginning of 2010 at Engineering Bldg.2 in the University of Tokyo. It manages 1714 points[9], which in detail are:

- Electricity at distribution boards (908 points)
- Electricity at outlets (67 points)
- HVAC working modes (639 points)
- Motion detection and light status (40 points)
- Room environment (36 points)
- Gas and water supply (17 points)
- Weather information (7 points)

The frequency of data recording is configured for each point. Some points record at every minute, but others at every 30 minute. The total number of data elements in the data storage for year 2010 was about 430 million records.

1) Benefit from application-independent data schema: We first implemented storage and UI-terminals, and deployed for the building. We, then, implemented gateways for BACnet, oBIX, SNMP and other proprietary systems, and incrementally applied to the multiple applications as above: i.e., electricity, HVAC, motion detection, light status, weather. We could include various applications because FIAP has taken application-independent data schema for the design principle.

2) Benefit from interface generalization: We had only to assemble individually developed components into the data-centric building automation system. Actually, we have implemented gateways, storages, and UI-terminals on many platforms, including Java (with Axis2[10]), PHP, Microsoft .NET Framework, Python, Ruby and Linux C. Those implementations came from different vendors and developers, and independently packaged as FIAP components. What we had done for integration is script-based configuration with a text editor.

3) Lightweight implementation: Application-independent data schema and interface generalization has certainly reduced the engineering cost for software design, implementation, test and so on. The source code for Java platforms is made only by about 9000 lines with 52 class files (not including auto-generated codes). This includes the implementation of gateway (for BACnet), gateway (for oBIX), gateway (for SNMP), storage, and several UI-terminals. They use external libraries of Axis2 and JDBC[11]. If we implement more functionalities, the size of source code will increase, but our prototype indicates that FIAP is basically lightweight.

The most lightweight implementation is programmed in C for Linux platform. Though it has only WRITE client functionality, it is implemented with only 264 lines without using XML and SOAP libraries.

B. Data Analysis

Fig. 5 is the distributions of temperature of a meeting room in the building. The room has a motion detector, which can identify the time in use. Using such information, we categorized temperature data into four classes: i.e., (1) summer (in use), (2) summer when not used (vacant), (3) winter (in use), and (4) winter (vacant). We used dataset for [2010-07-01, 2010-09-30] as summer, and for [2010-01-01, 2010-02-28] $\cup$ [2010-12-01, 2010-12-31] as winter.

[9]The number of points is the number of independent data sequences. E.g., The three sequences for voltage(V), current(A) and power(W) of an outlet are counted as three.

[10]Axis2: apache web services engine.

[11]JDBC: Java data base connectivity.

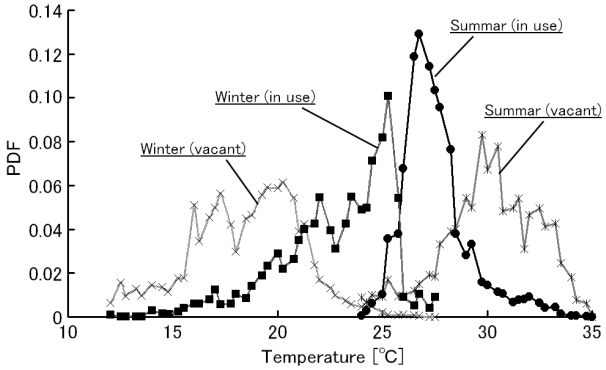

Fig. 5. Temperature distributions of a meeting room

From this result, we can see that temperature of the meeting room has been well controlled. When the meeting room was in use, the average temperature was 27.5 degrees Celsius in summer, and 22.8 degrees in winter. They did not become too low in summer or too hot in winter. This means that people set appropriate temperature for the room. When the meeting room was vacant, the temperature became higher in summer and colder in winter. This result indicates that if no body used the room, the air controller was switched off.

In order to enable this analysis, the analyzer had to read 3 months' history twice (for winter and summer) for 7 sensors: 6 sensors for motion detector, 1 sensor for temperature. Totally, there were about 1.5 million records. The FETCH procedure worked very well for retrieving such large amount of dataset.

VI. Related Work

Researchers from database communities have tried to develop wireless sensor networks [9], [10], [6] with the application of database management systems (DBMS) or data stream management systems (DSMS) [5], [1]. GSN[3] and Daniel J. Abadi et al.[2] has also studied DBMS-based or DSMS-based sensor networking over the Internet. The main focus in their research has been the reduction of data traffic by in-network data aggregation[8], and the engineering cost was not mainly focused.

From the system development point of view, DBMS-based or DSMS-based platform is too general. The system integrator has to identify the use cases and design the data schema. They also develop the software for the designed data schema, and need to maintain the software. As we have identified, this system integration manner falls into the pitfall as Fig. 2(a), indicating that it multiplies the engineering cost for both installation and maintenance.

The researches of sensor web [4] have challenged to allow the management of global environmental data such as weather information over the Internet. The main goal of their researches seems to allow searching application-specific statuses from shared "pre-defined environmental information". Thus, the data model presented in IrisNet[7] is designed to tell whether parking slot is available or not. The main goal of our research is to propose a system development method which minimizes engineering cost at the system assembly phase. Thus, we considered application-independent data model and interface generalization for the design of individual system components.

VII. Conclusion

We identified three major design pitfalls that multiply the engineering cost (i.e., installation and maintenance cost) of data-centric building automation systems. System owners operate their intelligent buildings for several decades. Thus, such engineering cost must be minimized, otherwise the operation becomes a burden.

We presented technical challenges that enable lightweight system integration. The challenges include application independent data model, interface generalization, and scalable data transmission.

We presented the design principles of FIAP and FIAP-based system integration. We have also developed and deployed FIAP-based data-centric building automation system in Engineering Bldg.2 in the University of Tokyo. This experiment has shown that FIAP allows incremental installation for wide-varieties of applications (i.e., electricity, HVAC, motion detection/lights, room environment, gas and water supply, weather) with CLI-based configuration.

Acknowledgement

This paper contains one of the results from the project "Promotion of standardization of integrated network management system", which is organized by Ministry of Internal Affairs and Communications, Japan.

References

[1] D. J. Abadi, Y. Ahmad, M. Balazinska, U. Cetintemel, M. Cherniack, J.-H. Hwang, W. Lindner, A. S. Maskey, A. Rasin, E. Ryvkina, N. Tatbul, Y. Xing, and S. Zdonik. The design of the borealis stream processing engine. In *CIDR*, 2005.

[2] D. J. Abadi, W. Lindner, S. Madden, and J. Schuler. An integration framework for sensor networks and data stream management systems. In *30th VLDB Conference*, pages 1361–1364, 2004.

[3] K. Aberer, M. Hauswirth, and A. Salehi. Infrastructure for data processing in large-scale interconnected sensor networks. In *IEEE MDM 2007*, may 2007.

[4] M. Balazinska, A. Deshpande, M. J. Franklin, P. B. Gibbons, J. Gray, S. Nath, M. Hansen, M. Liebhold, A. Szalay, and V. Tao. Data management in the worldwide sensor web. *IEEE Pervasive Computing*, 6(2):30–40, apr 2007.

[5] M. Cherniack, H. Balakrishnan, M. Balazinska, D. Carney, U. Cetintemel, Y. Xing, and S. Zdonik. Scalable distributed stream processing. In *CIDR*, 2003.

[6] J. Gehrke and S. Madden. Query processing in sensor networks. *IEEE Pervasive Computing*, 3:46–55, 2004.

[7] P. B. Gibbons, B. Karp, Y. Ke, S. Nath, and S. Seshan. Irisnet: an architecture for a world wide sensor web. *IEEE Pervasive Computing*, 2(4):22–33, dec 2003.

[8] B. Krishnamachari, D. Estrin, and S. Wicker. The impact of data aggregation in wireless sensor networks. In *IEEE Distributed Computing Systems*, 2002.

[9] S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. TinyDB: an acuisitional query processing system for sensor networks. *ACM Transactions on Database Systems*, 30(1):122–173, mar 2005.

[10] Y. Yao and J. Ghrke. Query processing for sensor networks. In *CIDR*, 2003.