5-2010

# A Scalable and Energy-Efficient Context Monitoring Framework for Mobile Personal Sensor Networks

Seungwoo KANG

Jinwon LEE

Hyukjae JANG

Youngki LEE
*Singapore Management University*, YOUNGKILEE@smu.edu.sg

Souneil PARK

*See next page for additional authors*



## Citation

KANG, Seungwoo; LEE, Jinwon; JANG, Hyukjae; LEE, Youngki; PARK, Souneil; and SONG, Junehwa. A Scalable and Energy-Efficient Context Monitoring Framework for Mobile Personal Sensor Networks. (2010). *IEEE Transactions on Mobile Computing*. 9, (5), 686-702.
**Available at:** https://ink.library.smu.edu.sg/sis_research/2071

# A Scalable and Energy-Efficient Context Monitoring Framework for Mobile Personal Sensor Networks

Seungwoo Kang, Jinwon Lee, Hyukjae Jang, Youngki Lee,
Souneil Park, and Junehwa Song, *Member*, *IEEE*

**Abstract**—The key feature of many emerging pervasive computing applications is to proactively provide services to mobile individuals. One major challenge in providing users with proactive services lies in continuously monitoring users' context based on numerous sensors in their PAN/BAN environments. The context monitoring in such environments imposes heavy workloads on mobile devices and sensor nodes with limited computing and battery power. We present SeeMon, a scalable and energy-efficient context monitoring framework for sensor-rich, resource-limited mobile environments. Running on a personal mobile device, SeeMon effectively performs context monitoring involving numerous sensors and applications. On top of SeeMon, multiple applications on the mobile device can proactively understand users' contexts and react appropriately. This paper proposes a novel context monitoring approach that provides efficient processing and sensor control mechanisms. We implement and test a prototype system on two mobile devices: a UMPC and a wearable device with a diverse set of sensors. Example applications are also developed based on the implemented system. Experimental results show that SeeMon achieves a high level of scalability and energy efficiency.

**Index Terms**—Context monitoring, shared and incremental processing, sensor control, energy efficiency, personal computing, portable devices, ubiquitous computing, wireless sensor network, pervasive computing.

## 1 Introduction

Proactively providing services to mobile users is essential for many emerging pervasive computing applications. Provision of situation-specific services without user intervention requires an involved process for acquiring users' contexts. Such services require different types of contexts with different degrees of context awareness. Individual users have different service requirements and preferences, personalized to their own needs. Increasingly, a number of wearable and wireless sensors with diverse capabilities are being densely deployed on users' bodies or in their personal areas. To provide much broader coverage and higher accuracy in recognized contexts, personal sensor networks will grow much in scale, diversity, and complexity. In such environments, the mobile device plays a key role as a full-fledged, integrated personal service agent, incorporating personal sensor networks and running multiple applications simultaneously. An effective personal mobile system must continuously process a large volume of sensor data while supporting a number of applications.

In this paper, we propose SeeMon, a scalable and energy-efficient context monitoring framework for personal context-aware applications. To provide proactive services to mobile users, these applications should continuously monitor users' contexts and capture their changes over time. A major challenge results from the key characteristics of sensor-rich and resource-limited mobile environments. In these environments, users carry a personal mobile device with a number of wireless sensor nodes in the BAN/PAN. Context monitoring in such environments imposes heavy workloads on a mobile device and sensor nodes. First, a high rate of data streams from numerous sensors should be collected and processed in the device. Data processing often involves complex operations such as feature extraction and context recognition. Second, a number of monitoring requests from many applications should be handled; the requests will be long-running, which requires continuous processing on the device. Finally and most importantly, with such workloads, the resource limitations of the device and sensors should be carefully considered, especially their battery power and processing capacity.

The proposed framework addresses these challenges in two main ways. First, context monitoring in SeeMon focuses on continuous detection of context changes. Note that this semantics is different from conventional context recognition, which identifies the current context only. Once a change is identified, it is not necessary to redundantly recognize the same context and send notification updates as long as the context remains unchanged.

Second, while conventional context processing occurs in a unidirectional fashion, SeeMon approaches the context monitoring problem in a bidirectional way. In the unidirectional approach described in Fig. 1, the processing flow proceeds in one direction through a pipeline, which consists of several stages, i.e., preprocessing, feature extraction, context recognition, and change detection. Change detection is performed at the last stage of the pipeline. To detect context changes, data should be collected from sensors and processed for recognition continuously. Moreover, the results of monitoring queries must be reevaluated based on the recognized context. This continuous process results in heavy

• *The authors are with the Department of Computer Science, KAIST, 373-1, Guseong-dong, Yuseong-gu, Daejeon 305-701, Republic of Korea. E-mail: {swkang, jcircle, hjjang, youngki, spark, junesong}@nclab.kaist.ac.kr.*

*Manuscript received 18 Dec. 2008; revised 5 June 2009; accepted 4 Aug. 2009; published online 13 Aug. 2009.*
*For information on obtaining reprints of this article, please send e-mail to: tmc@computer.org, and reference IEEECS Log Number TMC-2008-12-0502. Digital Object Identifier no. 10.1109/TMC.2009.154.*

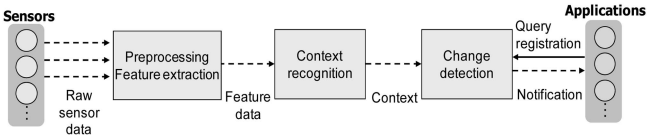



Fig. 1. Unidirectional approach in conventional context monitoring.

processing and high energy costs. However, the bidirectional approach in Fig. 2 forms a feedback path from the applications to sensors. This approach gives the opportunity to achieve a high degree of efficiency in computation and energy consumption. Such an advantage results from careful reflection of the high-level application requirements such as monitoring queries and the low-level status of sensor resources. This makes it possible to elaborate on the computational stages in the processing pipeline, and hence, to make a monitoring decision at an earlier stage, significantly saving computational overhead. As shown in Fig. 2, in our approach, a context change is detected directly from the feature data without going through the expensive context recognition stage as in Fig. 1. This is enabled by query translation that transforms a *context-level query* into a *feature data-level query*. The translation is performed only once whereas the savings in computational or energy cost are constantly achieved throughout successive monitoring operations. In addition, the low-level status of sensor resources can be dynamically analyzed considering the requirements of the monitoring queries. Thus, sensors necessary for context monitoring can be intelligently identified and controlled to save energy or increase utilization.

There has been much research on middleware for context-aware applications [9], [10], [11], [12], [13], [14], [15], [16]. Middleware provides basic functionalities required for context awareness, i.e., sensor data collection, data preprocessing, feature extraction, and context recognition. Different works study different aspects (e.g., different ways of modeling context, different ways of reasoning or inference to attain higher level context, and additional functionalities such as security or privacy). The research accumulates important results to realize context-aware services. However, the focus of previous research was mainly on providing context awareness; they are limited in terms of monitoring, especially issues concerning monitoring in sensor-rich, resource-limited environments. Most importantly, they approach the problem in a unidirectional way, resulting in heavy processing and high energy costs.

To the best of our knowledge, our work is the first attempt to present a scalable and energy-efficient context monitoring framework for mobile devices. Running on mobile devices, SeeMon effectively performs context monitoring involving numerous sensors and applications. On top of SeeMon, multiple applications simultaneously operating on the devices can understand the context of users and serve them appropriately.

### 1.1 Bidirectional Approach to Context Monitoring Problem

Our approach is to effectively remove unnecessary expensive computation and communication in the context monitoring process. We look into the context monitoring process shown in Fig. 1 and develop the proposed framework based on three observations.

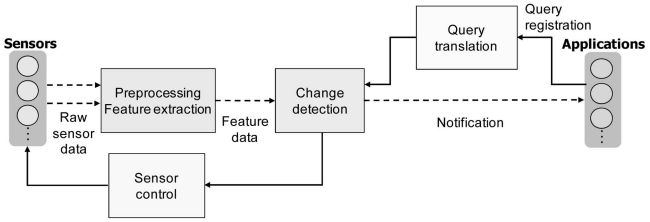



Fig. 2. Bidirectional approach to context monitoring problem.

First, we observe that it is computationally efficient if change of context can be identified at an early stage of the processing pipeline. The conventional way to detect a change of context is to compare contexts after inferring them via an algorithm like decision tree logic. However, we can avoid such costly operation when we translate a high-level application query into a lower level query. For example, we can skip the costly decision tree logic if we detect the change of activity using feature value changes from accelerometers. As far as we know, our work is the first attempt to exploit this novel observation for context change detection.

Second, we observe and exploit context continuity. This is possible because we continuously capture context to notice its changes. It is not just a single recognition task. Rather, it is a sequence of successive tasks, which should be performed continuously. From this perspective, we note that the context of an individual remains the same for a certain amount of time. This continuity of context can be understood in two levels: the context level as well as the source or feature data level. Consecutive readings from a data source change gradually and these small changes rarely lead to changes in context.

Based on the locality of the feature data, we greatly reduce the processing cost of the change detection process. Among numerous data updates, we effectively sort out the updates that are expected to result in context changes. Then, only a small number of registered queries relevant to the updates are quickly searched for and evaluated. Combined with the mechanism for feature data-level change detection described above, we achieve a high level of performance.

Third, a small subset of sensors is often sufficient to answer queries. For example, consider a query for the context "studying in the library." When the user is not in the library, her activity information is not useful; the query can be answered using only location information. However, even for such a simple query, finding the most efficient subset of sensors to activate is complex since it may involve numerous queries and many possible sensors. We develop a novel method for computing a reduced set of sensors that is sufficient for context monitoring and then only activate this subset. These techniques reduce the amount of wireless communication between sensors and a mobile device, leading to energy savings.

Based on these observations, we develop three methods for context monitoring: Context Monitoring Query (CMQ) translation, shared and incremental CMQ evaluation, and Essential Sensor Set (ESS) selection. Our framework automatically translates CMQs issued by applications into queries with feature data-level monitoring conditions. While the translation is performed only once for each query, the performance benefit is achieved constantly throughout the entire query lifetime. The shared and incremental CMQ evaluation method maximally utilizes the context continuity. By exploiting the locality of feature data, the method

significantly accelerates successive evaluation of numerous CMQs. Further, it only maintains compact lightweight data structures carefully designed. The method thereby achieves a high level of scalability even in a resource-limited environment. The framework is also successful in energy saving by computing the ESS and dynamically controlling sensors based on it. We show the complexity of ESS selection by proving that the problem is NP-complete. A practical heuristic algorithm with acceptable approximation ratio is developed to handle the ESS selection problem. Also, we develop ESS calculation policies, which can be alternatively used to cope with various environments and operational situations. Finally, we devise and examine two sensor control modes to evaluate the effectiveness of sensor control in terms of energy consumption.

### 1.2 Implementation and Evaluation

We have implemented a SeeMon prototype with core components for scalable and energy-efficient context monitoring. We have also built two pervasive computing applications that use the SeeMon prototype for context monitoring. In order to examine heterogeneous mobile environments, we have deployed and tested the prototype system on various types of mobile devices along with diverse sensors.

Experimental results show that SeeMon can achieve a high level of scalability and energy efficiency in sensor-rich and resource-limited mobile environments. SeeMon provides 4.6 times better throughput than an alternative context monitoring method under a workload of about 2,100 data samples per second. Also, SeeMon reduces a large number of wireless data transmissions, e.g., between about 50 and 90 percent, on average, while evaluating 4,000 CMQs under different conditions.

The rest of the paper is organized as follows: Section 2 discusses related work. Section 3 presents the overview of SeeMon framework. We describe the proposed processing-efficient CMQ evaluation method in Section 4 and the energy-efficient sensor control method in Section 5. Section 6 presents our prototype implementation and experiences on example applications. Section 7 shows experimental results. Finally, Section 8 presents discussion and future work and Section 9 concludes the paper.

## 2 Related Work

Context-aware applications and application-specific systems have been proposed in several application domains including healthcare and medical applications [4], [5], reminder applications [6], and activity recognition [7], [8]. Each system mainly utilizes an application-specific context such as location, activity, or biomedical information. However, the proposed framework is designed to support multiple applications, which utilize diverse contexts generated from numerous sensors in the BAN/PAN. Thus, the framework provides intuitive query interface to specify contexts of interest and corresponding processing mechanisms.

Some existing projects have proposed middleware to support context-aware applications. Their aim is to hide the complicated issues related to context awareness. Most middlewares are designed to run in a centralized server environment [9], [10], [11] or a distributed environment [12], [13]. This approach requires infrastructural support to deal with sensor data collection and context processing. Moreover, privacy issues can arise since context information of individual users is exposed to the server. Some context-aware middlewares target mobile devices [14], [15], [16], but do not consider tens/hundreds of BAN/PAN sensors and the processing and power limitations of mobile devices and sensors. Moreover, they do not focus on continuously detecting the changes of context.

Limited battery power has been a critical problem in the field of mobile computing. Many techniques have been proposed to improve the energy efficiency of mobile devices by reducing the wireless communication cost. They include a technique to delay the communication based on GPS-based movement prediction [20] and techniques to reduce the Wi-Fi connection establishment and maintenance cost based on a low-power radio interface [17], a Wi-Fi detector [18], or Wi-Fi network condition estimation [19]. SeeMon also enhances energy efficiency by reducing wireless communication. However, our approach utilizes the characteristics of personal context and applications' requirement for context monitoring.

Energy saving in wireless sensor networks is well studied, including MIMO systems at the physical layer [21], MAC protocols [22], routing mechanisms [23], and integrated solutions optimizing the energy consumption of all radio states [26]. SeeMon operates at the application layer and is complementary to these approaches.

A wearable activity recognition system considering the energy consumption of sensors has been proposed [43]. It utilizes the fact that required accuracy and granularity of recognition are different according to applications. They show that a subset of sensors suffices the coarse-grained recognition with desirable accuracy. SeeMon identifies and uses a reduced set of sensors, considering the type of context (e.g., location or activity). More importantly, unlike SeeMon, the system does not cover diverse context types or provide a general platform for multiple applications. However, it is complementary to utilize the trade-off between the accuracy of context and the energy consumption.

Our work on processing-efficient CMQ evaluation is broadly related to continuous query processing in Data Stream Management Systems [38], [39], [40]. These systems support monitoring query semantics over continuously streaming data and efficient processing mechanisms for continuous queries [40], [27]. However, such methods are not directly applicable to the context monitoring problem because they are not designed for efficient detection of changes in data values. Instead, they support continuous query evaluation to retrieve all matching data values. SeeMon adopts an efficient solution to detect context changes in terms of computation cost and memory consumption, which are especially critical in resource-limited mobile environments.

MyExperience [41] has been proposed to collect quantitative and qualitative usage data on personal mobile devices for studies of mobile technology usage and evaluation. For efficient data collection, it employs an efficient event-driven architecture of Sensors, Triggers, and Actions. Although the event-driven architecture is similar to SeeMon, SeeMon focuses on real-time context monitoring rather than the collection of usage data. In particular, SeeMon addresses the problem of sensor data processing in sensor-rich and resource-limited environments.

## 3 Context Monitoring Framework Overview

### 3.1 Motivating Environment

The rapid advance of mobile device and service technologies will lead to a new mobile environment in which

*personal sensor networks* as well as *personal context-aware applications* will grow in scale, diversity, and complexity.

Diverse sensors and sensor networks are increasingly being deployed in personal areas and on human bodies. For example, acceleration sensors, biomedical sensors (e.g., ECG, BVP, GSR, and EMG sensors), and environment sensors (e.g., temperature, humidity, light sensors, RFIDs, and GPS) are widely deployed across many domains. Even for a single sensor type, tens of sensors are sometimes used for accurate context recognition [1]. In practice, it is a key trend that a variety of sensors such as accelerometer and gyroscope is equipped inside the recent smart phones. At the current rate of advancement, future personal sensor networks will likely incorporate up to hundreds of sensors of various types.

At the same time, many new personal context-aware applications are being developed and deployed based on personal sensor networks. Emerging sensor types will lead to even more applications for mobile users. These applications will be deployed in domains such as healthcare [48], personal assistance, dietary monitoring [2], interactive art [3], gaming, and education.

An important characteristic of these applications is that they monitor individuals' context and surroundings. In the future, these applications will require even finer grained monitoring. For example, a current personal assistant service requires understanding the user's activity such as running, walking, or sitting, which is recognized using several accelerometers. However, in the near future, applications may need to understand and reflect finer movements such as delicate hand motions and even individual fingers' movements. This will require crafted placement of an increasing number of sensors and processing of much more monitoring requests. Most important, while personal applications expand in quantity and quality, users will not use separate hardware devices for each application. They will use a single mobile device as a full-fledged, integrated personal service agent and simultaneously run multiple applications on the device. In addition, the context monitoring requests from the applications will be long-standing, resulting in continuous operation of the mobile device, possibly for 24 hours per day 7 days per week. As a result, as an integrated personal service agent, personal mobile devices continuously process a high number of context monitoring requests as well as voluminous data from numerous sensor devices. This introduces new technical obstacles for future pervasive services, which will be compounded by the resource limitations and heterogeneity of the sensors and mobile devices.

### 3.2 Context Monitoring Query

SeeMon provides CMQ, an intuitive monitoring query language that supports rich semantics for monitoring a wide range of contexts. It is important for applications to catch the changes in users' context proactively. Applications do not necessarily know what the current context is, but must detect when the changes occur. CMQ is devised to support such monitoring semantics. The CMQ template has the following format:

```
Context <context element>
        (AND <context element>)*
ALARM <type>
DURATION <duration>.
```

A CMQ specifies three conditions: *context*, *alarm*, and *duration* conditions. First, the context condition describes the context of interest. It is presented as a Conjunctive Normal Form (CNF) of multiple *context elements.* Each context element is described by a specific context type, an operator, and a context value. SeeMon supports two types of operators: equality ( $==, !=$ ) and inequality ( $<, \leq, >, \geq$ ) operators. The state of the context condition becomes true if and only if all context elements are true. Context conditions containing negation ($\neg$) and OR operations can easily be supported in SeeMon. By using Boolean algebra, such context conditions are transformed into CNF containing only AND operation.

Second, the alarm condition determines when SeeMon delivers an alarm event to applications. Currently, SeeMon supports two types of alarm conditions: an *instant transition* alarm and a *timed transition* alarm. First, the instant transition alarm specifies SeeMon to give an alarm event right after the state of the context condition changes from false to true ($\mathrm{F} \rightarrow \mathrm{T}$) or from true to false ($\mathrm{T} \rightarrow \mathrm{F}$). Second, the timed transition alarm specifies SeeMon to deliver an alarm event when the state of the context condition continues to be true for a period of time and changes to false ($\mathrm{T}\ (period) \rightarrow \mathrm{F}$) or it does false for a period of time and changes to true ($\mathrm{F}\ (period) \rightarrow \mathrm{T}$). The period can be specified with a single value or a range of values. If a single value is given for the period, the alarm condition is satisfied when state continuation time is larger than the value. On the other hand, if a value range is given, the alarm condition is satisfied when state continuation time is within the range.

Finally, the duration condition specifies how long a registered CMQ should run. SeeMon maintains a CMQ for the specified duration as long as an application does not deregister the query.

The following is an example CMQ. As shown in the example, the context monitoring semantics required for applications can be easily expressed by a simple CMQ:

```
Context (location == Library)
        AND (activity == Sleeping)
        AND (time == Evening)
ALARM → T
DURATION 120 DAYS.
```

### 3.3 Architecture

SeeMon is a middle-tier framework between personal context-aware applications and a personal sensor network (see Fig. 3). SeeMon provides programming APIs and a runtime environment for applications. Multiple applications that require context monitoring can be developed through the APIs and can run on top of SeeMon concurrently. Meanwhile, SeeMon receives and processes sensor data and controls the sensors in the personal sensor network. For the wireless communication between them, protocols such as Bluetooth and ZigBee can be used. In addition to wireless personal sensor network, device-attached sensors such as accelerometer and gyroscope deployed on smart phones can be easily incorporated in the SeeMon framework without architectural change.

SeeMon consists of four components: the *CMQ Processor*, the *Sensor Manager*, the *Application Broker*, and the *Sensor Broker*. Based on these components, the operation of SeeMon is performed in three phases: query registration, query processing, and sensor control. First, applications initiate context monitoring by registering CMQs to the CMQ

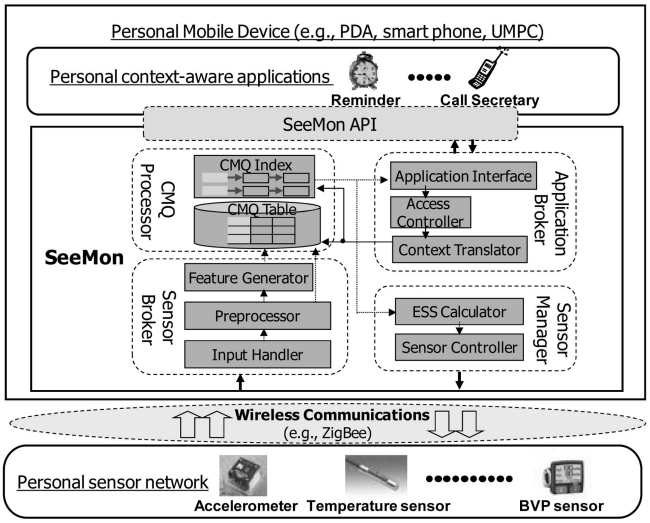



Fig. 3. Architecture of SeeMon.

Processor through the Application Broker. Then, the CMQ Processor performs scalable context monitoring by efficiently evaluating numerous CMQs over data delivered by the Sensor Broker; monitoring results are then forwarded to applications. Finally, the Sensor Manager finds a minimal set of sensors that is necessary to evaluate all registered CMQs. Then, the Sensor Manager forces unnecessary sensors to stop transmitting data to SeeMon, thereby saving energy.

The Application Broker consists of the *Application Interface*, the *Access Controller*, and the *Context Translator*. First, the Application Interface provides an interface to applications. Table 1 summarizes the APIs provided by SeeMon. The Access Controller manages privacy and security parameters in SeeMon. Since remote applications can request context monitoring, it is important to provide an appropriate access control mechanism. Currently, the Access Controller checks whether a requesting application is registered in an access control list [37]. The Context Translator translates a CMQ issued by a permitted application into a feature data-level CMQ. The translated data-level CMQ is registered with the CMQ Processor.

The CMQ Processor consists of the *CMQ-Table* and the *CMQ-Index*. The CMQ-Table stores registered CMQs and their evaluation results. Through the CMQ-Index, context elements for each feature data can be quickly evaluated. The evaluation of a CMQ is triggered by state changes in context elements of the CMQ. When the CMQ Processor detects that a certain CMQ is satisfied, an alarm event is promptly forwarded to corresponding applications.

The Sensor Broker consists of the *Input Handler*, the *Preprocessor*, and the *Feature Generator*. The Input Handler manages communication with sensors and receives sensor data. The Preprocessor removes noise and error from input data and performs simple computation such as data format conversion. The Feature Generator performs complex computation on data from the Preprocessor, such as Fast Fourier Transform, to derive feature data. It then inputs derived feature data into the CMQ Processor.

The Sensor Manager consists of the *ESS Calculator* and the *Sensor Controller*. The ESS Calculator computes an ESS necessary to evaluate CMQs and identifies unnecessary sensors based on the evaluation results of the CMQ Processor. Based on the calculated ESS, the Sensor Controller sends selected sensors control messages to reconfigure the sensors to stop transmitting data.

TABLE 1
SeeMon API

| Functionality | API List |
|---|---|
| Context Monitoring | registerCMQ (*CMQ_statement*) |
| | deregisterCMQ (*CMQ_ID*) |
| Context Customization | createMAP (*[Parent_Map_ID]*) |
| | deleteMAP (*Map_ID*) |
| | insertContextElement (*[Map_ID,] context_level_semantic, data_level_semantic*) |
| | deleteContextElement (*[Map_ID,] context_level_semantic*) |
| | updateContextElement (*[Map_ID,] context_level_semantic, data_level_semantic*) |
| Context Browsing | browseMAP () |
| | browseContextElement (*Map_ID [, context_level_semantic]*) |

## 4 Processing-Efficient CMQ Evaluation

Multiple applications running on SeeMon will be interested in different contexts. Thus, the CMQ Processor should handle a large number of CMQs issued by applications. To notify the changes of context immediately, CMQs must be continuously evaluated over data streams from the sensors. It is costly to evaluate all CMQs upon every data arrival. Furthermore, dealing with such voluminous data streams must be done in a resource-limited environment. SeeMon employs novel methods to significantly improve the evaluation performance under such query and data workloads.

SeeMon avoids the expensive context recognition process such as decision tree traversal by translating CMQs into feature data-level queries. The CMQ translation provides a chance to reduce the processing overhead by pruning out unnecessary context recognition at an early stage of the processing. SeeMon develops a shared and incremental processing method to efficiently process the translated feature data-level queries in the CMQ Processor.

The shared processing method efficiently processes a large number of data-level CMQs using a query index called the CMQ-Index. Once the index is built for all registered CMQs, upon a data arrival, only relevant queries will be searched for. This method provides significant performance benefit compared to CMQ evaluation without shared processing.

The key idea behind our incremental processing method is to utilize the locality of feature data streams and develop a stateful query index for incremental evaluation. Consecutive updates from a data stream usually show gradual changes. Thus, in many cases, consecutive updates from each sensor do not change the states of registered queries. For example, consider a query to monitor an energy feature value stream from an accelerometer with a range [70 < energy < 75]. If the energy feature values are [72, 71, 73, 74], the state of this query is true and it remains unchanged. Even if data updates incur state changes, it is highly possible that the changes will be restricted to a small number of queries that are interested in nearby ranges. The CMQ-Index exploits such locality and consequent overlaps between previous and current state evaluation results by remembering the previous states of all queries. Furthermore, it precomputes the queries whose states change at each value range. The CMQ-Index also partitions the

Meta-information
Map_ID: 14 Parent_Map_ID: 3
Map_Type: instance Application_ID: 14

| Context-level semantics | | Data-level semantics | | | |
|---|---|---|---|---|---|
| Type | Value | Feature# | ID | Low | High |
| location | Playground | 1 | longitude | 36°22'04.28" | 36°22'04.60" |
| | | 2 | latitude | 127°21'56.60" | 127°21'56.90" |
| noise | Quiet | 1 | sound pressure level | 20dB | 30dB |
| temperature | Hot | 1 | temp. | 28°C | 38°C |
| ... | ... | ... | ... | ... | ... |

Fig. 4. Example of context translation map.

domain space of a feature into consecutive range segments, and computes the difference of sets of queries whose state changes across consecutive segments. This structure is also memory efficient since it only stores the differences between queries over successive ranges without replication.

The structure often requires no further evaluation since a data update may fall into the same segment as before. Even if it does not, it is most likely that the update will fall into a nearby segment. In this case, a new evaluation can be performed by computing the union of the precomputed differences. No complex computations are involved in this process other than the union of differences. The union is taken over just a small number of consecutive segments starting from the previous segment. This approach outperforms the state-of-the-art query indexing mechanisms [27], [28] by orders of magnitude.

The CMQ evaluation approach, the shared and incremental processing, is based on our previous work [29]. In this paper, we extend the work for efficient CMQ evaluation.

### 4.1 CMQ Translation

CMQ translation is the first step to enable scalable CMQ evaluation. This process converts CMQs specified in context-level semantics into range predicates over continuous feature data. Through this translation, SeeMon avoids the overhead of continuous context recognition. The CMQ translation requires two major steps. First, SeeMon maps a context type to one or more features. A feature represents data values generated via preprocessing and feature extraction from sensor data. One or more features can be derived from a sensor. For example, DC and energy features are derived from an accelerometer [7]. Second, SeeMon transforms a context value into numerical data value ranges for corresponding features.[1] For example, $(\text{noise} == \text{Quiet})$ can be mapped to $(20\ \text{dB} \leq \text{sound pressure level} \leq 30\ \text{dB})$. Note that the query translation cost is negligible since the translation is a simple one-time operation performed during query registration.

SeeMon maintains a *context translation map* to support the CMQ translation effectively. Fig. 4 shows an example map. The map manages mappings between context-level semantics and data-level semantics for a context type and its possible value. By using it, SeeMon easily translates context elements in a CMQ into a set of corresponding features and data value ranges. The context translation map can be built through a machine learning process such as building a C4.5 decision tree [7], [24]. The decision tree can be easily transformed into the map.

1. This kind of mapping between a context and feature values is based on crisp limits, one of quantization methods used for context recognition [14].

**CMQ-Table**

| Query ID | State | Context Element List | Timestamp | Period |
|---|---|---|---|---|
| $Q_1$ | false | $[F_1, (b_0, b_1), \text{false}]$, $[F_2, (b_4, b_6), \text{false}]$ $[F_4, (b_4, b_7), \text{true}]$, $[F_8, (b_0, b_3), \text{false}]$ | 12:49:17 | null |
| $Q_2$ | false | $[F_1, (b_0, b_2), \text{false}]$, $[F_3, (b_3, b_4), \text{false}]$ $[F_6, (b_5, b_8), \text{undecided}]$ | 12:49:57 | 10:00 |
| $Q_3$ | true | $[F_1, (b_1, b_5), \text{true}]$, $[F_2, (b_2, b_4), \text{true}]$ | 12:50:23 | null |
| $Q_4$ | undecided | $[F_1, (b_2, b_3), \text{true}]$, $[F_5, (b_0, b_5), \text{true}]$ $[F_6, (b_3, b_5), \text{undecided}]$ | 12:50:23 | null |
| ... | ... | ... | ... | ... |

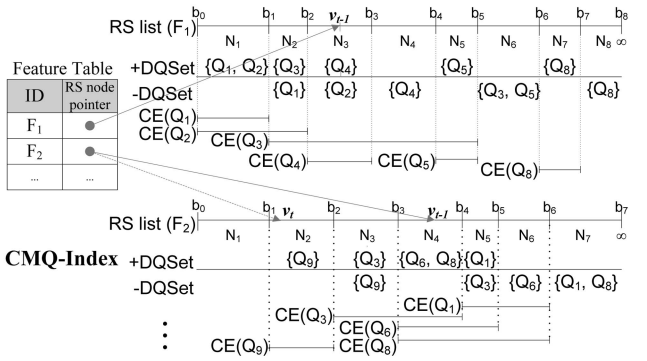



Fig. 5. CMQ-Table and CMQ-Index. The CMQ-Table shows four CMQs, $Q_1$-$Q_4$, and their states as well as the lists of included context elements. Among them, $Q_2$ has an auxiliary attribute, period for timed transition alarm. The CMQ-Index shows two RS lists, one for feature $F_1$ and the other for feature $F_2$. The RS list for feature $F_1$ currently has eight RS nodes, $N_1$-$N_8$.

SeeMon supports two types of maps: generic and customized maps. The generic map maintains mapping information generally usable to many applications. It is provided by the SeeMon framework and cannot be modified. For the customization of mappings between context-level semantics and data-level semantics, application developers can create customized maps. It is very useful to satisfy the different needs of a specific application.

### 4.2 CMQ-Index and CMQ-Table

For efficient CMQ evaluation, the CMQ Processor maintains two important data structures: the *CMQ-Table* and the *CMQ-Index* (see Fig. 5). First, the CMQ-Table stores CMQs using a hash structure, providing $O(1)$ lookup time. It contains four basic attributes: *query id*, *state* (evaluation result), *context element list*, and *time stamp* (evaluation time). For CMQs with a timed transition alarm condition, an auxiliary attribute, *period* is included. In the context element list, a context element is specified with three attributes: *feature id, range condition,* and *state*. A feature id indicates a feature associated with the context element. A range condition presents a data value range for the feature as described in Section 4.1. Note that the state of the context element is one of three states: *true*, *false*, and *undecided*. In particular, undecided states occur when feature data are unavailable due to the dynamic sensor control. After the states of a set of context elements are decided, the state of the query is decided according to the following rules (see Fig. 5):

1. The state of CMQ is false if the number of false context elements $>= 1$.
2. The state of CMQ is undecided if there is no false context element and the number of undecided context elements $>= 1$.
3. The state of CMQ is true if all context elements are true.

Second, the CMQ-Index is a query index to quickly access context elements relevant to incoming data. Using the index, context elements within range of where the data value falls can be easily identified. The index consists of multiple *Region Segment (RS) lists* and a *feature table*. An RS list is assigned to each feature and is built to maintain the value ranges of the context elements associated with the corresponding feature. Each entry of the feature table maintains a pointer to the value range, where the last data value fell.

The RS list is composed of a set of RS nodes, partitioning the *domain space* of feature values. Each RS node includes a set of context elements covered by its range (see Fig. 5). For each context element, a query id of the element is stored into only two RS nodes, where the range starts and ends. Compared to other indexes [27], [28], the CMQ-Index is more storage-efficient.

The RS list is formally defined as follows: Let $CE = \{CE_i\}$ be a set of context elements associated with a feature, where $CE_i$ has the range ($l_i$, $u_i$). Let $B$ denote the set of lower and upper bounds of the range of each $CE_i$ and minimum and maximum values of domain space, $b_{min}$ and $b_{max}$, i.e., $B = \{b \mid b \text{ is either } l_i \text{ or } u_i \text{ of a } CE_i \in CE\} \cup \{b_{min}, b_{max}\}$. We denote the elements of the set $B$ with a subscript in the increasing order of their values. That is, $b_0 < b_1 < \cdots < b_m$. An RS list is a list of RS nodes, $<N_1, N_2, \ldots, N_m>$. Each RS node $N_i$ is a tuple $(R_i, +DQSet_i, -DQSet_i)$, where

- $R_i$ is the range of region segment $(b_{i-1}, b_i)$, $b_i \in B$;
- $+DQSet_i$ is the set of CMQs, where the CMQs contain a context element $CE_k$ such that $l_k = b_{i-1}$ for the range $(l_k, u_k)$ of $CE_k$;
- $-DQSet_i$ is the set of CMQs, where the CMQs contain a context element $CE_k$ such that $u_k = b_{i-1}$ for the range $(l_k, u_k)$ of $CE_k$.

In Fig. 5, two RS lists are shown as an example. The upper RS list is built for six context elements, CE $(Q_1), \ldots,$ CE $(Q_5)$, and CE($Q_8$). Eight RS nodes are created and each of them has a range and $\pm DQSet$.

CMQs can be dynamically registered and deregistered. A CMQ $Q_{in}$ is registered as follows: First, an entry for $Q_{in}$ is added to the CMQ-Table. Since the states of $Q_{in}$ and its context elements are not determined yet, the CMQ Processor evaluates the states of $Q_{in}$ and context elements through current data values. Then, the CMQ-Index is updated. That is, the CMQ Processor updates the RS lists associated with features of context elements of $Q_{in}$. Consider a context element of $Q_{in}$, $CE_i$, whose range condition is ($l_i$, $u_i$). First, the CMQ Processor locates the RS node $N_i$, which contains $l_i$, i.e., $b_{i-1} \le l_i < b_i$. If $l_i$ is equal to $b_{i-1}$, $Q_{in}$ is inserted into the $+DQSet_i$ of $N_i$. Otherwise, $N_i$ is split into two RS nodes: the left node with the range of $(b_{i-1}, l_i)$ and the right node with the range of $(l_i, b_i)$. The left node has the $\pm DQSet$ of $N_i$ and the right node contains $Q_{in}$ in its $+DQSet$. Second, the CMQ Processor locates and processes the RS node, $N_j$ containing $u_i$ in a similar way. CMQs can be deregistered similarly.

### 4.3 CMQ Evaluation Mechanism

CMQ evaluation is performed in three steps. First, using the CMQ-Index, the CMQ Processor searches for the context elements whose state changes based on the arrival of feature data. Second, the CMQ Processor updates the CMQ-Table for the state-changed context elements. Then, it checks whether the state of corresponding CMQs should change or not. If they should, the CMQ Processor updates the CMQ-Table with the new state. To evaluate both CMQs with an instant transition alarm condition and CMQs with a timed transition alarm condition, such a state change detection is essential. Finally, the CMQ Processor checks an alarm condition of state-changed CMQs and notifies relevant applications through the Application Broker. For CMQs with an instant transition alarm condition, just the state change suffices for the notification. However, for CMQs with a timed transition alarm condition, difference between the current evaluation time and the time stamp should be compared with the specified period. If necessary, the time stamp is updated with the current evaluation time.

Searching the CMQ-Index is done as follows: Upon feature data arrival, the CMQ-Index locates an RS list associated with the feature and searches for an RS node that contains the value, i.e., a matching RS node. Queries with state-changed context elements are simply retrieved by traversing from the previous matching node to the current matching node. Due to the data locality, an updated data value will probably be available in a nearby node. Thus, the linear traversal is normally fast.

The CMQ-Index search results in two sets of queries containing state-changed context elements. 1) $QSet^+$, a set of queries containing context elements whose state changes from false to true. 2) $QSet^-$, a set of queries containing context elements whose state changes from true to false.

Given values of two consecutive updates, $v_{t-1}$ and $v_t$, let $v_{t-1}$ fall in the range of an RS node $N_j$ and $v_t$ fall in that of $N_h$, i.e., $b_{j-1} \le v_{t-1} < b_j$ and $b_{h-1} \le v_t < b_h$. While traversing from $N_j$ to $N_h$, $QSet^+$ and $QSet^-$ are computed as follows:

$$
\begin{aligned}
&\text{If } j = h, QSet^+ = QSet^- = \phi,\\
&\text{If } j < h, QSet^+ = \left[\cup_{i=j+1}^{h} + DQSet_i\right] - \left[\cup_{i=j+1}^{h} - DQSet_i\right],\\
&\qquad\qquad QSet^- = \left[\cup_{i=j+1}^{h} - DQSet_i\right] - \left[\cup_{i=j+1}^{h} + DQSet_i\right],\\
&\text{If } j > h, QSet^+ = \left[\cup_{i=j}^{h+1} - DQSet_i\right] - \left[\cup_{i=j}^{h+1} + DQSet_i\right],\\
&\qquad\qquad QSet^- = \left[\cup_{i=j}^{h+1} + DQSet_i\right] - \left[\cup_{i=j}^{h+1} - DQSet_i\right].
\end{aligned}
$$

In Fig. 5, we assume that the previous value $v_{t-1}$ of feature $F_2$ was located in $N_4$ of RS list ($F_2$). If the current value $v_t$ is located in $N_2$, $\pm DQSet$ is retrieved while visiting from $N_4$ to $N_2$. Thus, $QSet^+ = \{Q_9\}$ and $QSet^- = \{Q_3, Q_6, Q_8\}$ are obtained. Then, entries for queries in $QSet^+$ and $QSet^-$ are updated in the CMQ-Table. For instance, the context element of $Q_3, [F_2, (b_2, b_4), true]$ is updated to $[F_2, (b_2, b_4), false]$ since $Q_3$ is included in $QSet^-$. The state of $Q_3$ is also updated to false.

### 4.4 Analysis of Processing and Storage Costs

The processing cost of the CMQ Processor can be represented as the total number of retrieved context elements for each feature. The average number of retrieved context elements $U$ is determined by two factors. First, $U$ is proportional to the average distance between two consecutive data values. As the distance increases, more RS node visits are required to locate a new matching node, thereby increasing the number of retrieved context elements whose state changes. We define *Fluctuation Level (FL)* as the average distance normalized with respect to the domain size:

$$FL = \frac{\text{Average distance}}{\text{Domain size}} = \frac{\sum_{i=1}^{M} |v_i - v_{i-1}|}{M-1} \times \frac{1}{\text{Domain size}},$$

where $v_i$ is ith data value and $M$ is the total number of data values.

Second, $U$ is proportional to the average density of context elements in an RS list. As the density increases, more context elements are retrieved with the same $FL$. The average density of context elements in an RS list can be approximated as $(2 \times N_q/\text{Domain size})$, where $N_q$ is the number of CMQs, because each query id is inserted into only two nodes of an RS list. Thus, the average processing cost of the CMQ Processor for each feature can be formulated as $\Theta(2 \times N_q \times FL)$.

The storage cost of the CMQ Processor is decided by the size of the CMQ-Table and the CMQ-Index. First, the size of the CMQ-Table is proportional to the number of CMQs, i.e., $\Theta(N_q)$. Second, the size of the CMQ-Index is a function of the size of the feature table and the RS lists. The size of the feature table is proportional to the number of input data sources $N_d$, i.e., $\Theta(N_d)$. The size of an RS list is $\Theta(2N_q)$ since each context element is inserted once into $+DQSet$ and $-DQSet$, respectively. The number of RS lists is the same as the number of entries in the feature table. Thus, the storage cost of CMQ-Index is $\Theta(N_d + 2N_qN_d)$.

## 5 Energy-Efficient Sensor Control

SeeMon employs a novel sensor control method to enhance the energy efficiency of sensors and mobile devices. The key idea for efficient sensor control is that only a small number of sensors are necessary to determine the states of all registered CMQs. It is true that an increasing number of sensors will be required for various applications, especially for fine-grained monitoring and quality service. However, in a specific context, evaluation of the registered CMQs can be accomplished by monitoring a subset of sensors. We call a set of such sensors as the ESS. The ESS dynamically changes depending on the current context and registered CMQs. However, once a context is set to a situation, it tends to stay. Likewise, the ESS does not abruptly change. Once we know the ESS, sensors not in the ESS do not have to transmit data. In this section, we present the problem of ESS calculation and our sensor control methods in detail.

### 5.1 ESS Problem

Calculating the ESS is a complicated problem. The ESS should include as few sensors as possible to save energy without compromising correct CMQ evaluation. It is also important to consider data transmission rates of sensors as well as the number of sensors in the ESS. To effectively identify the ESS, the Sensor Manager utilizes the characteristics of a CMQ's structure. A CMQ is specified in a CNF of multiple context elements. A false state of a context element in a CMQ leads to a false state of the CMQ itself. The other context elements included in the CMQ are not necessary to determine the state of the CMQ. On the other hand, a CMQ in a true state requires all context elements included in the CMQ to be monitored.

As described before, the core of CMQ evaluation is to detect whether the states of CMQs change or not. For a true-state CMQ, if the state of a single context element changes to false, the state of the CMQ changes to false as well. Thus, we should monitor all the context elements in the CMQ to see if the CMQ state changes. All sensors related to the context elements should be included in the ESS. A CMQ in an undecided state should be handled similarly. To decide a

| Sensor_ID | $S_0$ | $S_1$ | | $S_2$ | $S_3$ | $S_4$ | $S_5$ |
|---|---|---|---|---|---|---|---|
| Update Rate | 1 | 1 | | 1 | 1 | 1 | 1 |
| Feature_ID | $F_0$ | $F_1$ | $F_2$ | $F_3$ | $F_4$ | $F_5$ | $F_6$ |
| Value | 19 | 27 | 2 | *U* | 6 | 72 | 38 |

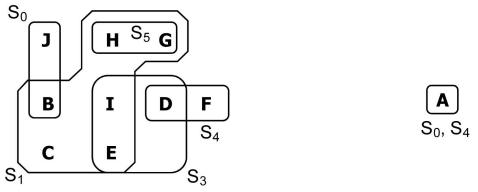
*U*: undecided

(a)

| QID | Condition | Result |
|---|---|---|
| A | $(12<F_0<25) \wedge (50<F_5<90)$ | T |
| B | **$(F_0<10) \wedge (F_1<16)$** $\wedge$ *$(60<F_3)$* $\wedge (F_4<20)$ | F |
| C | **$(12<F_1<24) \wedge (6<F_2<9)$** $\wedge (5<F_4<10)$ | F |
| D | *$(40<F_3<60)$* $\wedge$ **$(10<F_4<15) \wedge (50<F_5<60)$** | F |
| E | **$(3<F_2<9) \wedge (10<F_4<20)$** $\wedge (60<F_5<80)$ | F |
| F | $(1<F_4<10) \wedge$ **$(75<F_5<80)$** | F |
| G | **$(3<F_2)$** $\wedge$ *$(40<F_3<60)$* $\wedge$ **$(39<F_6<52)$** | F |
| H | **$(3<F_2<6)$** $\wedge (F_5<80) \wedge$ **$(F_6<26)$** | F |
| I | **$(9<F_2<18)$** $\wedge$ *$(50<F_3<60)$* $\wedge$ **$(10<F_4<20)$** | F |
| J | **$(20<F_0)$** $\wedge (F_2<6) \wedge$ *$(40<F_3<50)$* | F |

Bold character: false feature

(b)




(c) (d)

Fig. 6. Example of ESS problem. (a) Sensor set $S$, $\mathrm{S} = \{\mathrm{S}_0, \mathrm{S}_1, \mathrm{S}_2, \mathrm{S}_3, \mathrm{S}_4, \mathrm{S}_5\}$. (b) Query set. (c) False-state query set, $\mathrm{F\text{-}QSet} = \{\mathrm{B}, \mathrm{C}, \mathrm{D}, \mathrm{E}, \mathrm{F}, \mathrm{G}, \mathrm{H}, \mathrm{I}, \mathrm{J}\}$. (d) True-state query set. $\mathrm{T\text{-}QSet} = \{\mathrm{A}\}$.

CMQ's state, the states of all context elements must be checked and sensors related to the context elements should be included in the ESS. However, for false-state CMQs, monitoring only a single context element in a false state is sufficient as long as its state remains the same. Only when its state changes, do the states of the other elements need to be monitored. Thus, the opportunity to save energy comes from exploiting false-state CMQs. We select a single context element in a false state; sensors unrelated to the element can be put into an inactive state. It is also important to choose a false-state context element associated with the most energy-efficient sensor. For simplicity of discussion, we use data transmission rate as a stand-in for energy consumption.

The ESS problem consists of two subproblems: to find essential sensors for true-state and undecided-state CMQs and to find the essential sensors for false-state CMQs. Fig. 6 shows an example of ESS problem for a set of sensors and CMQs. Only query A is true. Thus, features $\mathrm{F}_0$ and $\mathrm{F}_5$ have to be monitored since they are related to the context elements of A. Accordingly, sensors $\mathrm{S}_0$ and $\mathrm{S}_4$ should be in the ESS and update data. On the other hand, query B is false and its state can be determined either by feature $\mathrm{F}_0$ or $\mathrm{F}_1$. Thus, we can put either $\mathrm{S}_0$ or $\mathrm{S}_1$ into an inactive state. Similarly, other CMQs can be evaluated using a small number of sensors. Sensors $\mathrm{S}_0$, $\mathrm{S}_1$, and $\mathrm{S}_4$ suffice to evaluate all the registered CMQs.

As described above, it is simple to calculate ESS for the true-state CMQs and undecided-state CMQs. However, it is

// ESS Calculation (*T-QSet*, *F-QSet*, *U-QSet*, *S*)
*S*: a set of all sensors
*T-QSet*: a set of all true-state CMQs
*F-QSet*: a set of all false-state CMQs
*U-QSet*: a set of all undecided-state CMQs
*q.sensor*: a set of sensors which are associated with the context elements of a CMQ *q*.

1. *TQCover, UQCover, TUQcover, RF-QCover* ← ∅
2. for ∀ $q_i$, where $q_i \in$ *T-QSet*,
   *TQCover* ← *TQCover* ∪ $q_i$ *.sensor*
3. for ∀ $q_i$, where $q_i \in$ *U-QSet*,
   *UQCover* ← *UQCover* ∪ $q_i$ *.sensor*
4. *TUQCover* ← *TQCover* ∪ *UQCover*
5. *RF-QSet* ← *F-QSet*
6. for ∀ $s_i$, where $s_i \in$ *TUQCover*,
   for ∀ $q_i$, where $q_i \in$ *F-QSet*,
   if $q_i$ evaluates to false by sensor $s_i$
   *RF-QSet* ← *RF-QSet* – { $q_i$ }
7. for ∀ $q_i$, where $q_i \in$ *RF-QSet*,
   for ∀ $s_i$, where $s_i \in q_i$*.sensor*,
   if $q_i$ evaluates to false by sensor $s_i$
   *RF-QCover* ← *RF-QCover* ∪ { $s_i$ }
8. *MCFQCover* ← Greedy-MCFSS (*RF-QSet*, *RF-QCover*)
9. *ESS* ← *TUQCover* ∪ *MCFQCover*
10. Output *ESS*

Fig. 7. ESS calculation algorithm.

complicated to compute the set of essential sensors with minimum cost for the false-state CMQs. We call this problem *minimum cost false-query covering sensor selection* (MCFSS). We formally define MCFSS problem as follows:

Given a finite set of false-state CMQs *F-QSet* and a set *S* of sensors, each of which covers a subset of *F-QSet*, find a subset $S' = \{S'_1, \ldots S'_k\}$ of $S$ such that $\cup_{i=1}^{k} F\text{-}QSet'(S'_i)$ covers *F-QSet* and $\sum_{i=1}^{k} COST(S'_i)$ is minimal, where $F\text{-}QSet'(S_i)$ is the set of false-state CMQs, which becomes false by a sensor $S'_i$ and $COST(S'_i)$ is the data transmission rate of $S'_i$.

**Theorem 1.** *MCFSS is NP-complete.*

**Proof.** We prove that MCFSS is NP-complete by reducing a well-known NP-complete problem, Minimum Cost Set Cover (MCSC) to MCFSS. MCSC consists of a finite set of elements U and a collection L of subsets of U. Each subset $L_i$ has a cost $C_i$. The objective is to choose a minimum cost subset $S'$ from $S$ that covers all elements of U.

Define F-QSet to be the set of all false-state CMQs that are false by the sensors of S, and define each sensor $S_i \in S$ to be the set of false-state CMQs that become false by $S_i$. Now, MCSC is easily transformed into MCFSS in polynomial time by considering U as F-QSet and L as $S_i$.

We have shown a reduction from MCSC to MCFSS, and therefore, MCFSS is NP-hard. Since solutions for the decision problem (i.e., $\sum_{i=1}^{k} \mathrm{COST}(S'_i) < w$, where $w$ is a positive constant) of MCFSS are verifiable in polynomial time, it is in NP. Consequently, the MCFSS problem is NP-complete. □

## 5.2 ESS Calculation

Fig. 7 shows the ESS calculation process. The ESS is computed through two stages: computing required sensors for CMQs in a true or undecided state (Steps 2-4 in Fig. 7), and then for CMQs in a false state (Step 8). We call the sensors required for true-state CMQs and undecided-state CMQs the

// Greedy-MCFSS (*F-QSet*, *S*)
*F-QSet*: a set of false-state CMQs
*S*: a set of sensors, each of which covers a subset of *F-QSet*

1. $M \leftarrow \emptyset$ // a minimum cost subset
   $S' = S$
2. while $F\text{-}QSet'(M) \subset F\text{-}QSet$ do
   Find $S_c$ in $S'$ such that $a(S_c) = \min_{s \in S'}(a(s))$,
   where $a(s) = \dfrac{COST(S_i)}{|F\text{-}QSet'(S_i) \cap F\text{-}QSet - F\text{-}QSet'(M)|}$
   , i.e., the cost-effectiveness of $s$
   $M \leftarrow M \cup S_c$
   $S' = S' - \{S_c\}$
3. Output the chosen sensors $M$

Fig. 8. Greedy-MCFSS algorithm.

TQCover and UQCover, respectively. Including TQCover (Step 2) and UQCover (Step 3) in the ESS in advance can reduce the overhead because there are false-state CMQs whose state can be identified by sensors in TQCover and UQCover. Since those sensors are already in the ESS, we can remove the false-state CMQs from the problem space of MCFSS, *F-QSet*. Steps 5-7 perform such a task. The reduced *F-QSet* is stored in *RF-QSet* in the algorithm.

Since the MCFSS problem is NP-complete, we employ a greedy heuristic algorithm, Greedy-MCFSS (see Fig. 8). The objective in designing the algorithm is to reduce the energy cost as much as possible while simplifying the computation. For this purpose, the algorithm iteratively selects the most cost-effective sensor until all false-state CMQs are covered (Step 2 in Fig. 8). The cost effectiveness of a sensor $S_i$ is defined as the average cost incurred by $S_i$ covering new false-state CMQs, i.e.,

$$\frac{COST(Si)}{|F\text{-}QSet'(Si) \cap F\text{-}QSet - F\text{-}QSet'(M)|},$$

where $M$ is the set of sensors already selected at the beginning of an iteration and $F\text{-}QSet'(M)$ is the set of false-state CMQs that are falsified by sensors in M.

The Greedy-MCFSS yields an MCFQCover, achieving an approximation ratio of $log|F\text{-}QSet|$. It is intuitive to see that the time complexity of the algorithm is $O(|S|^2)$ in the worst case, where $|S|$ is the number of sensors. For the brevity of presentation, we do not present the details of the algorithm analysis in this paper. Interested readers can refer to [46], [47] that analyze a greedy algorithm for the minimum set cover problem.

## 5.3 ESS Calculation Policy

We design two ESS calculation policies considering the trade-off between energy efficiency and ESS calculation overhead. The ESS needs to be calculated whenever the evaluation result of any CMQ changes. Frequent ESS calculation may be burdensome even with a greedy heuristic algorithm. To address this problem, an aggressive policy and a conservative policy are presented. The aggressive one is the default policy, which aims to maximize energy saving. Under the aggressive policy, the ESS Calculator calculates a new ESS to find the most cost-effective set of sensors whenever the evaluation results of any CMQ change. In contrast, the conservative policy is designed to reduce the

ESS calculation overhead rather than maximize energy efficiency. The conservative policy is effective when the processing overhead is high due to numerous CMQs and the mobile device's limited computing power.

The main idea of the conservative policy is to delay ESS calculation in order to reduce the computational overhead. It computes only TQCover and UQCover to identify the necessary sensors for correct CMQ evaluation without calculating a new ESS. While the ESS calculation is being delayed, sensors can be added to the TQCover and UQCover and become active. However, to achieve a certain level of energy efficiency, an ESS should be updated before too many sensors are activated. Thus, the conservative policy should have criteria to decide the time when a new ESS should be calculated.

Currently, the conservative policy defines the sensor activation ratio (SAR) as the deciding criteria. The SAR quantifies how many sensors become newly active after the last ESS calculation. It is not practical to use the number of currently active sensors as a criterion in deciding the ESS calculation timing because the number of necessary active sensors varies depending on the evaluation results of the registered CMQs. Thus, we focus on the change in the number of active sensors. To apply the SAR-based conservative policy, we provide the following metric:

$$SAR = N_{inactive \rightarrow active}/N_{inactive},$$

where $N_{inactive}$ is the number of sensors that became inactive at the last ESS calculation and $N_{inactive \rightarrow active}$ is the number of sensors that become newly active among the sensors that were inactive at the time of the last ESS calculation.

Given a predefined threshold value for the metric, the ESS Calculator updates the ESS if the metric value goes beyond the threshold value.

### 5.4 Sensor Control

The Sensor Controller controls sensors based on the ESS calculation result. Basically, it sends a control message to the sensors that are not included in the calculated ESS. The control message configures the sensors to be put into the inactive state so that the sensors stop transmitting data. Afterward, the ESS Calculator updates the state of context elements related to the inactive sensors in the CMQ-Table. Specifically, it changes the state of those context elements to undecided.

We design two sensor control modes for inactive sensors: a *data transmission avoidance* control and an *idle mode utilization* control. As a simple and basic approach, the data transmission avoidance control puts sensors into RX (receive) mode. A sensor in RX mode still can receive a message to restart transmission and promptly send data again. In this mode, the Sensor Controller can have full control over sensors; it can control when sensors stop data transmission and when sensors restart data transmission. However, energy saving is limited. The energy consumption of wireless sensors highly depends on the radio mode of the sensor's wireless transceiver. Generally, RX (receive) and TX (transmit) modes consume much more energy than idle mode or power down mode (e.g., for the CC2420 RF transceiver used for the MicaZ mote, current consumption of RX and TX mode is 18.8 and 17.4 mA, respectively, but that of idle and power down mode is 426 and 20 $\mu$A, respectively [44]). Thus, it is desirable to put sensors into idle mode or power down mode.

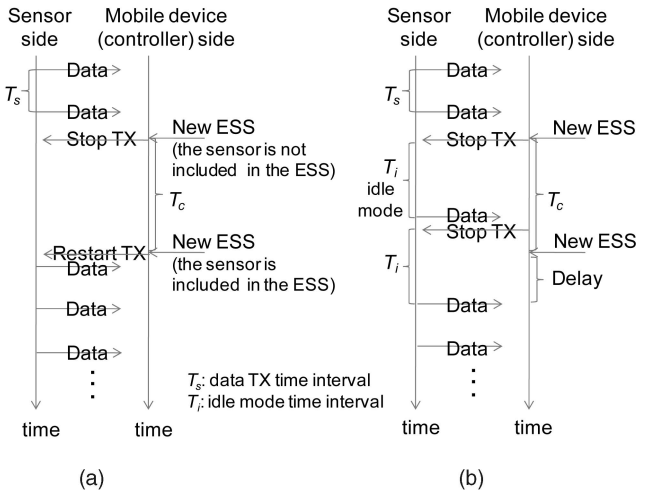



Fig. 9. Sensor control modes. (a) Data transmission avoidance. (b) Idle mode utilization.

To address this problem, we devise the idle mode utilization control. It puts sensors not included in the ESS into idle mode. As a result, the sensors can save much more energy than avoiding data transmission only. However, the Sensor Controller cannot configure the sensors to restart data transmission. In idle mode, sensors cannot perform wireless communication. Thus, they cannot receive a control message to restart data transmission once they are put into the idle mode. In this case, the sensors should check whether they need to transmit data again on their own.

Fig. 9 shows the operation of the two sensor control modes. With the data transmission avoidance control (Fig. 9a), an inactive sensor simply skips data transmission after receiving a control message (Stop TX). The sensor continues in RX mode. If the sensor is included in a new ESS result, the Sensor Controller sends a control message (Restart TX) to the sensor to restart data transmission. Energy saving depends on the number of reduced data transmissions for a time interval $T_c$. However, energy saving is restricted since the sensor remains in RX mode.

Fig. 9b shows the idle mode utilization control. A sensor is put into idle mode after receiving a control message (Stop TX). After an idle mode time interval $T_i$, the sensor goes into TX mode and transmits data to check whether it is included in the ESS. If so, the Sensor Controller does not send any message so that the sensor keeps transmitting data. If not, the Sensor Controller sends a control message (Stop TX) again. This may result in additional checking message overhead and time delay for data reception (at most $T_i$). It is important to determine $T_i$ carefully since it affects both energy saving and delay. In Section 7.5, we present an empirical study on the effects of $T_i$ selection and the trade-off between energy saving and delay in detail.

## 6 Implementation

We have implemented the SeeMon system architecture as a prototype system, carefully applying the scalable CMQ evaluation and energy-efficient sensor control mechanisms. We have also built two example applications on top of it, where SeeMon plays a critical role as an underlying context monitoring platform. The prototype is implemented in C++

TABLE 2
Sensor, Feature, and Context Profiles

| Sensor | Sampling rate | Feature | Feature generation rate | Context type (# of possible values) | Context value examples |
|---|---|---|---|---|---|
| BVP sensor | 60 Hz | Heart rate | ~3 Hz | Heart rate (10) | Fast, Normal |
| | | Stress | ~3 Hz | Stress (4) | High, Low |
| GSR sensor | 60 Hz | Skin conductance | 60 Hz | Strain (4) | High, Low |
| | | Startle event | 60 Hz | Startle event (2) | Yes, No |
| Light sensor | 0.72 Hz | Illumination | 0.72 Hz | Light (7) | Dark, Bright |
| Temperature sensor | 0.36 Hz | Temperature | 0.36 Hz | Temperature (8) | Cool, Hot |
| Humidity sensor | 0.18 Hz | Humidity | 0.18 Hz | Humidity (6) | Dry, Humid |
| Three 2-axial acceleration sensors | 48.08 Hz ×6 | DC | 4.808 Hz ×6 | Activity (13) | Running, Sitting |
| | | Energy | 4.808 Hz ×6 | | |
| GPS sensor | 2 Hz | Longitude | 2 Hz | Outdoor location (9) | CS building, East restaurant |
| | | Latitude | 2 Hz | | |
| | | Speed | 2 Hz | Speed (5) | Walking, Bicycling |
| | | Direction | 2 Hz | Direction (8) | North, West |
| Timer (S/W sensor) | - | Time | 0.1 Hz | Time (8) | Dawn, Noon |
| Indoor location (S/W sensor) | manual input | Indoor location | 1 Hz | Indoor location(12) | 1st floor lobby, Room 2432 |

on Linux. The total lines of prototype system code are about 8,700.

### 6.1 Prototype Hardware

Deploying SeeMon requires two important hardware sets: mobile devices and sensors. Currently, we have deployed the SeeMon prototype and its applications on two different mobile devices: 1) an Ultra Mobile PC (UMPC), SONY VAIO UX27LN with Intel U1500 1.33 GHz CPU and 1 GB RAM, and 2) a custom-designed wearable device with Marvell PXA270 processor and 128 MB RAM. The former represents powerful future mobile devices and the latter a relatively resource-limited current mobile device.

The diversity and scale of sensors determine the coverage and accuracy of the context monitoring of SeeMon. Currently, we have incorporated many sensors that are commercially available and widely used for diverse context-aware applications. Table 2 shows the sensors that we used in our current prototype. Considering the wearability and controllability of wireless sensors, we mainly use five of USS-2400 [31] sensor nodes, i.e., a light sensor, a temperature/humidity sensor, and three 2-axial acceleration sensors. They are equipped with Atmega 128L MCU, CC2420 RF module supporting 2.4 GHz band ZigBee protocol, and TinyOS as an operating system. To provide communication between the mobile device and sensors, we attach one base sensor node to the mobile device using serial or USB interfaces.

We incorporated several additional sensors to provide important context types not supported by USS-2400 nodes. First, we use a Bluetooth-enabled GPS sensor to position outdoor location. We also incorporate two biomedical sensors, a Blood Volume Pulse (BVP) sensor and a Galvanic Skin Response (GSR) sensor, which are essential to recognizing the user's affective context [32] and medical context. Finally, two software sensors are used for time and indoor location. Indoor location is positioned by manual input of predefined location. To automate this manual process, we plan to couple SeeMon and indoor positioning system deployed in our university [30].

### 6.2 SeeMon Implementation

Implementing a working prototype of the SeeMon architecture requires a careful choice of programming models. First, we implemented SeeMon as a multithread system for intuitive development and concurrency. Each system component runs as a single thread while the Application Broker is separated into two threads for query registration and result forwarding. Note that the Sensor Broker handles input data from multiple sources in a thread as well using efficient event-driven I/O multiplexing. The intercomponent communication is performed through message queues. To support frequent data transfer from the Sensor Broker to the CMQ Processor, we used double buffering.

The Sensor Broker extracts 15 features from data delivered from the sensors, as shown in Table 2. We implemented several simple techniques and utilized several existing libraries to compute features from sensor data. First, we used FFTW, a Fast Fourier Transform library [33], to obtain DC and energy features from acceleration data. Second, we implemented a NMEA data parser to extract the longitude, latitude, speed, and direction features from GPS data based on the NMEA 0183 protocol. Third, we utilized a convolution filter to remove errors, smooth signals, and detect peaks from BVP sensor data. The heart rate feature is derived from the detected peaks and stress feature is obtained through further frequency domain analysis.

The Application Broker uses the context translation map for CMQ translation. Since the context translation map influences the quality of monitoring, the learning process had to be extensive. We obtained mappings for activity contexts through user annotation-based learning [7]. The learning was done with C4.5 decision tree provided by Weka, a Java-based open source machine learning tool [34]. The learning for the level of strain, the level of stress, and startle event were conducted based on IAPS experiment [42].

The CMQ Processor and the Sensor Manager involve many operations and result in relatively high processing cost in SeeMon. We noticed that set operations such as union and difference are dominant and reducing their number and cost is essential to improve system performance. Thus, we developed a fine-tuned module for set operations to reduce their overhead. We observed that the CMQ Processor and the Sensor Manager generated many intermediate results that can be reused several times afterward. In particular, we designed a bitmap-like data structure to store the detailed information of false-state CMQs and effectively reuse it, thereby reducing a number of set operations. It improves ESS calculation performance significantly.

We implemented TinyOS applications for USS-2400 sensor nodes to apply the devised sensor control modes. The applications have two main modules. The first one is a data transmission module to transmit sensing data based on transmission timer events. The second one is a control module to receive control messages and control data transmission based on the sensor control mode. For the data transmission avoidance control, the control module simply halts the transmission timer. Accordingly, the data transmission module stops sending data. For the idle mode utilization control, the control module uses strobe command registers, SRFOFF and SRXON [44]. Upon receiving a control message, it halts the transmission timer and sets SRFOFF to put CC2420 RF transceiver into idle mode. After a specified time interval, it sets SRXON to enable RX mode

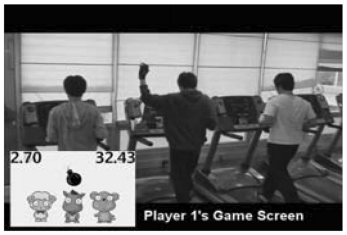



Fig. 10. Running Bomber.


Fig. 11. SympaThings.

and restarts the transmission timer to check whether data transmission should be performed again.

### 6.3 Application Development

Emerging areas such as pervasive gaming and affective computing are domains in which many new applications will be developed. For evaluation, we have prototyped two applications for each of them: Running Bomber and SympaThings. Application developers have used our prototype system and considered it effective, efficient, and stable.

Running Bomber is the first step toward applying the SeeMon framework to pervasive games. Pervasive games utilize users' various contexts and reflect their physical actions from their everyday activities. Running Bomber is a pervasive game designed to make treadmill running less boring. Fig. 10 shows a picture of Running Bomber demo. For the Running Bomber game, a player holding a bomb should pass the bomb to others within 3 seconds. Bomb passing is signaled by shaking an arm wearing an acceleration sensor. With SeeMon, developing pervasive games is much simpler; game developers only need to define the game rules and design user interfaces. In Running Bomber's case, complexities such as processing acceleration data and recognizing the motion are completely handled by SeeMon while the game rules can be reduced to a simple CMQ registration with SeeMon.

SympaThings, an application inspired by affective computing, is a demonstration of SeeMon's wide applicability. SympaThings runs on a wearable device and controls nearby smart objects to sympathize with a person's affective context. For example, a picture frame changes the picture inside and a lighting fixture adjusts its color (e.g., red color for the high degree of strain or yellow color for the low degree of strain). Efficient processing is crucial in the operating environment of SympaThings: high-rate data from BVP and GSR sensors, and many queries for nearby smart objects. SeeMon's shared and incremental processing is essential to satisfy these requirements. SympaThings is a collaborative project with HCI Lab of ICU and Semiconductor System Lab of KAIST. Fig. 11 shows the demonstration of SympaThings at Nextcom Show 2007 [35], one of the biggest IT exhibitions in Korea, held in Seoul in November 2007.

## 7 Experiments

### 7.1 Experimental Setup

We have conducted extensive experiments to evaluate the scalability and energy efficiency of SeeMon. We generated sensor data and CMQ workloads based on our motivating environment. First, we produced a data workload by collecting raw sensor data from the daily activities of a person. For data collection, a student in our laboratory carried a UMPC with eight sensors in Table 2 except BVP and GSR for 12 hours in campus. The total data rate was 291.74 samples per second. To replay and feed the collected data to SeeMon, we implemented a simple data sender. Thus, we were able to conduct our experiments multiple times under the same data workload. Second, we synthetically generated CMQ workloads to simulate numerous CMQs registered by multiple applications. They reflected various monitoring conditions on different types of contexts. Table 3 summarizes the parameters and default values used for CMQ generation (refer to Table 2 for context details). All CMQs' alarm condition is instant transition alarm, specifically $F \rightarrow T$.

TABLE 3
Parameters for CMQ Workload

| | Parameter | Default value |
|---|---|---|
| CMQ Workload | # of CMQ | 4096 |
| | # of context elements per CMQ | 4 |
| | Distribution of context type | Uniform distribution |
| | Distribution of context value | Uniform distribution |

For all experiments, we ran SeeMon on the UMPC. We scaled down the CPU frequency to 200 MHz to validate our system, considering widely used mobile devices such as Nokia N95 (330 MHz CPU, 64 MB RAM) and Samsung Blackjack (220 MHz CPU, 64 MB RAM). Memory constraints were not seriously considered since SeeMon consumes less than 5 MB even with 2,000 registered CMQs. This amount of memory is reasonable for most smart phones. The default ESS calculation policy was the aggressive policy.

### 7.2 Scalability

In this experiment, we compare the scalability of SeeMon with that of an alternative approach called *context recognition-based monitoring method*, which carefully models existing context-aware systems [13], [14], [15], [16]. It receives and preprocesses continuously arriving data from sensors, processes the data to recognize contexts, and evaluates monitoring queries to detect specified context changes as shown in Fig. 1. We assume that the alternative processes each query independently since existing work does not consider the efficient shared processing of concurrent queries.

We measure the scalability in terms of throughput while increasing input data scale from 1 to 7. Throughput is the maximum number of queries that can be handled without causing system overload.[2] Data scale 1 is the data workload

2. Currently, overload is determined by the size of the data queue, which should be processed by the CMQ Processor. It is important to detect context changes without long delay. We assume that a delay of a couple of seconds is tolerable. Accordingly, acceptable maximum queue size is set to three times of data rate.

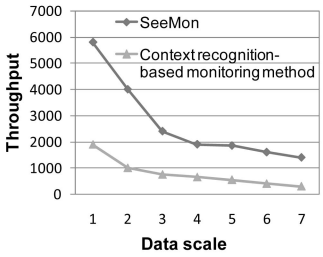



Fig. 12. Throughput.

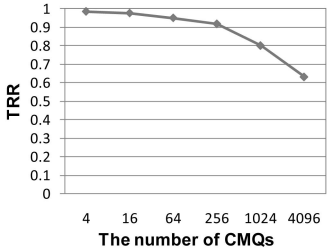



Fig. 13. TRR for number of CMQs.

under our initial sensor settings described in Section 7.1. We synthetically increase the size of data workload by replicating data traces of data scale 1. At the data scale $k$, the number of sensors and data rate becomes $k$ times larger than the initial sensor setting. We assume that the data scale 7, i.e., 56 sensors and about 2,100 samples/second, is sufficient to represent a large-scale personal sensor network. We use query workloads generated by our default setting.

Fig. 12 demonstrates the high level of scalability of SeeMon. First, SeeMon scales well with data scale. Even under data scale 7, SeeMon can process 1,400 queries, which is a reasonably large number, given the device's limited computing resources (200 MHz CPU) and the high rate of sensor data (about 2,100 samples/second). Note that such a high level of scalability is critical since the number of sensors and data rate will dramatically increase to deal with broader and more accurate contexts. Second, SeeMon scales better than a context recognition-based monitoring method. For all data scales, the throughput of SeeMon is higher than that of the alternative. Furthermore, the benefit of SeeMon becomes relatively larger as data scale increases. At data scale 1, SeeMon processes three times more queries than the context recognition-based monitoring method. However, it processes 4.6 times more queries at data scale 7. Such benefits mainly come from the shared and incremental processing of SeeMon. In contrast, context recognition-based monitoring method processes monitoring queries independently. Moreover, it does not employ any incremental processing method that can accelerate repeated CMQ evaluation. Consequently, as data scale increases, the gap between SeeMon and the context recognition-based method becomes relatively larger.

### 7.3 Energy Efficiency

In this experiment, we evaluate the energy efficiency of SeeMon in terms of Transmission Reduction Ratio (TRR). TRR quantifies the amount of reduction in wireless transmission, which is the main factor of sensors' energy consumption [36], [25]. TRR is defined as follows: $\mathrm{TRR}_i$ denotes a TRR of a sensor $i$, and $\mathrm{TRR}_S$ denotes an averaged TRR of a sensor set S:

$$TRR_i = \frac{reducedNumberOfTransmission_i}{totalNumberOfTransmission_i} = \frac{InactiveTime_i \times TransmissionRate_i}{SimulationTime \times TransmissionRate_i},$$

$$TRR_S = \frac{\sum reducedNumberOfTransmssion_i}{\sum totalNumberOfTransmission_i}, i \in S.$$

To evaluate the energy efficiency under various query workloads, we measured TRR as varying the number of registered CMQs, the number of context elements in a CMQ, and the context value distribution. To measure TRR, we logged ESS calculation results generated by the ESS Calculator and their time stamp. Unless specified, the number of CMQs and context elements is fixed to 256 and 4, respectively. Context values follow a uniform distribution. We generated 10 different query sets for each parameter setting. Each TRR value presented below was obtained by averaging TRR values of 10 measurements for the 10 query sets. Total elapsed time is 46,309 seconds.

TABLE 4
TRR of Each Sensor

| Sensor \ # of CMQ | | 16 | | 256 | | 4096 | |
|---|---|---|---|---|---|---|---|
| ID | Name | Inactive Time (s) | TRR | Inactive Time (s) | TRR | Inactive Time (s) | TRR |
| 0 | Light | 21092 | 0.4554 | 11427 | 0.2467 | 205 | 0.0044 |
| 1 | Temperature | 21100 | 0.4556 | 42 | 0.0009 | 1 | 0 |
| 2 | Humidity | 18091 | 0.3906 | 28 | 0.0006 | 75 | 0.0016 |
| 3 | Acceleration (we assign a sensor ID per axis) | 46234 | 0.9984 | 45145 | 0.9748 | 33439 | 0.7220 |
| 4 | | 46296 | 0.9997 | 45597.1 | 0.9846 | 30749 | 0.6640 |
| 5 | | 46048 | 0.9943 | 37905 | 0.8185 | 7519 | 0.1623 |
| 6 | | 46294.4 | 0.9996 | 45003 | 0.9718 | 36743 | 0.7934 |
| 7 | | 46293.8 | 0.9996 | 44903 | 0.9696 | 32821 | 0.7087 |
| 8 | | 46300 | 0.9998 | 45844 | 0.9899 | 42142 | 0.9100 |

Fig. 13 shows $\mathrm{TRR}_S$, where s is the whole sensor set used for experiments, as we increase the number of CMQs. SeeMon reduces more than 90 percent of data transmissions when the number of CMQs is fewer than 256. Even with 4,096 queries, more than 60 percent of data transmissions are eliminated. Such an energy efficiency is achieved through the ESS mechanism, which turns on only a small number of sensors to evaluate all registered CMQs. The high level of energy efficiency is critical in our target environments since high-rate communication between a mobile device and a large number of sensors will shorten the battery life of the mobile device and the sensors. In addition, we observe that TRR decreases as the number of CMQs increases. This is mainly due to the increase in the number of true-state CMQs. More true-state CMQs make more sensors active, which decreases TRR.

Table 4 describes the inactive time and $\mathrm{TRR}_i$ of each sensor when 16, 256, and 4,096 CMQs are registered. Interestingly, acceleration sensors show much higher TRRs than other sensors. Since the transmission rate of the acceleration sensor is the highest, sensor control mechanism of SeeMon frequently excludes the acceleration sensors from the ESS, thereby increasing TRR. This confirms that the transmission rate of sensors is correctly reflected in the ESS calculation algorithm. The GPS sensor, timer, and indoor location sensor are always included in the ESS. Note that the GPS sensor is not programmable. The timer and the indoor sensor are software sensors, and thus, there are no wireless transmissions to eliminate.

In our second experiment, we measure $\mathrm{TRR}_S$ as increasing the number of context elements in a CMQ. Fig. 14

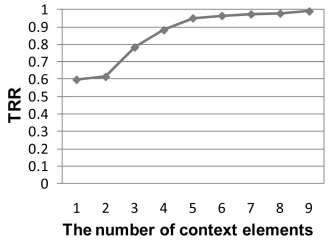



Fig. 14. TRR for number of CEs.

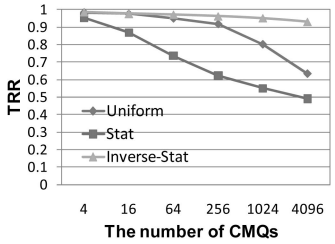



Fig. 15. TRR for distribution.

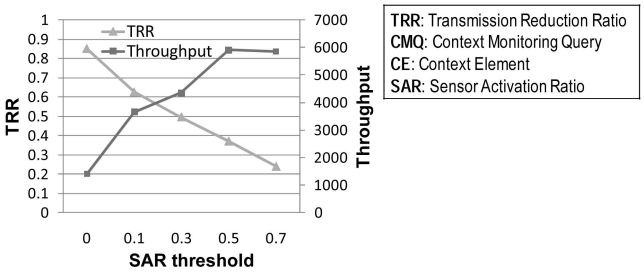



Fig. 16. Processing and energy efficiency trade-off.

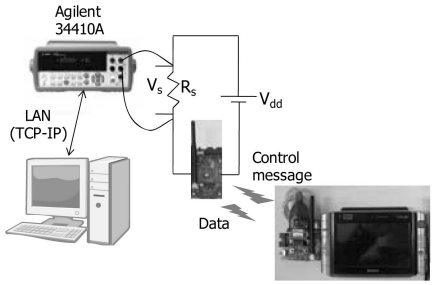



Fig. 17. Power measurement setup.

demonstrates that TRR increases as the number of context elements increases. There are two main reasons for this. First, the number of active sensors for true-state CMQs decreases. As the number of context elements increases, CMQs are more likely to be false state due to their CNF structure. The reduction in true-state CMQs results in fewer active sensors for them. Second, the number of active sensors for false-state CMQs decreases. As the number of context elements increases, the number of context elements associated with a sensor increases. Then, the number of false-state CMQs associated with the sensor also increases. Therefore, all false-state CMQs can be covered by fewer sensors.

To investigate the effect of query distribution, we generate three different CMQ distributions and measure TRR with them. To model three different realistic distributions of context element values, we generate Stat, Inverse-Stat, and Uniform distributions. The Stat distribution represents a common querying pattern in which users are interested in frequently occurring context values. The Inverse-Stat distribution represents the opposite case. By analyzing our real data trace, we extract the probability density of each context value, and then generate Stat and Inverse-Stat distributions. The Uniform distribution is used for a primitive comparison. The number of CMQs is varied from 4 to 4,096, and the number of context elements is fixed to 4.

Fig. 15 shows TRR according to the CMQ distributions. The key observation is that the Stat and Inverse-Stat distributions show the lowest and the highest TRRs, respectively. This holds regardless of the number of queries. In the Stat distribution, most CMQs contain frequently occurring context values. Thus, the state of the CMQs can be true with a high probability. Corresponding sensors have to be active, resulting in the lower TRR. In contrast, sensors in the Inverse-Stat distribution are likely to be inactive, resulting in the higher TRR.

### 7.4 Processing-Energy Efficiency Trade-Off

This experiment shows a trade-off between processing efficiency and energy efficiency determined by the ESS calculation policies described in Section 5.3. Such a trade-off characteristic is very important to adapt SeeMon to various computing- and battery-resource environments. We measure throughput as a processing efficiency metric and TRR as an energy efficiency metric while varying SAR threshold values. Note that threshold 0 represents the aggressive policy. Data scale 7 is used as a sensor data workload and a query workload is generated with the default setting.

Fig. 16 shows a trade-off between throughput and TRR. As we expected, the aggressive policy (threshold 0) shows the highest TRR, but shows the lowest throughput. As an SAR threshold value increases, the TRR linearly decreases, but the throughput increases accordingly. Compared to the aggressive policy, the conservative policy with threshold 0.7 achieves 4.2 times greater throughput with 3.6 times less TRR. Such results are mainly due to SeeMon performing complex ESS calculations less frequently with a higher SAR threshold value. Thus, the energy efficiency degrades while processing efficiency is enhanced.

### 7.5 Effect of Sensor Control Modes

We examine the effect of sensor control modes presented in Section 5.4. First, we compare the energy saving effect of different control modes by measuring the energy consumption of different modes. Second, we show a trade-off between delay and energy saving under the idle mode utilization control. For the measurement, we performed trace-based emulation. We used the log file, which was obtained in the TRR experiment. The Sensor Controller reads the file and generates control messages for both modes. The base node sends the generated messages to the sensor. The sensor then operates according to the configured mode.

For this experiment, we built a power measurement setup as illustrated in Fig. 17. The measurement setup consists of a computer running a program to collect measurement data, an Agilent 34410A digital multimeter, a USS-2400 sensor node, a series resistor ($R_s$), a DC power supply ($V_{dd}$), and a base node attached to a UMPC sending control messages. The multimeter is connected to the data collection computer via a LAN. It measures the voltage ($V_s$) across the resistor, which is connected in series with the DC power supply. Measured voltage values are transferred to the data collection program using TCP/IP. The current consumption of the sensor can be calculated by dividing the

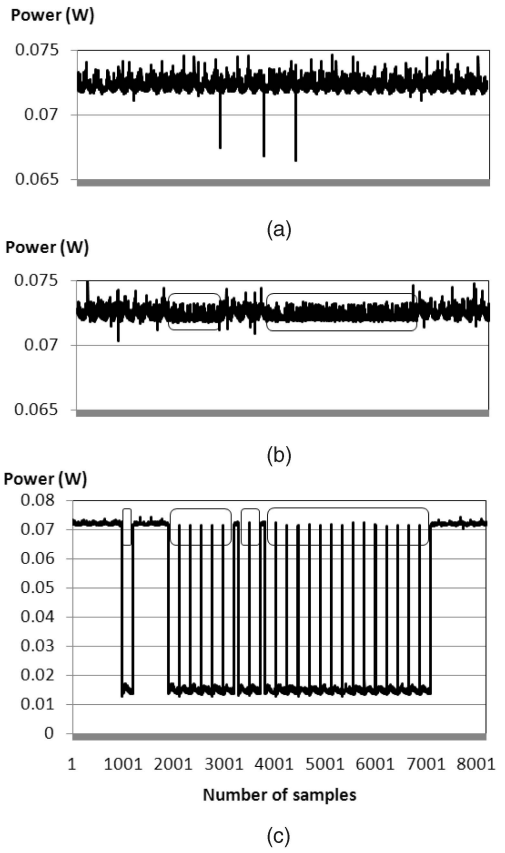



Fig. 18. Power consumption of different control modes. (a) No control. (b) Data transmission avoidance control. (c) Idle mode utilization control.

voltage drop across the resistor ($V_s$) by the resistance value. The instantaneous power consumption of the sensor is then calculated using the following equation:

$$P_s(t) = (V_{dd} - V_s(t)) \times \frac{V_s(t)}{R_s}.$$

We can obtain the energy consumption value using the equation below:

$$E_s(t) = \sum_{i=0}^{n-1} P_s(t_i) \times (t_{i+1} - t_i).$$

A control message log file used for measurement is one of 10 log files generated for light sensor TRR measurement with 256 CMQs (corresponding to ID 0 in Table 4). The obtained TRR value was 0.256716. Note that TRR values in Table 4 are average values across 10 measurements. The sampling rate of the multimeter is about 156 samples per second. The resistance value of $R_s$ is 20.43 Ω and the DC voltage value of $V_{dd}$ is 3.2 V. During the measurement, the sensor's LED was turned off so as to not include the power consumption effects of the LED in the calculation. The idle mode time interval $T_i$ is the same as the data transmission time interval $T_s$, 1.389 seconds. As mentioned, the total elapsed time was 46,309 seconds.

Fig. 18 presents the power consumption of the light sensor for about 52 seconds in the middle of the log file (about 6 seconds active, 0.936 seconds inactive, 4.91 seconds active, 7.95 seconds inactive, 0.984 seconds active, 1.46 seconds inactive, 1.929 seconds active, 20.15 seconds inactive, and 8 seconds active). The power consumption of the data transmission avoidance control (Fig. 18b) shows little difference from that of the no control case (Fig. 18a). Only some portion of the peaks was removed. Also, the short inactive state is not clearly seen in the figure. In contrast, the idle mode utilization control shows a noticeable reduction in power consumption (Fig. 18c).

TABLE 5
Total Energy Consumption

| | Sensor control mode | Energy consumption | TRR |
|---|---|---|---|
| Light sensor (256 CMQs) | No control | 3331.97J | 0 |
| | Data transmission avoidance | 3313.47J (0.6% reduction) | 25.6716% |
| | Idle mode utilization | 2632.23J (21% reduction) | 25.6716% |

Table 5 shows the total energy consumption. The energy consumption without any control was 3,331.97 J. As expected from the previous result, skipping data transmission alone hardly saves on energy consumption; the total energy consumption was 3,313.47 J, a 0.6 percent reduction. The idle mode utilization control presented 2,632.23 J of energy consumption, i.e., a 21 percent reduction compared to the no control case. The reduction ratio is relatively small compared to the given TRR value. This is mainly due to the energy consumption of probe data transmission to check whether data transmission should be performed again.

Finally, we investigate the effect of $T_i$ selection on the trade-off between energy saving and delay. To demonstrate the trade-off, we measured the delay and energy consumption as a function of $T_i$. We varied $T_i$ from $T_s$ to $6T_s$, where $T_s$ was the same as before, 1.389 seconds. To examine the impact of the $T_s$ value, we additionally performed a measurement for a relatively small $T_s$, 0.104 seconds, in consideration of sensors such as accelerometers. We used average values from two measurements for each $T_i$. All energy consumption was normalized by that of $T_s$.

Fig. 19a shows that a trade-off between delay and energy saving is hardly seen for a large $T_s$. As expected, the average delay increases as $T_i$ increases. However, the energy consumption remains almost the same (with at most a 0.6 percent difference). In this case, $T_i$ is much longer than the duration of active mode for data transmission until switching back to idle mode, i.e., $T_p$ (about 0.025 seconds). Thus, the energy overhead during $T_p$ does not constitute a significant portion of the energy consumption during $T_i$. Increasing $T_i$ does not result in noticeable energy saving. It is reasonable to use a small $T_i$ to avoid a long delay.

Fig. 19b presents the result for a small $T_s$. As opposed to the previous result, it shows a trade-off between delay and energy saving. As $T_i$ increases, the average delay also increases. However, the energy consumption decreases. For a small $T_s$, the ratio of $T_p$ to $T_i$ becomes relatively larger and the resulting energy overhead increases. Thus, it is possible to reduce energy consumption by increasing $T_i$. In this case, it is necessary to make a proper trade-off based on the delay requirement of applications in selecting $T_i$. If the delay requirement is not strict, selecting a larger $T_i$ will achieve more energy saving.

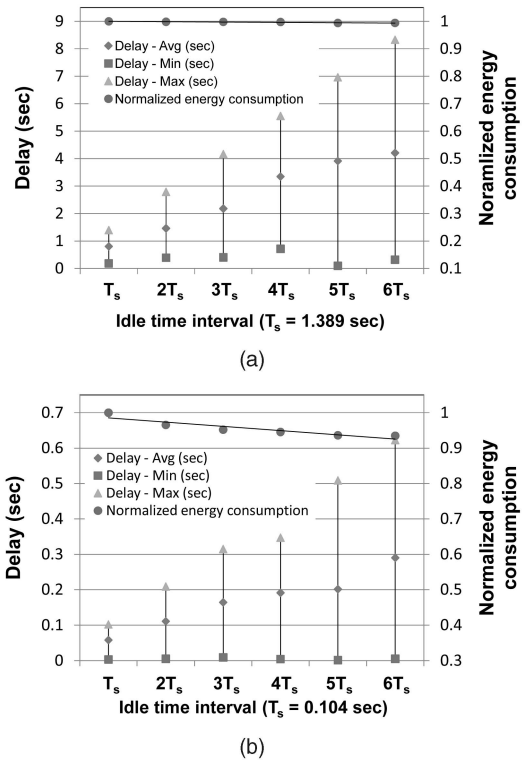



Fig. 19. Delay and energy saving trade-off. (a) Result for large $Ts$. (b) Result for small $Ts$.

## 8 Discussion and Future Work

We have presented and demonstrated the advantages and characteristics of SeeMon. We can summarize the experimental results and impact in two main ways. First, SeeMon achieves a high level of processing and energy efficiency, i.e., processing 3 to 4.6 times more queries and reducing more than 50 percent of sensor data transmission. The results are promising in that SeeMon can play a critical role in concurrently supporting multiple context monitoring applications based on a number of sensors in a highly efficient manner. Second, the experiments show that two trade-offs should be carefully considered to maximize the effectiveness of SeeMon, i.e., a trade-off between processing and energy efficiency in the ESS calculation, and a trade-off between delay and energy consumption in the idle mode control of sensors.

We will enrich context monitoring semantics in future work. Currently, our context monitoring language supports conjunctive composition in context monitoring. Other composition operators such as disjunction and sequencing could be supported along with efficient evaluation methods. We also plan to implement the framework more concretely and gain more experience with it. In particular, we will implement our SeeMon framework on top of off-the-shelf smart phones such as Nokia N96 while fully considering their resource limitations and using their on-board sensors, e.g., GPS, camera, and accelerometers.

## 9 Conclusion

We have presented SeeMon, a scalable and energy-efficient context monitoring framework for sensor-rich and resource-limited mobile environments. The key idea behind SeeMon is twofold. First, context monitoring in SeeMon focuses on the continuous detection of context changes. Second, SeeMon approaches the context monitoring problem in a bidirectional way. Applying the bidirectional approach, SeeMon achieves a high degree of efficiency in computation and energy consumption. We implemented the prototype of SeeMon system architecture, carefully applying scalable CMQ processing and energy-efficient sensor control mechanisms. We also developed several example applications on top of it in which SeeMon plays a critical role as an underlying context monitoring platform. Our evaluation shows that SeeMon achieves a high level of scalability and energy efficiency.

## Acknowledgments

This research was supported in part by the Ministry of Knowledge Economy, Korea, under the Information Technology Research Center support program supervised by the Institute of Information Technology Advancement (grant number IITA-2009-C1090-0902-0006). Also, this work was supported in part by a Korea Research Foundation Grant funded by the Korean Government (KRF-2008-220-D00113). The authors thank Taiwoo Park and Chulhong Min for their support for application development and demonstration as well as experiments. They also thank the anonymous reviewers for their valuable comments to improve the quality of this paper. The early version of this paper was presented at the Sixth International Conference on Mobile Systems, Applications, and Services, Colorado, June 2008 [45].

## References

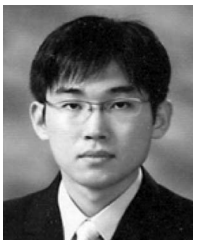
[1] K.V. Laerhoven, A. Schmidt, and H. Gellersen, "Multi-Sensor Context Aware Clothing," *Proc. Int'l Symp. Wearable Computers,* 2002.
[2] O. Amft et al., "Analysis of Chewing Sounds for Dietary Monitoring," *Proc. Conf. Ubiquitous Computing (UbiComp),* 2005.
[3] C. Park et al., "A Wearable Wireless Sensor Platform for Interactive Dance Performances," *Proc. Int'l Conf. Pervasive Computing and Comm. (PerCom),* 2006.
[4] M. Sung, C. Marci, and A. Pentland, "Wearable Feedback Systems for Rehabilitation," *J. Neuro Eng. and Rehabilitation,* vol. 2, no. 1, 2005.
[5] J.E. Bardram, "Applications of Context-Aware Computing in Hospital Work—Examples and Design Principles," *Proc. ACM Symp. Applied Computing (SAC),* 2004.
[6] T. Sohn et al., "Place-Its: A Study of Location-Based Reminders on Mobile Phones," *Proc. Conf. Ubiquitous Computing (UbiComp),* 2005.
[7] L. Bao and S.S. Intille, "Activity Recognition from User-Annotated Acceleration Data," *Proc. Pervasive,* 2004.
[8] J. Lester et al., "A Practical Approach to Recognizing Physical Activities," *Proc. Pervasive,* 2006.
[9] P. Fahy and S. Clarke, "CASS—A Middleware for Mobile Context-Aware Applications," *Proc. MobiSys,* 2004.
[10] T. Gu et al., "A Middleware for Building Context-Aware Mobile Services," *Proc. IEEE Vehicular Technology Conf. (VTC),* 2004.
[11] H. Chen, T. Finin, and A. Joshi, "An Ontology for Context-Aware Pervasive Computing Environments," *Proc. Workshop Ontologies in Agent Systems (AAMAS),* 2003.
[12] D. Salber, A.K. Dey, and G.D. Abowd, "The Context Toolkit: Aiding the Development of Context-Enabled Applications," *Proc. ACM CHI,* 1999.
[13] A. Ranganathan and R.H. Campbell, "A Middleware for Context-Aware Agents in Ubiquitous Computing Environments," *Proc. Conf. Middleware,* 2003.
[14] P. Korpipää et al., "Managing Context Information in Mobile Devices," *IEEE Pervasive Computing,* vol. 2, no. 3, pp. 42-51, 2003.
[15] T. Hofer et al., "Context-Awareness on Mobile Devices—The Hydrogen Approach," *Proc. Hawaii Int'l Conf. System Sciences,* 2003.
[16] O. Riva, "Contory: A Middleware for the Provisioning of Context Information on Smart Phones," *Proc. Conf. Middleware,* 2006.
[17] E. Shih et al., "Wake-on-Wireless: An Event Driven Energy Saving Strategy for Battery Operated Devices," *Proc. ACM MobiCom,* 2002.

[18] J. Sorber et al., "Turducken: Hierarchical Power Management for Mobile Devices," *Proc. MobiSys,* 2005.
[19] A. Rahmati and L. Zhong, "Context-for-Wireless: Context-Sensitive Energy-Efficient Wireless Data Transfer," *Proc. MobiSys,* 2007.
[20] S. Chakraborty et al., "On the Effectiveness of Movement Prediction to Reduce Energy Consumption in Wireless Communication," *IEEE Trans. Mobile Computing,* vol. 5, no. 2, pp. 157-169, Feb. 2006.
[21] S. Cui et al., "Energy-Efficiency of MIMO and Cooperative MIMO Techniques in Sensor Networks," *IEEE J. Selected Areas Comm.,* vol. 22, no. 6, pp. 1089-1098, 2004.
[22] W. Ye, J. Heidemann, and D. Estrin, "An Energy-Efficient MAC Protocol for Wireless Sensor Networks," *Proc. IEEE INFOCOM,* 2002.
[23] K. Seada et al., "Energy-Efficient Forwarding Strategies for Geographic Routing in Lossy Wireless Sensor Networks," *Proc. Int'l Conf. Embedded Networked Sensor Systems (SenSys),* 2004.
[24] I.H. Witten and E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques.* Morgan Kaufmann, 2005.
[25] G. Anastasi et al., "Performance Measurements of Motes Sensor Networks," *Proc. Int'l Symp. Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM),* 2004.
[26] G. Xing et al., "Minimum Power Configuration for Wireless Communication in Sensor Networks," *ACM Trans. Sensor Networks (TOSN),* vol. 3, no. 2, 2007.
[27] K.L. Wu and P.S. Yu, "Interval Query Indexing for Efficient Stream Processing," *Proc. Int'l Conf. Information and Knowledge Management (CIKM),* 2004.
[28] E. Hanson and T. Johnson, "Selection Predicate Indexing for Active Databases Using Interval Skip Lists," *Information Systems,* vol. 21, no. 3, pp. 269-298, 1996.
[29] J. Lee et al., "BMQ-Index: Shared and Incremental Processing of Border Monitoring Queries over Data Streams," *Proc. Conf. Mobile Data Management (MDM),* 2006.
[30] KAIST UFC Project, http://ufc.kaist.ac.kr, 2008.
[31] HUINS, http://www.huins.com, 2009.
[32] MIT Affective, http://affect.media.mit.edu/areas.php?id=sensing, 2009.
[33] FFTW, http://www.fftw.org, 2009.
[34] *Weka 3: Data Mining Software in Java,* http://www.cs.waikato.ac.nz/~ml/weka/index.html, 2009.
[35] Next Generation Computing Show, http://www.nextcomshow.com/en, 2007.
[36] V. Shnayder et al., "Simulating the Power Consumption of Large-Scale Sensor Network Applications," *Proc. Int'l Conf. Embedded Networked Sensor Systems (SenSys),* 2004.
[37] R.S. Sandhu and P. Samarati, "Access Control: Principles and Practice," *IEEE Comm. Magazine,* 1994.
[38] D. Abadi et al., "Aurora: A New Model and Architecture for Data Stream Management," *Very Large Data Bases J.,* vol. 12, no. 2, 2003.
[39] R. Motwani et al., "Query Processing, Resource Management, and Approximation in a Data Stream Management System," *Proc. Conf. Innovative Data Systems Research (CIDR),* 2003.
[40] S.R. Madden et al., "Continuously Adaptive Continuous Queries over Streams," *Proc. SIGMOD,* 2002.
[41] J. Froehlich et al., "MyExperience: A System for In Situ Tracing and Capturing of User Feedback on Mobile Phones," *Proc. MobiSys,* 2007.
[42] P.J. Lang et al., "International Affective Picture System (IAPS): Instruction Manual and Affective Ratings," Technical Report A-4, Center for Research in Psychophysiology, Univ. of Florida, 1999.
[43] K. Murao et al., "A Context-Aware System that Changes Sensor Combinations Considering Energy Consumption," *Proc. Pervasive,* 2008.
[44] CC2420 Datasheet, http://focus.ti.com/docs/prod/folders/print/cc2420.html, 2009.
[45] S. Kang et al., "SeeMon: Scalable and Energy-Efficient Context Monitoring Framework for Sensor-Rich Mobile Environments," *Proc. MobiSys,* 2008.
[46] V. Chvatal, "A Greedy Heuristic for the Set-Covering Problem," *Math. Operations Research,* vol. 4, no. 3, pp. 233-235, 1979.
[47] P. Slavik, "A Tight Analysis of the Greedy Algorithm for Set Cover," *Proc. Symp. Theory of Computing (STOC),* 1997.
[48] M. Popescu and E. Florea, "Linking Clinical Events in Elderly to In-Home Monitoring Sensor Data: A Brief Review and a Pilot Study on Predicting Pulse Pressure," *J. Computing Science and Engineering,* vol. 2, no. 1, pp. 180-199, Mar. 2008.

**Seungwoo Kang** received the PhD degree in computer science from the Korea Advanced Institute of Science and Technology (KAIST), Daejeon, South Korea, in 2010. He is a postdoctoral researcher at KAIST. His research interests include mobile and ubiquitous computing such as system support for mobile context monitoring and high-performance systems for city-scale ubiquitous services, and Internet services and technologies.

**Jinwon Lee** received the PhD degree in computer science from the Korea Advanced Institute of Science and Technology (KAIST), Daejeon, South Korea, in 2009. He is a postdoctoral researcher at KAIST. His research interests include mobile and ubiquitous computing, data stream processing system, peer-to-peer overlay network, and high-performance Internet cache system.

**Hyukjae Jang** received the BS degree in computer science and electrical engineering from Yonsei University, Seoul, South Korea, in 2003. He is currently working toward the PhD degree in computer science at the Korea Advanced Institute of Science and Technology (KAIST), Daejeon, South Korea. His research interests include mobile and ubiquitous computing such as mobile context monitoring framework, context-aware service, and interaction design for mobile system.



**Youngki Lee** received the BS degree in computer science from the Korea Advanced Institute of Science and Technology (KAIST), Daejeon, South Korea, in 2004, where he is currently working toward the PhD degree. His research interests include mobile and ubiquitous computing systems, system support for context awareness, high-performance systems for city-scale ubiquitous services, and large-scale distributed systems and networking.

**Souneil Park** received the BS and MS degrees in computer science, respectively, from the Soongsil University, Seoul, Korea, in 2004, and the Korea Advanced Institute of Science and Technology (KAIST). His research interests include Internet service and technologies, ubiquitous computing systems.

**Junehwa Song** received the PhD degree in computer science from the University of Maryland at College Park in 1997. He is an associate professor in the Department of Computer Science at the Korea Advanced Institute of Science and Technology (KAIST), Daejeon, Korea. Before joining KAIST, he worked at IBM T.J. Watson Research Center, Yorktown Heights, New York, as a research staff member from 1997 to 2000. His research interests include mobile and ubiquitous computing systems, Internet technologies such as intermediary devices, high-performance Web serving, electronic commerce, and distributed multimedia systems. He is a member of the IEEE.