6-2008

SeeMon: Scalable and Energy-efficient Context Monitoring Framework for Sensor-rich Mobile Environments

Seungwoo KANG

Jinwon LEE

Hyukjae JANG

Hyonik LEE

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

See next page for additional authors

Author
Seungwoo KANG, Jinwon LEE, Hyukjae JANG, Hyonik LEE, Youngki LEE, Souneil PARK, Taiwoo PARK, and Junehwa SONG

SeeMon: Scalable and Energy-efficient Context Monitoring Framework for Sensor-rich Mobile Environments

Seungwoo Kang, Jinwon Lee, Hyukjae Jang, Hyonik Lee,
Youngki Lee, Souneil Park, Taiwoo Park, Junehwa Song
Computer Science, KAIST
Daejeon, 305-701, Republic of Korea
{swkang, jcircle, hjjang, hilee, youngki, spark, twpark, junesong}@nclab.kaist.ac.kr

ABSTRACT
Proactively providing services to mobile individuals is essential for emerging ubiquitous applications. The major challenge in providing users with proactive services lies in continuously monitoring their contexts based on numerous sensors. The context monitoring with rich sensors imposes heavy workloads on mobile devices 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 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.

Categories and Subject Descriptors
K.8 [**Personal Computing**]: General; C.5.3 [**Microcomputers**]: Portable devices; C.3.3 [**Special-Purpose and Application-based Systems**]: Microprocessor/microcomputer applications

General Terms
Design, Experimentation, Measurement, Performance.

Keywords
Context monitoring, Sensor-rich mobile environment, Context Monitoring Query (CMQ), Shared and incremental processing, Sensor control, Essential Sensor Set (ESS).

1. INTRODUCTION
Proactively providing services to mobile individuals is essential for emerging ubiquitous applications. Situational provision of services without user intervention requires an involved process for acquiring individuals' contexts. Individual users have different service requirements and preferences such as the system's level of proactiveness and users' privacy concerns. Applications require different types of contexts in different degrees of awareness. Personal sensor networks will become increasingly complicated, composed of heterogeneous sensors with diverse capabilities, and densely deployed on users' bodies or in their personal area. Future services will require much broader coverage and higher accuracy in recognized contexts. An effective personal mobile system must continuously process a large volume of contexts while supporting a number of concurrent applications.

In this paper, we propose SeeMon, a scalable and energy-efficient context monitoring framework for sensor-rich and resource-limited mobile environments. A major challenge in providing users with proactive services lies in monitoring their contexts continuously. More important, the context monitoring in a sensor-rich environment imposes heavy workloads on personal mobile devices such as PDAs and mobile phones. These devices are often limited in computing and battery power. Running on such devices, SeeMon effectively performs context monitoring involving numerous sensors and applications. On top of SeeMon, multiple applications simultaneously operating on the device can understand the contexts of users and serve them appropriately.

The key to the proposed framework is twofold. First, the context monitoring in SeeMon focuses on the continuous detection of context changes. Note that this semantics is different from conventional context recognition, which only identifies the current context. Once a change is identified, it is not necessary to recognize and notify the same context redundantly as long as it remains unchanged.

Second, while conventional context processing occurs in a uni-directional fashion, SeeMon approaches the context monitoring problem in a bi-directional way. In the uni-directional approach described in Figure 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. The change detection is made at the last stage of the pipeline. However, the bi-directional approach in Figure 2 forms a feedback path in the pipeline. This approach gives an 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 requests and the low-level status of sensor resources. This makes it possible to elaborate 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 Figure 2, in our approach, the context change is detected directly upon feature

MobiSys'08, June 17–20, 2008, Breckenridge, Colorado, USA.

data without going through the expensive context recognition stage. In addition, detailed resource status can be dynamically analyzed considering application requirements. Thus, resources can be intelligently allocated to save energy or increase utilization. The reflection of each high level requirement is performed only once whereas the savings in computational or energy cost are constantly achieved throughout successive monitoring operations. 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.

1.1 Bi-directional 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 Figure 1 and develop the proposed framework based on three observations.

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 which 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 mechanisms 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.

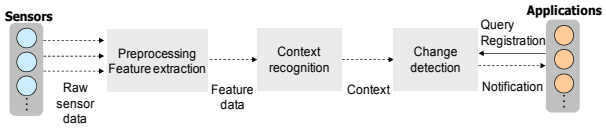


Figure 1. Conventional context monitoring process

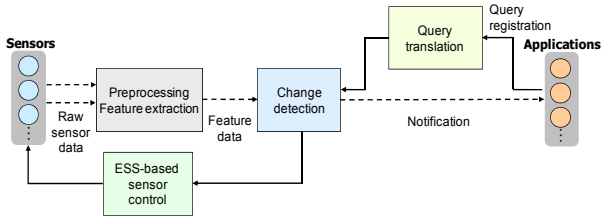


Figure 2. Bi-directional approach to context monitoring problem

Based on these observations, we develop three methods for context monitoring: CMQ (Context Monitoring Query) translation, shared and incremental CMQ evaluation, and ESS (Essential Sensor Set) 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 light-weight 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. Finally, we develop sensor control policies which can be alternatively used to cope with various environments and operational situations.

1.2 Implementation and Evaluation

We have implemented a SeeMon prototype as a prototype system with core components for scalable and energy-efficient context monitoring. We have also built two ubiquitous computing applications that use SeeMon for context monitoring. In order to examine heterogeneous mobile environments, we have been deploying and testing the prototype system on various types of mobile devices along with diverse sensors. We demonstrated the developed system with an example application, SympaThings which enables interactive objects to sympathize with user's affective state, in a public exhibition.

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 2,100 data samples per second. Also, SeeMon reduces the number of wireless data transmissions by more than 60% while evaluating 4,000 CMQs.

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 sample applications. Section 7 shows experimental results. Finally, Section 8 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 various applications which utilize diverse contexts generated from numerous sensors in 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 middleware is 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 middleware targets mobile devices [14][15][16], but does not consider devices with tens/hundreds of BAN/PAN sensors or devices' processing and power limitations. Moreover, they do not focus on continuously detecting context changes.

Recently, 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.

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.

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 the 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.

3. CONTEXT MONITORING FRAMEWORK OVERVIEW

3.1 Motivating Environment

The rapid advance of device and mobile 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]. 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, 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 even finer movements such as delicate hand motions and 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 in the environment. This introduces new technical obstacles for future ubiquitous services, which will be compounded by the resource limitations and heterogeneity of the sensors and mobile devices.

3.2 Context Monitoring Query

SeeMon provides *Context Monitoring Query* (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 (¬) 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 conditions: T → F and F → T. For instance, a condition F → T means that SeeMon gives a notification when the state of the context condition changes from false to true. We are developing more types of alarm conditions to support a wider range of monitoring semantics such as delivering an alarm event when a context condition remains true or false for a period of time.

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 F → T
DURATION 120 DAYS
```

3.3 Architecture

SeeMon is a middle-tier framework between personal context-aware applications and a personal sensor network (see Figure 3). SeeMon provides programming APIs and a run-time environment for applications. Multiple applications that require context monitoring can be developed through the APIs and can run on top

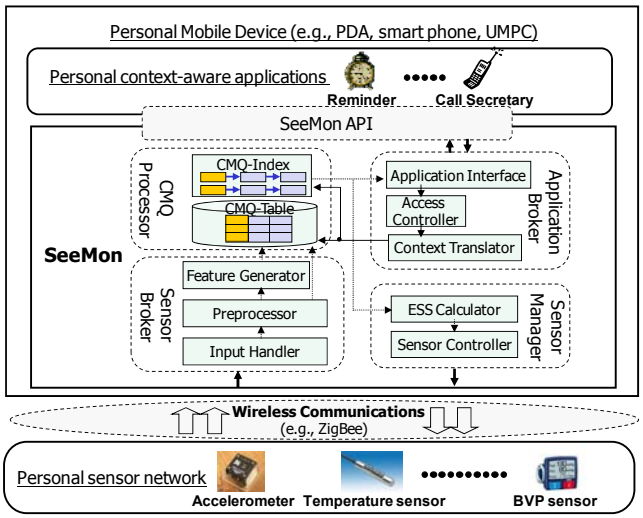


Figure 3. Architecture of SeeMon

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 ZigBee, Bluetooth, or 6LoWPAN can be used.

In our architecture design, SeeMon directly performs context monitoring on mobile devices like PDAs and smart phones. As a design alternative, we can consider server-based context monitoring, in which a server processes sensor data for context monitoring and the mobile device only relays sensor data to the server and receives monitoring results. However, for context-aware applications, especially based on personal contexts, our design choice makes more sense than the server-based approach[1] in two ways: privacy and network cost. First, the mobile device-based approach avoids the exposure of private context, whereas the server-based one must be carefully designed for privacy protection. Second, the mobile device-based approach does not incur continuous mobile networking costs. Even though our approach requires processing in a resource-limited mobile device, we carefully address this problem, supporting processing-efficient context monitoring.

SeeMon consists of four components: the *CMQ Processor*, the *Sensor Manager*, the *Application Broker*, and the *Sensor Broker*. The CMQ Processor is responsible for scalable context monitoring. It efficiently evaluates numerous CMQs over a continuous stream of sensor data. The Sensor Manager enables SeeMon to achieve a high level of energy efficiency. It dynamically controls sensors to avoid unnecessary data transmissions. The Application Broker manages interactions with applications and the Sensor Broker deals with communication with various heterogeneous sensors.

Based on these components, the operation of SeeMon is performed in three phases: query registration, query processing

[1] Note that there are certain types of context-aware applications that require a server and a mobile device to cooperate with each other to provide services such as services based on context information derived from aggregated data from multiple individuals or environments.

and sensor control. First, applications initiate context monitoring by registering CMQs to the CMQ Processor through the Application Broker. Then, the CMQ Processor performs context monitoring by evaluating 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. Details of each component are described as follows.

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, private context information can be exposed to other individuals. Thus, it is important to provide an appropriate control mechanism for privacy and security. In our current design, the Access Controller utilizes an ACL-based approach [37], checking whether a requesting application is registered in an access control list (ACL). The Context Translator translates a CMQ issued by a permitted application into a feature data-level CMQ (see Section 4.1 for details). 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 data from sensors. 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. Since different types of feature and computation are needed to analyze sensor data, the Sensor Broker is designed to be extensible enough to incorporate many types of sensors. Some sensor data is used directly by the CMQ Processor after preprocessing without feature generation (e.g., temperature and humidity data). For simplicity, we will regard all the output data from the Sensor Broker as feature data in the following sections.

The Sensor Manager consists of the *ESS Calculator* and the *Sensor Controller*. The ESS Calculator discovers an *Essential Sensor Set (ESS)* necessary to evaluate CMQs and identifies unnecessary sensors based on the evaluation results of the CMQ Processor. As described in Section 5.1, we abstract ESS calculation as a variation of *Minimum Set Cover problem* and employ a practical heuristic solution. Based on the calculated ESS, the Sensor Controller sends selected sensors control messages to reconfigure the sensors to stop transmitting data. This sensor control is performed whenever the result of any CMQ changes.

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 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 and Bayesian network evaluation 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. (Data may show sudden changes from time to time; however, we believe that changes are more often gradual, especially in the streams of physical data.) 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 pre-computes the queries whose states

change at each value range. The CMQ-Index also partitions the 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 pre-computed 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 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]. Note that currently we consider features derived from a single sensor, although features derived from multiple sensors can be incorporated. Second, SeeMon transforms a context value to numerical data value ranges for corresponding features[2]. For example, (noise == Quiet) can be mapped to (20dB $\leq$ sound pressure level $\leq$ 30dB). 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. Figure 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.

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

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

Context-level semantic		Data-level semantic						
Type	Value	Feature1			Feature2			...
		ID	Low	High	ID	Low	High	...
location	Playground	longitude	36°22'04	36°22'05	latitude	127°21'56	127°21'57	
temperature	Hot	Temp.	28 °C	38 °C				...
...	...							

Figure 4. An example of context translation map

mappings between context-level semantics and data-level semantics, application developers can create customized maps. It is very useful to satisfy the different need 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*. First, the CMQ-Table stores CMQs using a hash structure, providing $O(1)$ lookup time. It contains three attributes: *query id*, *state* (evaluation result), and *context element list* (see Figure 5). 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 is unavailable due to 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 the examples in Figure 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 *RS (Region Segment) 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 Figure 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 indices [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., $B = \{b \mid b$ is either l_i or u_i 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

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

$< b_1 < ... < b_m$. An RS list is a list of RS nodes, $<N_1, N_2, ..., N_m>$. 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 Figure 5, two RS lists are shown as an example. The upper RS list is built for six context elements, CE(Q_1), …, 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 two 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 and notifies the applications that issued the CMQs.

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 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 a 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.

CMQ-Table

Query ID	State	Context Element List
Q_1	false	[F_1, (b_0, b_1), false], [F_2, (b_4, b_6), false] [F_4, (b_4, b_7), true], [F_8, (b_0, b_3), false]
Q_2	false	[F_1, (b_0, b_2), false], [F_3, (b_3, b_4), false] [F_6, (b_5, b_8), undecided]
Q_3	true	[F_1, (b_1, b_5), true], [F_2, (b_2, b_4), true]
Q_4	undecided	[F_1, (b_2, b_3), true], [F_5, (b_0, b_5), true] [F_6, (b_3, b_5), undecided]
...	...	...

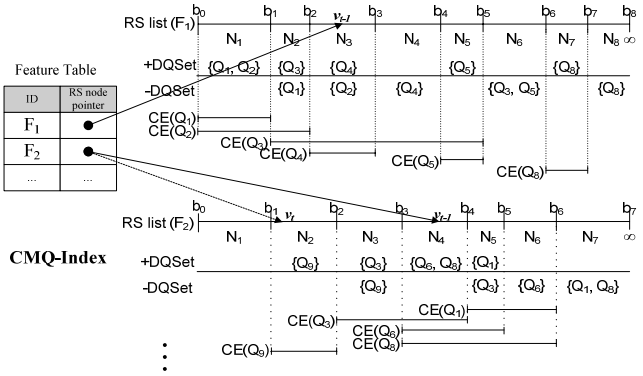


Figure 5. An example CMQ-Table and CMQ-Index:

CMQ-Table shows four CMQs, $Q_1 \sim Q_4$, and their states as well as the lists of included context elements. 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 8 RS nodes, $N_1 \sim N_8$.

If $j = h$, $QSet^+ = QSet^- = \phi$

If $j < h$, $QSet^+ = [\bigcup_{i=j+1}^{h} +DQSet_i] - [\bigcup_{i=j+1}^{h} -DQSet_i]$

$QSet^- = [\bigcup_{i=j+1}^{h} -DQSet_i] - [\bigcup_{i=j+1}^{h} +DQSet_i]$

If $j > h$, $QSet^+ = [\bigcup_{i=j}^{h+1} -DQSet_i] - [\bigcup_{i=j}^{h+1} +DQSet_i]$

$QSet^- = [\bigcup_{i=j}^{h+1} +DQSet_i] - [\bigcup_{i=j}^{h+1} -DQSet_i]$

In Figure 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, $\pm DQSet$ are 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.

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

(v_i is i^{th} 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 ($2 \times N_q$ / 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 the *Essential Sensor Set* (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 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.

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

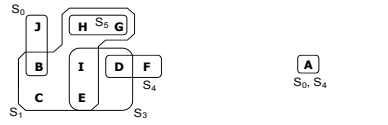
(a) Sensor set S — *U*: undecided

S = {S_0, S_1, S_2, S_3, S_4, S_5}

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

Bold character: false feature

(b) Query set



(c) False-state query set **F-QSet = {B,C,D,E,F,G,H,I,J}**

(d) True-state query set **T-QSet = {A}**

Figure 6. An example of ESS problem

The ESS problem consists of two sub-problems: to find essential sensors for true-state and undecided-state CMQs and to find the essential sensors for false-state CMQs. Figure 6 shows an example of ESS problem for a set of sensors and CMQs. Only query A is true. Thus, features F_0 and F_5 have to be monitored since they are related to the context elements of A. Accordingly, sensor S_0 and 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 F_0 or F_1. Thus, we can put either S_0 or S_1 into an inactive state. Similarly, other CMQs can be evaluated using a small number of sensors. Sensor S_0, S_1 and 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 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.

Minimum Cost False-query covering Sensor Selection Problem:

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 $\bigcup_{i=1}^{k} F\text{-}QSet'(S'_i)$ 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 become 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 which are false by the sensors of S, and define each sensor $S_i \in S$ to be the set of false-state CMQs which 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., $\sum_{i=1}^{k} COST(S'_i) < w$, 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 and Sensor Control

Figure 7 shows the ESS calculation process. The ESS is computed through two steps: computing required sensors for CMQs in a true or undecided state, and then for CMQs in a false state. We call the sensors required for true-state CMQs and undecided state CMQs the TQCover and UQCover, respectively. Including TQCover and UQCover in the ESS in advance can reduce the overhead while computing sensors for the MCFSS problem, 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 such CMQs from the problem space of MCFSS, *F-QSet*.

As the MCFSS problem is NP-complete, we employ a heuristic algorithm, Greedy-MCFSS (see Figure 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. 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., $\frac{COST(S_i)}{|F\text{-}QSet'(S_i) \cap F\text{-}QSet - F\text{-}QSet'(M)|}$, 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 a MCFQCover whose cost is guaranteed to be no more than $\log |F\text{-}QSet|$ times the cost of an optimal solution. It is intuitive to see that the time complexity of the algorithm is $O(|S|^2)$ in the worse 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.

The Sensor Controller controls sensors based on the calculated ESS. It sends a control message to the sensors that are not included in the ESS. The message configures the sensors to stop transmitting data. Afterwards, the ESS Calculator updates the state of context elements related to the controlled sensors in the CMQ-Table. Specifically, it changes the state of those context elements to undecided. On the other hand, the Sensor Controller sends a different type of control message to sensors that are newly included in the current ESS. When a sensor receives this message, it is reconfigured to transmit data.

// 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. $TQCover, UQCover, TUQcover, RF\text{-}QCover \leftarrow \varnothing$
2. for $\forall\ q_i$, where $q_i \in T\text{-}QSet$,
   $TQCover \leftarrow TQCover \cup q_i.sensor$
3. for $\forall\ q_i$, where $q_i \in U\text{-}QSet$,
   $UQCover \leftarrow UQCover \cup q_i.sensor$
4. $TUQCover \leftarrow TQCover \cup UQCover$
5. $RF\text{-}QSet \leftarrow F\text{-}QSet$
6. for $\forall\ s_i$, where $s_i \in TUQCover$,
   for $\forall\ q_i$, where $q_i \in F\text{-}QSet$,
   if q_i evaluates to false by sensor s_i
   $RF\text{-}QSet \leftarrow RF\text{-}QSet - \{ q_i \}$
7. for $\forall\ q_i$, where $q_i \in RF\text{-}QSet$,
   for $\forall\ s_i$, where $s_i \in q_i.sensor$,
   if q_i evaluates to false by sensor s_i
   $RF\text{-}QCover \leftarrow RF\text{-}QCover \cup \{ s_i \}$
8. Greedy-MCFSS (*RF-QSet*, *RF-QCover*)

Figure 7. ESS calculation algorithm

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

Figure 8. Greedy-MCFSS algorithm

5.3 Sensor Control Policy

For sensor control based on the ESS, we carefully consider the ESS computation overhead. The ESS needs to be calculated whenever the evaluation result of any CMQ changes. Such a frequent ESS computation may be burdensome even with a heuristic-based algorithm. To address this problem, we propose two different policies for sensor control: an aggressive policy and a conservative one. The aggressive one is the default policy. It aims to maximize energy saving. In contrast, the conservative policy is designed to reduce the processing cost while sacrificing some energy efficiency. The conservative policy will be most

effective when mobile devices' computing power is limited or processing overhead is high due to numerous CMQs.

Under the aggressive policy, the ESS Calculator continuously updates the ESS to find the most cost effective sensors. On the other hand, under the conservative policy, the ESS Calculator delays ESS computation to reduce the processing overhead. It calculates only TQCover and UQCover to identify necessary sensors for correct CMQ evaluation. While the ESS computation is being delayed, sensors can be added to the TQCover and UQCover and become active. However, to maintain a certain level of energy efficiency, the ESS must be updated before too many sensors are activated. Thus, the ESS Calculator monitors the ratio of active sensors using a metric called the *Sensor Turn-on Level* (STL) defined below. To drop the number of active sensors, it updates the ESS when the STL goes beyond a predefined threshold value.

$STL = N_{inactive \rightarrow active} / N_{inactive}$

($N_{inactive}$ 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 last ESS calculation.)

Such configurability is important to adapt SeeMon to various processing capacity and battery power constraints. We are currently developing a method that performs automatic adaptation. The method dynamically changes the STL threshold value to cope with different throughput demands. To maximize energy efficiency, the default STL threshold value is set to 0, or the equivalent of the aggressive policy. When the number of CMQs becomes larger than the currently achievable throughput, SeeMon automatically increases the STL threshold value. While increasing the value, the method tries to find a value to meet the requested throughput without sacrificing too much energy efficiency.

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. Currently, the prototype is implemented in C++ on a Linux. The total lines of prototype system code are about 8,700. We have been deploying the prototype on various types of mobile devices such as smart phones and wearable devices. In addition, we continue to incorporate diverse and numerous sensors to support rich and fine-grained context specifications. Multiple application developers have used our prototype system and considered it effective, efficient, and stable. Furthermore, we demonstrated an application called SympaThings on top of SeeMon at the Nextcom Show 2007 [35], one of the biggest IT exhibitions in Korea, held in Seoul in November 2007.

Figure 9. H/W setup

Table 2. Sensor-Feature-Context

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 (12)	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
S/W sensor (timer)	-	Time	0.1 Hz	Time (8)	Dawn, Noon
S/W sensor (indoor location)	manual input	Indoor location	1 Hz	Indoor location (12)	1st floor lobby, Room 2432

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 1GB RAM, and (2) a custom-designed wearable device with Intel® PXA270 processor[3] and 128MB RAM. The former represents powerful future mobile devices and the latter a relatively resource-limited current mobile device. We plan to port our system to widely used smart phones as well. Figure 9 shows a snapshot of currently used hardware including the two mobile devices and sensors.

The diversity and scale of sensors determine the coverage and accuracy of context monitoring of SeeMon. From this viewpoint, we have been incorporating as many as sensors that a person can carry. Table 2 shows the sensors that we used in our current prototype. We prefer small-size controllable sensors with processing and wireless communication capabilities appropriate for mobile environments. Such sensors can be deployed in a wearable or a carry-able form and adopt the sensor control mechanism of SeeMon easily. Considering this, 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[4], CC2420 RF module supporting 2.4GHz 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. The node receives sensor data from other sensor nodes and forwards the data to a mobile device. Also, it transmits control messages to the sensor nodes on behalf of the mobile device.

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 BVP (Blood Volume Pulse) sensor and a GSR (Galvanic Skin

[3] This processor supports flexible clocking from 104 to 624 MHz.

[4] This processor supports maximum 8 MHz 8MIPS CPU Clock.

Response) 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 in-door 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 multi-thread system for performance. 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 inter-component communication is performed through message queues. To support frequent data transfer from the Sensor Broker to the CMQ Processor, we used double-buffering. Currently, we are extending the prototype to include an advanced thread scheduler and queue management mechanism to further improve system performance.

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. Fourth, to get the strain feature from GSR data, we implemented simple technique to analyze the magnitude, relative to normal conditions, of GSR signals. Finally, we implemented simple conversion functions to compute features from other USS-2400 sensors such as illumination and temperature features.

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 afterwards. By effectively reusing the results, we reduced the number of set operations. In particular, we designed a bit-map like data structure to store the detailed information of false-state CMQs. It improves ESS calculation performance significantly.

Figure 10. Running Bomber

Figure 11. SympaThings

6.3 Application Development

Emerging areas such as ubiquitous 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.

Running Bomber is the first step toward applying the SeeMon framework to ubiquitous games (U-games). U-games utilize users' various contexts and reflect their physical actions from their everyday activities. Running Bomber is a U-game designed to make treadmill running less boring. Figure 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 U-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. Figure 11 shows the demonstration of SympaThings at Nextcom Show 2007.

7. EXPERIMETS

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 for 12 hours in campus. The eight sensors were a light sensor, a temperature /humidity sensor, three 2-axial acceleration sensors, a GPS sensor, and two software sensors for time and indoor location (see Table 2 for details). The total data rate was 291.74 Hz. Feature data was generated from the sensor data with the rate of 68.06 Hz. We implemented a simple data sender to replay and feed the collected

data to SeeMon. Thus, we were able to conduct our experiments multiple times under the same data workload.

We synthetically generated CMQ workloads to simulate numerous CMQs registered by multiple applications. They reflected various monitoring conditions on different types of contexts. We generated different sets of CMQs with four parameters: the number of CMQs, the number of context elements per CMQ, the distributions of context types and values in context elements. (see Table 2 for the possible context types and values) The default CMQ settings were four context elements per CMQ with uniform distributions for selecting context types and values.

For all experiments, we ran SeeMon on the UX27LN UMPC. We scaled down the CPU frequency to 200MHz to validate our system under a resource-limited mobile environment[5]. 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 sensor control 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 pre-processes continuously arriving data from sensors, processes the data to recognize contexts, and evaluates monitoring queries to detect specified context changes as shown in Figure 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[6]. Data scale 1 is the data workload 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 2100 samples/sec, is sufficient to represent a large-scale personal sensor network. We use query workloads generated by our default setting.

Figure 12 demonstrates the high level of scalability of SeeMon. First, SeeMon scales well with data scale. Even under data scale 7, SeeMon can process 1400 queries, which is a reasonably large number, given the device's limited computing resources (200MHz CPU) and the high rate of sensor data (2100 samples/sec). 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 benefit mainly comes 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 which can accelerate repeated CMQ evaluation. Consequently, as data scale increases, the gap between SeeMon and the context recognition-based method becomes relatively larger.

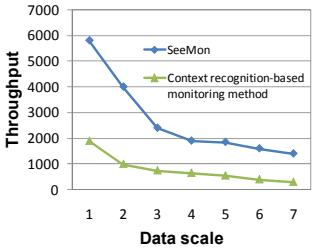


Figure 12. Throughput

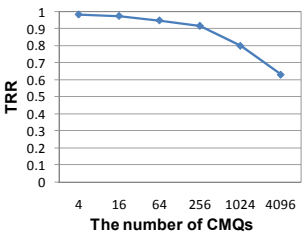


Figure 13. TRR for # of CMQs

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. TRR_i denotes a TRR of a sensor *i*, and TRR_S denotes an averaged TRR of a sensor set S.

$$TRR_i = \frac{ReducedNumberOfTransmission_i}{TotalNumberOfTransmission_i} = \frac{InactiveTime_i \times TransmissionRate_i}{SimulationTime \times TransmissionRate_i}$$

$$TRR_S = \frac{\sum ReducedNumberOfTransmission_i}{\sum TotalNumberOfTransmission_i}, \; i \in S$$

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. Unless specified, the number of CMQs and context elements is fixed to 256 and 4, respectively. Context values follow a uniform distribution. Note that each TRR value is obtained by averaging TRRs of 10 repeated measurements.

Figure 13 shows TRR_S, where s is the whole sensor set in Table 3, as we increase the number of CMQs. SeeMon reduces more than 90% of data transmissions when the number of CMQs is fewer than 256. Even with 4096 queries, more than 60% of data transmissions are eliminated. Such 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

[5] We consider widely used mobile devices, Nokia N95 (330MHz CPU, 64MB of RAM) and Samsung Blackjack (220MHz CPU, 64MB of RAM).

[6] 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 a delay of a couple of seconds is tolerable. Accordingly, acceptable maximum queue size is set to three times of data rate.

the number of true-state CMQs. More true-state CMQs make more sensors active, which decreases TRR.

Table 3 describes the inactive time and TRR_i of each sensor when 16, 256 and 4096 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 TRR_S as increasing the number of context elements in a CMQ. Figure 14 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 Inver-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 4096, and the number of context elements is fixed to 4.

Figure 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 Tradeoff

This experiment shows a tradeoff between processing efficiency and energy efficiency determined by the sensor control policies described in Section 5.3. Such a tradeoff 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 STL threshold values. Note that threshold 0 represents the aggressive policy. Similar to the previous experiment, the data scale 7 is used as a sensor data workload and a query workload is generated with the default setting.

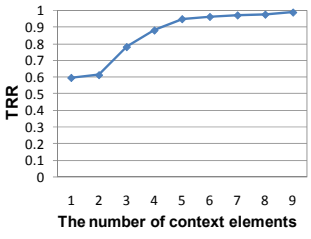
Figure 16 shows a tradeoff between throughput and TRR. As we expected, the aggressive policy (threshold 0) shows the highest TRR, but shows the lowest throughput. As an STL threshold value increases, the TRR linearly decreases, but the throughput increases accordingly. Compared to the aggressive policy, 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 STL threshold value. Thus, the energy efficiency degrades while processing efficiency is enhanced.

8. CONCLUTION

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 bi-directional way. Applying the bi-directional 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.

Table 3. TRR of each sensor

Sensor ID	Name	# of CMQ: 16 Inactive Time(s)	16 TRR	256 Inactive Time(s)	256 TRR	4096 Inactive Time(s)	4096 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	455971	0.9846	30749	0.6640
5		46048	0.9943	37905	0.8185	7519	0.1623
6		462944	0.9996	45003	0.9718	36743	0.7934
7		462938	0.9996	44903	0.9696	32821	0.7087
8		46300	0.9998	45844	0.9899	42142	0.9100



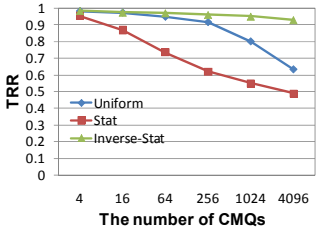


Figure 14. TRR for # of CEs Figure 15. TRR for distributions

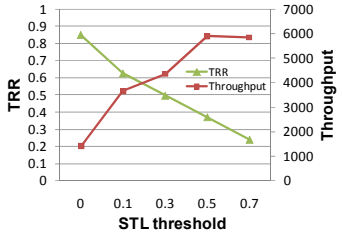


Figure 16. Processing and Energy Efficiency Tradeoff

9. ACKNOWLEDGMENTS

We thank Chulhong Min for developing and demonstrating applications and Yongjoon Son for his initial work on the project. We also thank the anonymous reviewers for their valuable comments. Our special thanks go to our shepherd, Landon Cox, for his valuable suggestions to improve the quality of this paper.

10. REFERENCES

[1] K. V. Laerhoven, A. Schmidt and H. Gellersen, "Multi-Sensor Context Aware Clothing," *In Proc. of ISWC*, 2002.

[2] O. Amft, et al., "Analysis of Chewing Sounds for Dietary Monitoring," In *Proc. of UbiComp*, 2005.

[3] C. Park, et al., "A wearable wireless sensor platform for interactive dance performances," In *Proc. of PerCom*, 2006.

[4] M. Sung, C. Marci, and A. Pentland, "Wearable feedback systems for rehabilitation," *Journal of Neuro Engineering and Rehabilitation*, vol.2, no. 1, 2005.

[5] J.E. Bardram, "Applications of Context-Aware Computing in Hospital Work – Examples and Design Principles," In *Proc. of ACM SAC*, 2004.

[6] T. Sohn, et al., "Place-Its: A Study of Location-Based Reminders on Mobile Phones," In *Proc. of UbiComp*, 2005.

[7] L. Bao and S.S. Intille, "Activity recognition from user-annotated acceleration data," In *Proc. of Pervasive*, 2004.

[8] J. Lester, et al., "A Practical Approach to Recognizing Physical Activities," In *Proc. of Pervasive*, 2006.

[9] P. Fahy and S. Clarke, "CASS – a middleware for mobile context-aware applications," In *Proc. of Workshop on Context Awareness*, *MobiSys* 2004.

[10] T. Gu, et al., "A Middleware for Building Context-Aware Mobile Services," In *Proc. of IEEE VTC*, 2004.

[11] H. Chen, T. Finin, and A. Joshi, "An Ontology for Context-Aware Pervasive Computing Environments," In *Proc. of the Workshop on 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," In *Proc. of the ACM CHI*, 1999.

[13] A. Ranganathan and R. H. Campbell, "A Middleware for Context-Aware Agents in Ubiquitous Computing Environments," In *Proc. of Middleware*, 2003.

[14] P. Korpipää, et al., "Managing Context Information in Mobile Devices," *IEEE Pervasive Computing*, 2003.

[15] T. Hofer, et al., "Context-Awareness on Mobile Devices – the Hydrogen Approach," In *Proc. of the 36th Hawaii Int. Conf. on System Sciences*, 2003.

[16] O. Riva, "Contory: A Middleware for the Provisioning of Context Information on Smart Phones," In *Proc. of Middleware*, 2006.

[17] E. Shih, et al., "Wake-on-wireless: An Event Driven Energy Saving Strategy for Battery Operated Devices," In *Proc. of MobiCom*, 2002.

[18] J. Sorber, et al., "Turducken: Hierarchical Power Management for Mobile Devices," In *Proc. of MobiSys*, 2005.

[19] A. Rahmati and L. Zhong, "Context-for-Wireless: Context-Sensitive Energy-Efficient Wireless Data Transfer," In *Proc. of MobiSys*, 2007.

[20] S. Chakraborty, et al., "On the Effectiveness of Movement Prediction to Reduce Energy Consumption in Wireless Communication," *IEEE Trans. on Mobile Computing*, vol. 5, no. 2, 2006.

[21] S. Cui, et al., "Energy-Efficiency of MIMO and Cooperative MIMO Techniques in Sensor Networks," *IEEE Journal on Selected Areas in Communications*, vol. 22, no. 6, 2004.

[22] W. Ye, J. Heidemann, and D. Estrin, "An Energy-Efficient MAC Protocol for Wireless Sensor Networks," In *Proc. of IEEE INFOCOM*, 2002.

[23] K. Seada, et al., "Energy-Efficient Forwarding Strategies for Geographic Routing in Lossy Wireless Sensor Networks," In *Proc. of 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," In *Proc. of MSWiM*, 2004.

[26] G. Xing, et al., "Minimum Power Configuration for Wireless Communication in Sensor Networks", *ACM Tran. on Sensor Networks* (*TOSN*), vol. 3, no. 2, 2007.

[27] K. L. Wu and P. S. Yu, "Interval Query Indexing for Efficient Stream Processing," In *Proc. of 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," In *Proc. of MDM* 2006.

[30] KAIST UFC project. http://ufc.kaist.ac.kr

[31] HUINS. http://www.huins.com/

[32] MIT Affective, http://affect.media.mit.edu/areas.php?id=sensing

[33] FFTW. http://www.fftw.org/

[34] Weka 3: Data Mining Software in Java. http://www.cs.waikato.ac.nz/~ml/weka/index.html

[35] Next Generation Computing Show 2007. http://www.nextcomshow.com/en/

[36] V. Shnayder, et al., "Simulating the Power Consumption of Large-Scale Sensor Network Applications," In *Proc. of SenSys*, 2004.

[37] R.S. Sandhu and P. Samarati, "Access Control: Principles and Practice," *IEEE Communications Magazine*, 1994.

[38] D. Abadi, et al., "Aurora: A New Model and Architecture for Data Stream Management," *VLDB Journal*, vol. 12, no. 2, 2003.

[39] R. Motwani, et al., "Query Processing, Resource Management, and Approximation in a Data Stream Management System," In *Proc. of CIDR*, 2003.

[40] S.R. Madden, et al., "Continuously Adaptive Continuous Queries over Streams," In *Proc. of SIGMOD*, 2002.

[41] J. Froehlich, et al., "MyExperience: A System for *In situ* Tracing and Capturing of User Feedback on Mobile Phones," In *Proc. of MobiSys*, 2007.

[42] P. J. Lang, et al., "International affective picture system (IAPS): Instruction manual and affective ratings," Tech. Rep. No. A-4, The Center for Research in Psychophysiology in University of Florida, 1999.