

3-2012

MobiCon: A mobile context-monitoring platform

Youngki LEE

Singapore Management University, YOUNGKILEE@smu.edu.sg

S. S. Iyengar

Louisiana State University

Chulhong MIN

Korea Advanced Institute of Science & Technology

Younghyun JU

Korea Advanced Institute of Science & Technology

Taiwoo PARK

Korea Advanced Institute of Science & Technology

See next page for additional authors

DOI: <https://doi.org/10.1145/2093548.2093567>

Follow this and additional works at: https://ink.library.smu.edu.sg/sis_research



Part of the [Software Engineering Commons](#)

Citation

LEE, Youngki; Iyengar, S. S.; MIN, Chulhong; JU, Younghyun; PARK, Taiwoo; LEE, Jinwon; RHEE, Yunseok; and SONG, Junehwa. MobiCon: A mobile context-monitoring platform. (2012). *Communications of the ACM*. 55, (3), 54-65. Research Collection School Of Information Systems.

Available at: https://ink.library.smu.edu.sg/sis_research/2068

Author

Youngki LEE, S. S. Iyengar, Chulhong MIN, Younghyun JU, Taiwoo PARK, Jinwon LEE, Yunseok RHEE, and Junehwa SONG

User context is defined by data generated through everyday physical activity in sensor-rich, resource-limited mobile environments.

BY YOUNGKI LEE, S.S. IYENGAR, CHULHONG MIN, YOUNGHYUN JU, SEUNGWOO KANG, TAIWOO PARK, JINWON LEE, YUNSEOK RHEE, AND JUNEHWA SONG

MobiCon: A Mobile Context-Monitoring Platform

SMART MOBILE DEVICES are the gateway for personal services in the emerging pervasive environment, enabling context-aware applications involving personal sensor networks with sensor devices on the human body and/or surrounding spaces. Diverse sensors function as tools applications use to acquire user context, or current individual status, without user intervention²⁴ (see Table 1); for example, physical contexts (such as heart rate) are recognized through biomedical devices (such as electrocardiogram, or ECG, galvanic skin response, or GSR, and blood volume pulse, or BVP, sensors) and gait is derived through accelerometers and gyroscopes. Likewise, environmental status can be obtained from light/temperature/dust sensors, GPS, RFID,^{3,8} and related networks. Diverse contexts enable mobile applications to proactively provide users customized personal

services. Such applications are emerging in diverse research domains, including health care, elderly support,¹² dietary monitoring, daily life assistance, and sports training.

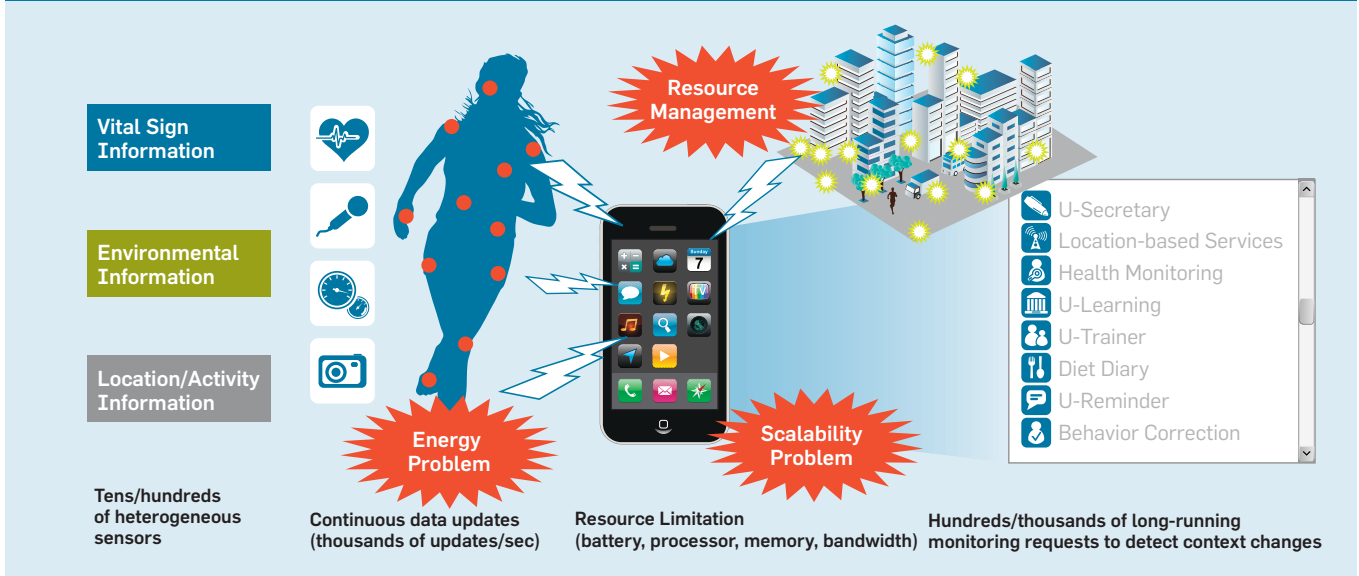
Context-monitoring semantics are different from the context-recognition semantics in early context middleware,^{5,19} aiming to identify a person's context at a specific moment in time. Context monitoring often involves continuous execution of complex, simultaneous, multi-step operations (such as feature extraction and context recognition across mobile and sensor devices). However, individual applications have difficulty performing context-monitoring processing on their own, especially over shared devices with limited resources. A common monitoring platform is essential for effective context monitoring.

Enabling mobile context monitoring involves multiple applications, users, devices, and techniques, as well as different contexts reflecting different degrees of awareness and accuracy. Users have different requirements and preferences for such services, as well as privacy concerns. Efficient resource utilization and management are also important for continuous context monitoring. Enabling diverse context-aware applications requires infrastructure, including a platform to process multiple streams of sensor data and

» key insights

- **Understanding the physical context in users' everyday activity promises a new proactive, automated model for mobile applications beyond conventional user-initiated, reactive services.**
- **MobiCon middleware lets mobile-application developers leverage diverse user contexts without concern for the precision of context recognition or the battery and computational resources of smartphones and other sensor devices in context monitoring.**
- **MobiCon translates physical context into the most energy-and-processing-efficient combination of resources at runtime—the kind of technology needed by practical context-monitoring platforms or applications.**

Figure 1. Challenges in mobile context processing.



coordinate multiple applications. Here, we cover four major areas:

Artificial intelligence and machine learning. Many related efforts recognize user context automatically, focusing on recognition accuracy and context extraction through sophisticated sensors (such as cameras for arm tracking and face-gesture detection). They attempt to address challenges in context recognition (such as preprocessing, feature extraction, segmentation, and recognition algorithms); for example, some systems extract the most useful features necessary for accurate recognition (such as mel-frequency cepstral coefficients for sound modeling for audio and statistical, or frequency-domain features, for detecting physical activity³) from raw sensing data (such as from cameras, microphones, and biomedical and motion sensors). Others automatically segment sensor data or feature data corresponding to valid context vocabularies (such as probability-based segmentation) without user intervention. Most important, diverse context-recognition algorithms have been developed to accurately classify segmented features into defined context vocabularies (such as hidden Markov models for speech and gesture recognition, decision trees for physical-activity recognition,^{3,8} and support vector machine for face recognition). They are effective handling context uncertainty, as most are based on probabilistic inference and reasoning. However, in practice,

applying research systems to real mobile environments is time consuming and raises even more technical challenges. In general, context monitoring should be performed in resource-constrained sensor-enabled mobile platforms, especially those with limited battery and computing power. Continuous processing of complex AI and machine-learning algorithm-based context monitoring imposes a heavy workload on mobile platforms, limiting their practicality.

Mobile and sensor systems. Previous research addressed the challenges of limited resources on mobile and sensor systems, including: OS and system middleware providing system functionality for executing mobile and sensor applications in a resource-aware/adaptive manner (such as ECOSystem²⁶ and Odyssey¹⁶ for mobile systems and Pixie¹⁴ and Eon²² for sensor systems); component techniques for resource efficiency optimizing resource utilization, especially energy consumption,^{4,21,23,25} and distributed resource management across multiple devices or applications (such as load balancing in sensor networks and adaptive node selection in dynamic networks). Unfortunately, as most such work has not focused on emerging context-monitoring applications, their architectures and mechanisms are not fully optimized for them.

Context-aware applications. Diverse context-aware applications have been proposed in health care and medicine,

sports training, and activity/motion analysis,³ tapping application-specific contexts (such as location, activity, and biomedical response). However, despite providing useful functionality, they are limited as a general platform for multiple applications involving diverse sensors and associated contexts. In addition, application-specific platforms have not fully addressed resource limitations, since they are not designed for sharing multiple applications.

Context middleware. Some projects have proposed middleware to support context-aware applications, aiming to hide the complexity of context inference to help identify contexts of interest. However, they are generally limited in terms of continuous monitoring in sensor-rich, resource-limited environments. Most early examples were designed to run in server environments⁵ so did not deal with the inherent resource constraints of mobile and sensor environments. Some context-aware middleware is designed for mobile devices²⁰ but does not consider the tens or even hundreds of sensors in personal-area networks or the processing and power limitations of mobile devices and sensors.

Research Objective

Our aim is to implement a practical context-monitoring system for context-aware applications in sensor-rich, resource-limited mobile environments (see Figure 1), so it must address the challenges of emerging sensor-rich

Table 1. Example contexts and applications.

Category	Context	Application	Sensor
Health	Fall	Elder care ¹²	Accelerometer, gyroscope, microphone
	Motion, gait	Patient monitoring (such as for Parkinson's disease) ¹⁴	Accelerometer, gyroscope
	Heart condition (such as interbeat interval and heart rate)	Heart monitor, jogging support	ECG sensor, BVP sensor, SpO ₂ sensor
	Calories	Activity-based calorie monitor	Accelerometer
Activity/Status	Affective status	SympaThings ⁸	BVP sensor, GSR sensor
	Activity	Activity-based calorie monitor, activity-based service initiation	Accelerometer, gyroscope, camera, RFID (object-attached)
	Gesture	User interaction ^{17,18}	Accelerometer, gyroscope, camera
	Sleep pattern	Sleep monitoring	Accelerometer, microphone
Location	Outdoor	Navigation, traffic monitoring	GPS, accelerometer
	Indoor	microblogging, mobile advertising, personal logging	WiFi, infrared sensor, ultrasound sensor
Place	Mood, crowdedness, noise, illumination	Recommendation, social networking, party thermometer	Microphone, camera, illuminometer, GPS, WiFi
Environment	Weather	Weather monitoring	Thermometer, hygrometer, anemometer
	Pollution	City pollution monitoring	CO ₂ sensor, O ₃ sensor, S sensor, dust sensor
Sports Training	Golf swing	Virtual golf trainer	Accelerometer
	Swim posture	Wearable swim assistant ²	Accelerometer
	Treadmill	SwanBoat ¹	Proximity sensor, accelerometer
Companion	Number, relationship	Advertising, groupware	Microphone, proximity sensor

mobile environments. First, it must be able to continuously run multi-step computation over mobile and sensor devices to ensure accurate context recognition. Such computation requires significant computing resources, often involving high-rate data sampling from multiple sensors, continuous feature extraction and context recognition over massive amounts of sensor data, and transmission of intermediate results among devices.

Second, it must concurrently support a number of complicated requests from multiple applications as a shared underlying platform, further complicating management of limited resources. It must also be able to resolve potential conflicts among multiple application resources. Finally, it must address the dynamic availability of wearable or space-embedded sensors and their resource status in continuous monitoring for seamless service.

Table 2 lists the resource availability of example sensor nodes. In practice, available computing resources (such as CPUs and memory in sensor devices) are often short of the capacity required for individual context-monitoring requests; for example, a MicaZ wireless sensor mote has an 8MHz CPU and 4KB RAM but is incapable of running a light FFT library, *kiss_fft*,⁹ used to monitor activity. In addition, limited battery capacity could compromise continuous monitoring, possibly resulting in early shutdown of relevant applications. Table 3 lists average energy consumption of example tasks performed on a Knode wireless sensor mote. Given the limited battery capacity of Knode, 250mAh, useful time is about 21.5 hours for fall detection and heartbeat monitoring (the third case in the table). Concurrent execution of additional tasks makes energy consumption even more critical. Lorincz et al.¹⁴ wrote about energy deficiency with the SHIMMER sensor, a coin-size wireless sensor for motion analysis, in which the worst-case lifetime was 9.2 hours for motion analysis, including continuous sampling and logging of accelerometer and gyroscope data, maintaining time sync, and raw sample transmission.

We have, since 2007, been developing MobiCon, a novel mobile context-monitoring framework intended to address the resource challenges of sensor-rich mobile environments in support of context-aware applications. Though the research efforts we outlined earlier provide basic techniques for context monitoring, consideration of other techniques is essential for addressing the emerging interdisciplinary challenges involved in practical context-monitoring systems. MobiCon research can be viewed as an initial step toward bridg-

Table 2. Resource availability of example sensor nodes.

Sensor node	MCU	Comm.	Flash memory	Onboard sensor	Optional sensor	Battery
SHIMMER ¹⁴	TI MSP430	CC2420 (802.15.4, 2.4GHz, 256Kbps)	3GB	Triple-axis accelerometer	Gyroscope, ECG, EMG	250mAh Li-polymer rechargeable
Knode	Atmega128 (8MHz CPU)		1MB	Triple-axis accelerometer, dual-axis gyroscope, light	Gyroscope (to cover 3D), SpO ₂	250mAh Li-polymer rechargeable
USS2400 (MicaZ clone)	(4KB RAM)		None	None	Dual-axis accelerometer, thermometer	2 x 1200mAh alkaline

ing the gap between these efforts for context monitoring. Note that many design details and techniques are covered in Ahn et al.,¹ Kang et al.,⁷ Kang et al.,⁸ and Lee et al.¹¹

Framework Development

The MobiCon middleware framework mediates context-aware applications and personal sensor networks (see Figure 2), providing application programming interfaces (APIs) (see Table 4) and a runtime environment for applications. Multiple applications requiring context monitoring can be developed through the APIs and run concurrently atop MobiCon, with MobiCon functioning as a shell above personal-area networks, providing a neat interface for users receiving and processing sensor data and controlling sensors.

It includes five components: Context Processor, Sensor Manager, Resource Coordinator, Application Broker, and Sensor Broker. Applications initiate context monitoring by registering context-monitoring queries through the Application Broker. The Context Processor performs context monitoring by evaluating the queries over data delivered by the Sensor Broker, with monitoring results then forwarded to applications. In this process, the Sensor Manager finds a minimal set of sensors needed to evaluate all registered queries, forcing unnecessary sensors to stop transmitting data to MobiCon. In addition, the Resource Coordinator mediates potential conflicts among low-power sensor nodes when handling multiple requests from concurrent applications. It also supports adaptation to the dynamic sensor join/leave.

The MobiCon architecture prototype runs on Java based on an Android OS (version 2.3) from Google.

Hardware. Deploying MobiCon requires two sets of hardware—mobile devices and sensors—on three mobile devices—a Sony Vaio UX27LN with Intel U1500 1.33GHz CPU and 1GB RAM, a custom-designed wearable device with Marvell PXA270 processor and 128MB RAM, and a NexusOne smartphone with 1GHz Scorpion CPU and 512MB RAM.

We incorporated as many sensors as we could to increase MobiCon’s cover-

Figure 2. MobiCon architecture.

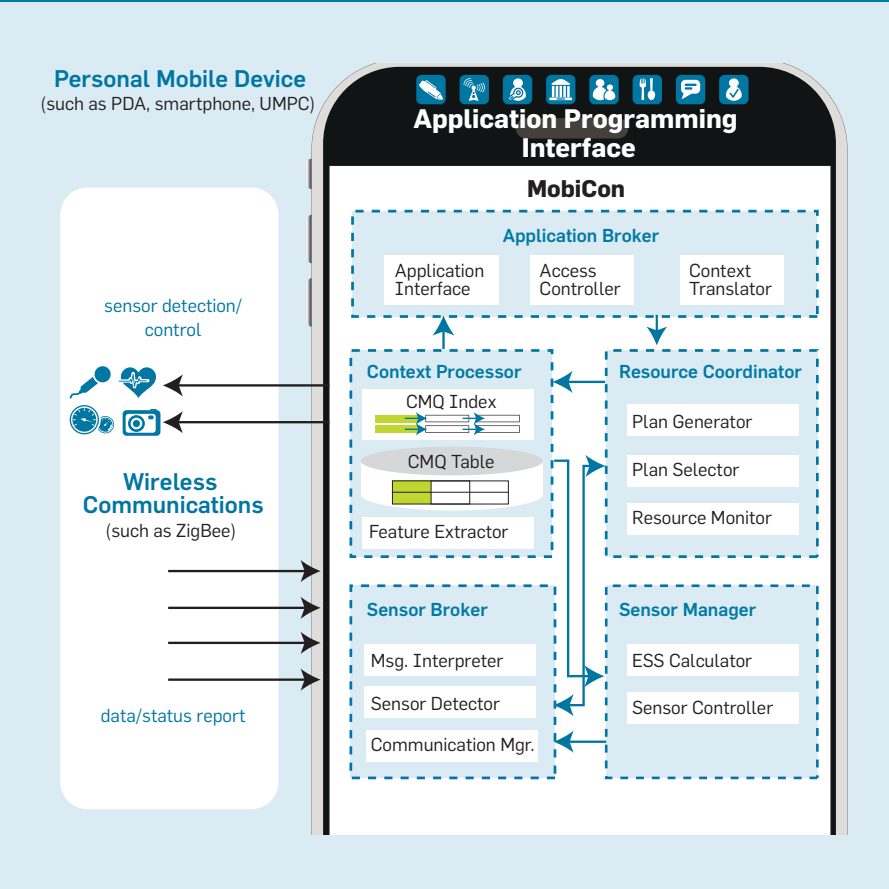


Table 3. Energy consumption of applications on a watch-type Knode sensor mote.

Application	Tasks running on Knode	Avg. energy consumption (mJ/s)
Fall detection 1	Sensing: 30Hz from 3D accelerometers Feature extraction: "norm" Transmission: 6Hz to a mobile device	3.6
Fall detection 2	Sensing: 30Hz from 3D accelerometers and 2D gyroscopes Feature extraction: FFT with window size of 64 and sliding window of 32 Transmission: 1.07Hz to a mobile device	13.1
Fall detection 1 + heartbeat monitor	Tasks for Fall detection 1 + Sensing: 60Hz from SpO2 sensor Feature extraction: peak detection and count Transmission: 1Hz to mobile devices	33.8
Base energy consumption (no task)		8.8

Table 4. MobiCon API.

Functionality	API List
Context monitoring-related APIs	registerCMQ (CMQ_statement)
	deregisterCMQ (CMQ_ID)
Query translation map-related APIs	createMAP ([Parent_Map_ID])
	deleteMAP (Map_ID)
	browseMAP ()

age and accuracy of context monitoring. We selected small-size controllable sensors with processing and wireless communication capabilities appropriate for mobile environments. Deployed in compact, portable form, the sensors work easily with MobiCon's sensor-control mechanism.

The prototype mainly uses two types of MicaZ clone sensor nodes—USS-2400 and Knode—on eight USS-2400 sensor nodes—four dual-axis accelerometers and two light and two temperature/humidity sensors. We designed Knode to support diverse sensing modalities in a single mote; Knode includes the Atmega 128L microcontroller unit (MCU), or low-power CMOS eight-bit microcontroller based on the AVR enhanced RISC architecture, 1G flash memory, CC2420 RF transceiver supporting 2.4GHz band ZigBee protocol, and TinyOS operating system. Also included are a triple-axis accelerometer, dual-axis gyroscope, and support extensions, as are additional sensors that provide context types not

supported by the MicaZ clone sensor nodes, including three biomedical sensors—ECG, BVP, and GSR—as well as a Bluetooth-enabled GPS sensor to position outdoor location. We also developed two wearable sensor devices with different form factors: Sensor-Bracelet (with basic Knode) and U-Jacket (with BVP and GSR sensors).

Translation-based Approach

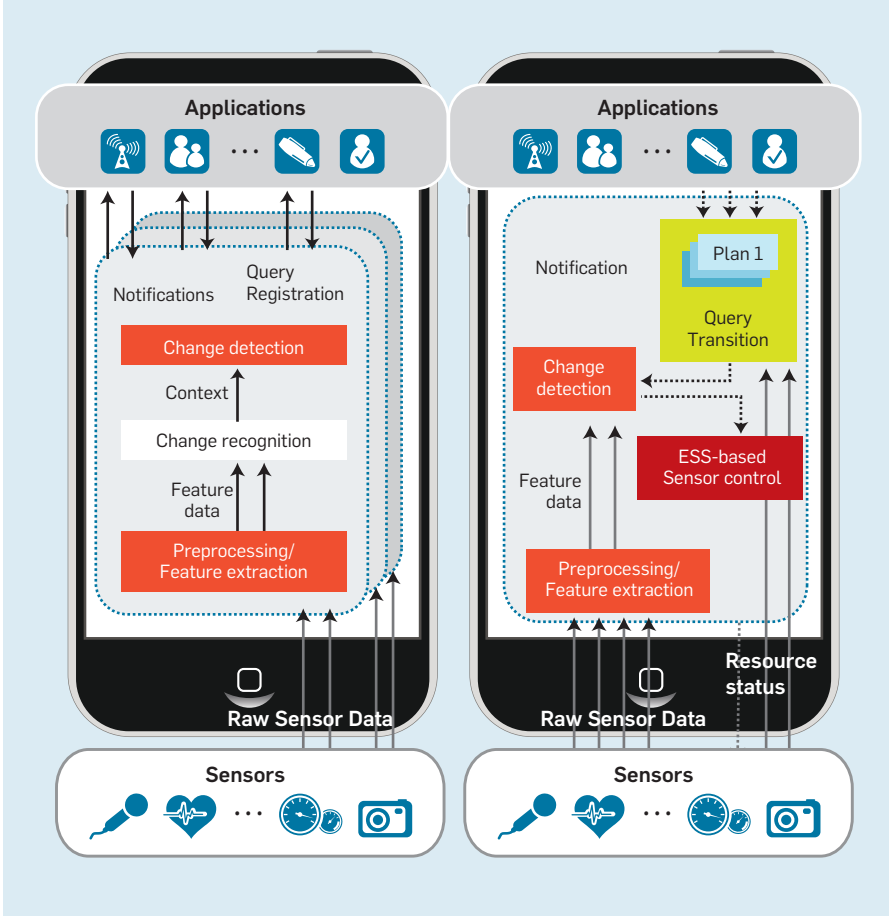
We devised a translation-based approach to facilitate context-aware application development and resource management over dynamic resource-limited sensor-rich environments. Figure 3a outlines the flow of a conventional non-MobiCon context-monitoring process, usually proceeding in one direction through a pipeline with multiple stages: preprocessing, feature extraction, and context classification/inference, as defined by a programmer. Context monitoring is more than a single-step recognition task, continuously recognizing a context of interest and detecting changes

as an ongoing task. In non-MobiCon approaches, change detection is performed in the final step, following final classification of a context. It usually supports a single or small number of monitoring requests through a small number of sensors within a general computing environment with sufficient resources to sense, transmit, and process related data.

Unlike MobiCon, such approaches include a computation-intensive processing pipeline for each context. As a result, they are not efficient enough to handle concurrent monitoring requests on mobile platforms or on sensors with limited resources. Moreover, applications generally use resources in a greedy manner, further straining resource availability. With a limited view of system status, individual applications have more difficulty addressing resource contention. Moreover, they are unable to achieve global efficiency, shared resource use, and computation on their own. Applications must also be able to adapt themselves according to dynamic resource availability. In essence, the fixed processing defined by programmers before an application is put to use restricts diverse attempts to mediate between a system and its applications.

Addressing limited resource coordination in a conventional approach, we proposed a translation-based approach for MobiCon in 2008,⁸ extending it in 2010.⁷ MobiCon-based applications specify the monitoring requests of interest in a high-level language, submitting them to the platform. MobiCon translates each request to a lower-level representation. When translated, the lower-level representation gives the platform a detailed understanding of application requirements, as well as the low-level status of associated sensor resources. In addition, detailed resource status can be analyzed dynamically in light of application requirements. With this understanding, MobiCon supports a feedback path between each application and its lower-level processing and further extends the computational stages in the processing pipeline, saving computational, as well as energy, overhead. MobiCon also takes an active role in orchestrating application requests and system resources, significantly improv-

Figure 3. (a) Non-MobiCon context-monitoring process; (b) Translation-based approach to context monitoring.



ing overall system performance.

This translation-based approach lets each application inform the system of its requirements so it gains a comprehensive understanding of each application, especially in terms of resource and computational demands. Taking a holistic view of system status, including resource availability and running applications, it determines a globally optimized translation for each monitoring request. As a result, resource use is orchestrated among competing applications in highly dynamic and resource-constrained environments; for example, it mediates applications to achieve a systemwide goal (such as energy balancing) not necessarily accomplished through application-level strategies.

Figure 3b outlines the flow of a translation-based context-monitoring process, with the system adaptively determining an appropriate resource-use plan for an application's monitoring request according to dynamic resource availability. Note that the system prepares (in advance) several alternative plans for each monitoring context. Unlike non-MobiCon processes, it supports system-level optimization in both resource use and computation and could also reorganize processing stages so context changes are identified early in the processing pipeline. Early change detection is contrary to the conventional way of detecting changes only after inferring them through an algorithm (such as decision-tree logic). However, such costly operations can be avoided if a change of activity is detected early on through a change in feature values from accelerometers.

In terms of processing efficiency, the MobiCon translation-based approach also helps developers devise a shared, incremental sensor data processor for context processing. Some dependence among seemingly unrelated contexts is often exposed by the translation, with related contexts grouped together. The system can then compose a shared processing operator, through which the grouped requests are evaluated in a single run. Shared processing is achieved at different stages of context monitoring, including context recognition, feature extraction, and sensor reading. The feature-extraction specification also lets MobiCon take advan-

tage of continuity of contexts and locality in sensor readings. MobiCon also delivers an incremental processing method to accelerate the successive monitoring processes exploiting locality in streams of input data.

MobiCon includes an efficient sensor-control mechanism to enhance the energy efficiency of sensors and mobile devices. The key idea is that given a number of queries and sensors, only a subset of sensors may be sufficient for answering context-monitoring queries; for example, a query regarding the context "studying in the library" would be answered without having to activate an accelerometer to recognize user activity, assuming, say, MobiCon knows the user is not in the library. The feature-level specification of requests gives MobiCon a better way to identify the sensors required to process an application's monitoring requests. MobiCon thus develops a sensor-control method to compute and activate a small set of sensors we call the Essential Sensor Set, or ESS, limiting wireless communication between sensors and mobile devices and saving energy.

The translation-based approach also relieves developers from having to devise their own context-recognition methods requiring special expertise. Finally, it enables the sharing of common context vocabularies, spurring quicker development of new personal-context applications.

Application Interfaces

Like MobiCon, any context-monitoring platform must give application developers intuitive, generic context-monitoring APIs. These APIs should facilitate applications to delegate complex context-monitoring tasks to the platform while focusing on application-specific logics (such as UI) beyond context monitoring. Applications need not specify which sensors to use, data to collect, how often to collect the data, feature-extraction and classification modules to apply, or computing resources to use to execute modules.

As a middleware foundation for context-monitoring APIs, MobiCon includes the Context Monitoring Query, or CMQ, declarative query language supporting rich semantics for monitoring a range of contexts while abstracting device details. It also proactively

detects changes in user context. CMQ specifies three conditions: context, alarm, and duration. Alarm determines when MobiCon delivers an alarm event to an application, and duration specifies the amount of time MobiCon must evaluate a registered CMQ. The following code is an example CMQ

```
CONTEXT (location == library)
        AND (activity == sleeping)
        AND (time == evening)
ALARM F → T
DURATION 120 days
```

and is registered or deregistered by a set of APIs supported by MobiCon (see Table 4).

CMQ translation (bridging logical and physical resources). CMQ translation is a key to using MobiCon's translation-based approach to developing and running context-aware applications. Bridging the gap between high-level (logical) application requests and low-level (physical) resource utilization, CMQ translation enables efficient context monitoring and resource orchestration. The CMQ translation process converts CMQs specified in context-level semantics into range predicates over continuous feature data, along with associated sensors and corresponding resource demands. Here is an example

Original CMQ:

```
CONTEXT (activity == running)
        AND (temp == hot)
        AND (humidity == wet)
ALARM F → T
DURATION 1 month
```

Translated CMQ:

```
CONTEXT (acc1_y_energy > 52)
        AND (acc3_x_dc < 500)
        AND (temp > 86) AND
        (humidity > 80)
ALARM F → T
DURATION 1 month
```

MobiCon maps context types specified in a registered CMQ to one or more features and their associated sensors; for example, it converts an activity context type into multiple features like direct current derived from accelerometers.³ A context value is then transformed into numerical value ranges for corresponding features; for example, “hu-

midity == wet” can be mapped to “80% < humidity.” MobiCon also maps resource demands for computing feature data. Feature computation can be performed on sensor or mobile devices, so corresponding resource demands may be different. They are collected through either offline profiling of tasks or online hardware/program state tracing. MobiCon maintains a context-translation map to support the CMQ translation;⁸ note the cost of translation is negligible since it is a simple one-time operation performed upon query registration.

Resource Coordination

Context monitoring with only a few dynamic and resource-limited sensor devices is a challenge for any system, and greedy, careless use of resources aggravates contention for resources among applications, possibly reducing system capacity. Most sensor devices have less capacity than required for context-monitoring tasks. Moreover, the availability of the devices changes dynamically due to their wearable form, as well as to user mobility. Without system-level support, an individual application might find it impossible to address the challenge of resource management. Applications have only a limited view of resource use of other concurrent applications and cannot negotiate with them for coordinated resource use.

In many mobile and sensor systems (such as Chameleon,¹⁴ Pixie,¹⁵ and Odyssey¹⁶) resource management is based mainly on application-driven decisions involving resource allocation or passive resource-use management. They expose APIs to applications that determine the type and amount of resources required to execute program code and explicitly request resources through APIs; for example, Pixie provides resource tickets (such as <Energy, 700mJ, 10sec>), and Chameleon provides systems calls (such as set-speed()) to control CPU speed directly. They then allocate the requested resources if available. If not, the related applications would change their resource use according to predefined code blocks (such as by trading off data or functional fidelity). However, these approaches to passive resource-use management impose a huge time

and intellectual burden on programmers, and their flexibility cannot be utilized in practice since applications find it almost impossible to estimate dynamic resource status and prepare code blocks for all cases.

MobiCon takes an active resource-use-orchestration approach in which the system explores the use of alternative resources.⁷ The translation-based approach is a key method for enabling active resource coordination at the system level. Each application submits a high-level context specification to MobiCon while delegating the decision of how to process the context through available resources. That is, the approach decouples resource selection and binding from the application’s logical resource demands. The system then actively finds the best combination of resources to process context specifications through current resources and applications.

The MobiCon approach to orchestrating resource use can be characterized in three steps (see Figure 4): MobiCon first generates multiple alternative resource-use plans to process a high-level context from an application; the alternatives result from the diversity of semantic translation. A context can be derived from a variety of processing methods; for example, when the context-quality level required by an application is conditionally tolerable, MobiCon monitors a “running” context through diverse methods (such as direct current and energy features from acceleration data or statistical features from GPS location data). MobiCon then dynamically and holistically selects a set of plans to execute at runtime, reflecting resource availability, application requests, and system-level policy. MobiCon thus resolves contentions and maximizes the sharing of resources in multiple applications. It also supports systemwide policy for resource use (such as minimizing total energy consumption).

Finally, MobiCon continuously changes executed resource-use plans to adapt to the dynamic system environment. A set of plans selected by MobiCon may not persistently ensure optimal use of resources since resource availability and application requests are constantly changing. Upon a change of application request or

sensor availability, MobiCon updates existing plans and reselects plans that result in resource use that still meets systemwide policy under the changed system conditions. Such flexibility enables MobiCon to support multiple applications as long as possible in inherently dynamic environments.

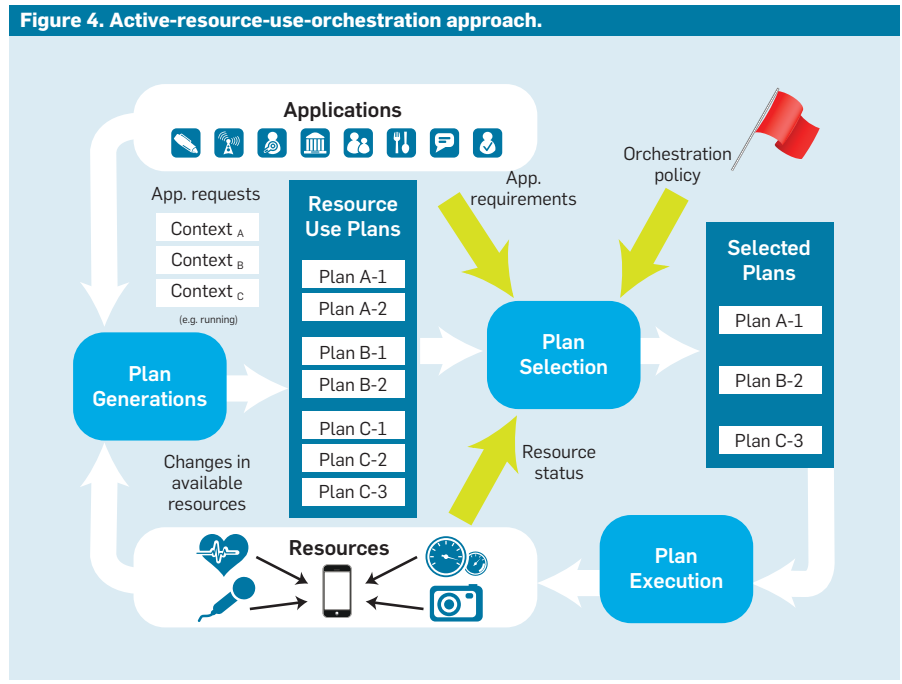
Shared Evaluation of Context-Monitoring Requests

Much research effort has sought to accelerate a single processing pipeline of context recognition consisting of feature-extraction and pattern-classification steps. Here are three representative examples: First, feature-reduction methods were proposed for reducing the number of features that might increase processing time (such as Sequential Forward and Backward Search and Linear Discriminant Analysis). Second, feature quantization is beneficial for reducing the computational cost of context-recognition algorithms through only a few discrete feature values, rather than continuous feature values. Finally, context-recognition algorithms are widely used to optimize the context-processing pipeline; the processing time of non-stochastic algorithms (such as Decision Tree and support vector machine) is much less than that of stochastic algorithms (such as Bayesian networks and neural networks). Hence, developers prefer non-stochastic algorithms if both types

of algorithm produce similar recognition accuracy. While these techniques complement the MobiCon approach, MobiCon is further tuned to speed up multiple processing pipelines for monitoring user contexts.

Addressing multiple processing pipelines, MobiCon is designed to manage shared evaluation of multiple application requests,⁹ looking into requests from different applications and exploring their potential dependence. This system-driven approach represents a significant advantage over application-driven approaches in which each application handles its own requests. Shared evaluation can be exploited in many ways; for example, several applications might want to monitor the exact same context simultaneously. Intermediate results within a processing pipeline, even in different contexts, can be shared, at least in part.

MobiCon also takes advantage of potential dependencies between repeated monitoring processes. The monitoring process for each sensor-data input can be considered not as independent but as part of successive data inputs. Thus, MobiCon’s processing steps can be streamlined to use such dependencies and speed up continuous processes. A useful approach to speed up is exploitation of locality in sensor-data streams. Consecutive sensor readings usually show gradual changes, especially for sensing physi-



cal phenomenon (such as a person’s physical activity and the pollution level of the nearby environment).

MobiCon automatically generates a shared index of context-monitoring requests we call CMQ-Index; upon the arrival of each sensing data, all CMQs are processed with a single evaluation. Incremental processing exploits the locality and consequent overlaps between successive evaluations. To support incremental processing, MobiCon manages the CMQ-Index as a stateful index and remembers the state of the most recent evaluation. The evaluation for new data input proceeds from the stored state.

Consider the following example of CMQ-Index evaluation: A CMQ-Index is created for five CMQs by building a linked list data structure (see Figure 5); MobiCon already translates each CMQ into features and thus has a range (such as $\text{energy} > 52$). The figure shows a simple case, with each translated CMQ containing a single feature type. Rather than evaluate each CMQ separately, a CMQ-Index enables shared evaluation upon arrival of a new feature data value. Moreover, the pointer in the feature table enables incremental processing by remembering the state of the previous evaluation. In Figure 5, the pointer stores the computation ended at N_4 at the last evaluation with data v_{i-1} . With new data v_i , the computation starts from N_4 and proceeds to N_2 . While pro-

ceeding from N_4 to N_2 , state-changed CMQs are quickly identified; that is, CMQ2, CMQ3, and CMQ4 become false, and CMQ5 becomes true. Due to the locality of consecutive feature values, such traversal is often quick, significantly accelerating repeated CMQ-Index evaluation. The CMQ-Index approach outperforms state-of-the-art query indexing mechanisms by orders of magnitude.¹¹ Moreover, the structure is memory efficient, since it stores only the differences between queries over successive ranges without replication.

Energy-Efficient Sensor Use

Low-energy sensors carrying out continuous operations (such as sensing, filtering, feature extraction, and transmission) in context monitoring quickly drain their batteries, significantly compromising the full functioning of context monitoring. Thus, to be practical, any mobile context-monitoring platform, including MobiCon, must be able to achieve energy efficiency in its use of sensors.

Approaches to energy efficiency in sensor and mobile systems in previous research include hierarchical sensor management, energy-fidelity trade-offs, interest-aware sensor management, sampling-rate adjustment, and sensor-data compression. Here we review these approaches, comparing them to the MobiCon solution.

Hierarchical sensor management

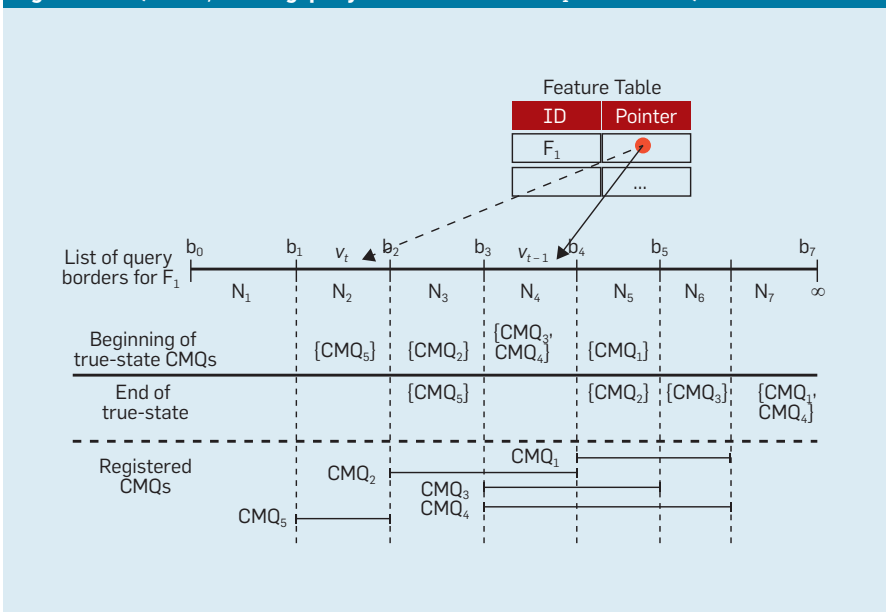
reduces energy consumption by exploiting a low-power tier that consumes less energy, thus avoiding continuous operation of a high-power tier that drains energy. Different types of hierarchical relationships include, for example, Turducken²³ who exploited the hierarchy between low-power devices and high-power devices, while Mercury¹⁴ exploited the hierarchy between different sensors in a single-sensor device; that is, a gyroscope consumes much more energy than an accelerometer.

Techniques based on an energy-fidelity trade-off involve several levels of fidelity associated with different rates of energy consumption and alternatively select an appropriate level for the sake of saving energy; for example, Eon²² allows applications to select from multiple program flows associated with different fidelity levels for different energy states (such as high-power and low-power). Likewise, applications in Levels¹⁰ provide alternative code blocks for different energy levels. Selecting from the alternatives, runtime systems determine an appropriate level of fidelity that can be supported under given conditions of energy availability.

Interest-aware on-the-fly sensor management activates only the sensors necessary to generate contexts or data only when applications are interested, avoiding unnecessary energy consumption by other sensors (such as SeeMon⁸ and EEMS²⁵). Techniques leveraging sampling-rate adjustment reduce the data-sampling rate as much as possible without affecting recognition accuracy, thus saving energy for sensing data.⁴ Moreover, sensor-data compression reduces the amount of transmitted data, saving energy that would otherwise be consumed for radio transmission.²⁰

MobiCon’s solution to energy efficiency involves interest-aware sensor management using a minimal set of sensors to extract the contexts of application interest. MobiCon supports applications without having to activate all potentially available sensors by exploiting the characteristics of personal context and application requirements.⁸ It also employs a novel sensor-control method—the ESS—to leverage the structure of context-monitoring

Figure 5. CMQ-Index, showing query borders for feature F_1 and five CMQs.



queries and their co-dependence. The key idea is that only a small number of sensors is likely to be enough to deliver responses for all context-monitoring queries. The ESS changes depending on the current context and registered monitoring requests. However, a user's context often remains the same once calibrated to the user's particular situation. Likewise, the ESS does not change abruptly.

ESS-based sensor control. Calculating the ESS is complicated, ideally including only a minimal number of sensors to limit overall system energy use. MobiCon's selection of ESS sensors should also consider sensor data-transmission rates.

Consider a CMQ represented in conjunctive normal form—multiple conditional clauses connected through an “and” operation—of context elements. As described earlier, CMQ evaluation aims to detect whether the states of CMQs change, since CMQ semantics aim to deliver the results only upon a change in context. During that evaluation, a CMQ is in one of three states:

True-state. The state of a true-state CMQ changes to false if the state of its single context element changes to false. MobiCon must monitor all context elements to see if the CMQ state changes, and all sensors related to the context elements must be included in the ESS;

Undecided-state. An undecided-state CMQ is managed the same way. MobiCon checks the states of all context elements, and all related sensors must be included in the ESS; and

False-state. Monitoring any single context element in a false state is sufficient, as long as that state is unchanged; only when it does change must the states of the other elements be monitored.

The ESS-calculation challenge involves computing the minimum-cost ESS for the false-state CMQs. However, false-state CMQs are also an opportunity to advance energy efficiency, with MobiCon choosing a false-state context element associated with the most energy-efficient sensors. Formulated as “minimum cost false query covering sensor selection,” or MCFSS,⁸ the problem is NP-complete, proved by reducing another well-known NP-complete

problem, “minimum cost set cover,” or MCSC, to the MCFSS problem. Accordingly, with MobiCon employing a heuristic algorithm—Greedy-MCFSS,⁸ now consider a MobiCon operation example with four sensors— S_0 , S_1 , S_2 , and S_3 —and three queries—A, B, and C—where

$$A = (2 < F_0 < 15),$$

$$B = (F_0 < 5) \wedge (F_1 < 15) \wedge (F_3 < 20), \text{ and}$$

$$C = (12 < F_1 < 18) \wedge (1 < F_2 < 20) \wedge (10 < F_3 < 34).$$

F_0 , F_1 , F_2 , and F_3 are features calculated from the values of S_0 , S_1 , S_2 , S_3 , respectively, and have 7, 31, 2, and 15 as their

current feature values, and A is a true-state CMQ. Thus, sensor S_0 related to the context element of A should be in the ESS and update data. The other queries—B and C—are false; for example, the state of query B can be determined through either S_0 or S_1 . Sensor S_0 and S_1 are therefore sufficient for evaluating all registered CMQs; that is, $ESS = \{S_0, S_1\}$.

Applications

Consider the following three MobiCon-supported applications:

Swan Boat. Designed to make treadmill running less boring, *Swan Boat*,^{1,17}



Figure 6. Applications: (a) *Swan Boat*: (a1) player screenshot, (a2) gameplay; (b) *U-theater*: (b1) *Smash the Beehive!*, (b2) gameplay; and (c) *SympaThings*: (c1) use case, (c2) changing picture images in a frame, (c3) changing lamp colors.

is a step toward applying the MobiCon framework to pervasive games reflecting users' contexts and physical actions (see Figure 6a(1)). Basically, two players in a team collaboratively control a boat, with the difference in running speed between team members determining the boat's direction. Hand gestures are also used as input for additional game interaction. Players can also attack opponents by punching (see Figure 6a(2)). MobiCon simplifies development of pervasive games so game developers need only define game rules and design user interfaces. In *Swan Boat*, MobiCon manages complexities (such as processing acceleration data and recognizing motion) through a simple CMQ registration.

U-theater. U-theater is a group-interactive gaming application for public places, like movie theaters, with large screens. Related games are likely played by dozens of players (see Figure 6b(2)) wearing sensors on their wrists, watching the screen, and interacting through body motion. Leveraging MobiCon and its APIs, developers implement applications without having to know much about sensor-data acquisition, feature extraction, or motion pro-

cessing; consequently, they are better able to concentrate on game content. Three MobiCon-supported games have been developed for U-theater: *Cheer Together!*, *Smash the Beehive!* (see Figure 6b(1)), and *Jump, Jump!* In *Smash the Beehive!*, a beehive is swarming with bees when, suddenly, the bees are gone. Players then punch the beehive, with the quickest to do so winning the game.

SympaThings. Inspired by affective computing and running on wearable devices, the object of *SympaThings* is to get nearby smart objects to "sympathize" with a user's affective context; for example, a picture frame might change the picture it frames, and a lighting fixture might adjust its color (such as red for strain and yellow for ease). Efficient processing is crucial, including high-rate data from BVP and GSR sensors and multiple queries for smart objects. MobiCon's shared, incremental processing is essential for satisfying these requirements. *SympaThings* is a collaboration of the Human Computer Interaction Lab and the Semiconductor System Lab of the Korea Advanced Institute of Science and Technology, Daejeon, Korea.

Experiments

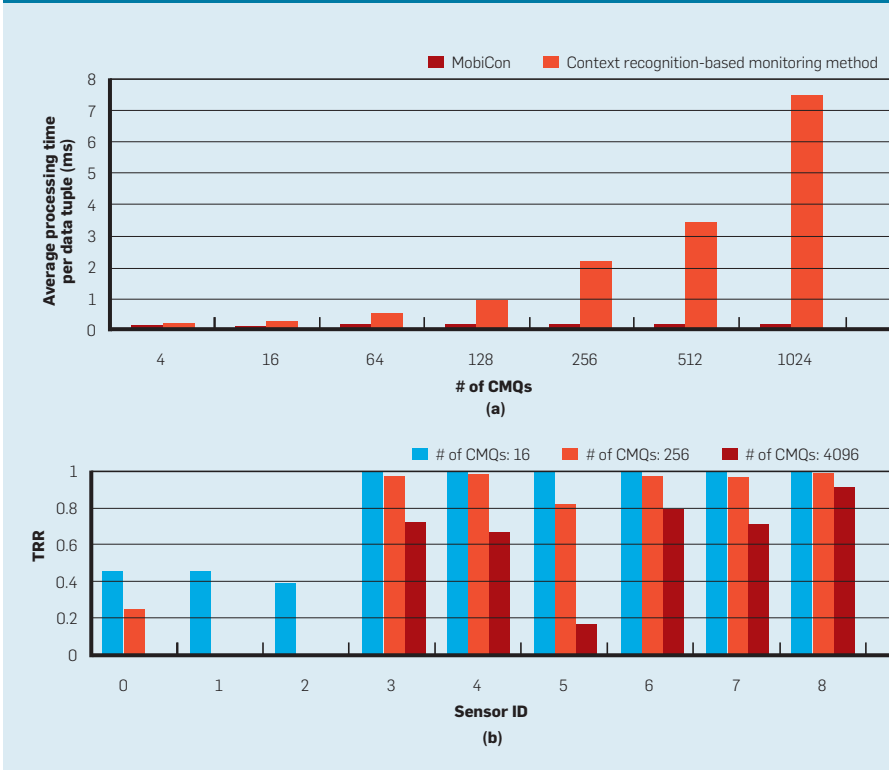
In their experiments, MobiCon developers used a data workload from raw data generated by eight sensors over the course of a student's day on the Daejeon campus. Sensors included five USS-2400 nodes developed by Huins Inc.—a light sensor, a temperature/humidity sensor, and three dual-axis acceleration sensors—along with a GPS sensor and two software sensors for time and indoor locations. The total data rate for all was 291.74Hz. MobiCon developers also synthetically generated CMQs to simulate various monitoring conditions in different contexts. Each CMQ included four context elements, with uniform distributions applied to selecting context types and values in context elements. For all experiments, MobiCon ran on Sony's UX27LN ultra-mobile PC, with CPU frequency scaled down to 200MHz to validate the platform within a resource-limited mobile environment.

CMQ-Index-based context monitoring. Here, we outline the performance benefit of the CMQ-Index-based context monitoring method, comparing its performance against an alternative approach—context recognition-based monitoring—that models non-MobiCon context middleware.²⁰ The context recognition-based monitoring method continuously receives sensor data, processes it to recognize contexts, and evaluates queries to detect specified context changes; individual queries are evaluated separately.

Figure 7a outlines average processing time per data tuple with increasing numbers of registered CMQs. MobiCon shows significant processing efficiency compared to alternative context-recognition-based monitoring platforms. It also scales well with increasing numbers of queries. Processing time of MobiCon for 1,024 queries is orders of magnitude shorter than in the context-recognition-based monitoring approach.

ESS-based sensor control. The performance of ESS-based sensor control demonstrates MobiCon's potential for energy efficiency. As a metric, MobiCon uses the Transmission Reduction Ratio (TRR), defined as a ratio of the reduced number of transmissions to the total expected number of trans-

Figure 7. Experiment results: (a) average processing time per data tuple; (b) TRR of each sensor device with different sensing modules: id 0 (illuminometer); id 1 (thermometer); id 2 (hygrometer); and id 3–8 (accelerometers).



missions of sensor devices, measuring TRR for eight sensing sources deployed on five USS-2400 sensor nodes.


Figure 7b lists the TRR of 10 measurements for each sensing source, with acceleration sensors producing a much higher TRR than other sensors. Due to their high transmission rates, ESS-based sensor-control mechanisms frequently exclude acceleration sensors from the ESS. On average, MobiCon eliminates over 90% of sensor-data transmissions when the number of CMQs is fewer than 256. Moreover, ~63% of sensor-data transmissions are eliminated even with 4,096 queries. As the number of CMQs increases, the TRR decreases because the number of true-state CMQs increases.

Conclusion

The theoretical foundation and implemented technology described here support a number of heterogeneous applications for simultaneously running and sharing limited dynamic resources. We showed that non-MobiCon context-monitoring processes are inadequate for dynamic situations, while MobiCon promises new sensing devices, mobile and sensor system technologies, data processing technologies, and mobile service models, as well as human computer interaction in everyday mobile environments.

Acknowledgments

This work is supported in part by a Korea Research Foundation Grant funded by the Korean Government (KRF-2008-220-D00113), by the Future-based Technology Development Program through the National Research Foundation of Korea funded by the Ministry of Education, Science, and Technology (20100020729), and by the Ministry of Knowledge Economy, Korea, under the Information Technology Research Center-support program supervised by the National IT Industry Promotion Agency (NIPA-2010-(C1090-1011-0004)), the U.S. National Science Foundation (grant # CNS-0963793), U.S. Department of Defense ONR (grant # N0014-08-1-0856), Hankuk University of Foreign Studies (grant # 2011-1079001), and Louisiana Board of Regents (grant # LEQSF-2007-12-ENH-P-KSFI-PRS-03). We are also grateful to the Korea Advanced

Institute of Science and Technology's Ubiquitous Fashionable Computer project for its collaboration. We give special thanks to Kyuho Park, Geehyuk Lee, Hoi-Jun Yoo of KAIST, Xin Lin and Srivathsan Srinivasagopalan of Louisiana State University and to Mani Chandy of CalTech. 

References

1. Ahn, M. et al. SwanBoat: Pervasive social game to enhance treadmill running. In *Proceedings of ACM Multimedia Technical Demonstrations* (Beijing, Oct. 19–23). ACM Press, New York, 2009, 997–998.
2. Bächlin, M. et al. SwimMaster: A wearable assistant for swimmers. In *Proceedings of the International Conference on Ubiquitous Computing* (Orlando, FL, Sept. 30–Oct. 3). ACM Press, New York, 2009, 215–224.
3. Bao, L. and Intille, S.S. Activity recognition from user-annotated acceleration data. In *Proceedings of the International Conference on Pervasive Computing* (Vienna, Austria, Apr. 21–23). Springer, Berlin/Heidelberg, 2004, 1–17.
4. Bharatula, N.B. et al. Empirical study of design choices in multi-sensor context-recognition systems. In *Proceedings of the International Forum on Applied Wearable Computing* (Zürich, Mar. 17–18). VDE Verlag, Berlin, 2005, 79–93.
5. Fahy, P. and Clarke, S. CASS: A middleware for mobile context-aware applications. In *Proceedings of the Workshop on Context Awareness*, part of the *International Conference on Mobile Systems, Applications, and Services* (Boston, June 6, 2004).
6. Iyengar, S.S., Parameshwaran, N., Phoha, V.V., Balakrishnan, N., and Okoye, C.D. *Fundamentals of Sensor Network Programming: Applications and Technology*. Wiley-IEEE Press, Hoboken, NJ, Dec. 2010.
7. Kang, S. et al. Orchestrator: An active resource orchestration framework for mobile context monitoring in sensor-rich mobile environments. In *Proceedings of the IEEE International Conference on Pervasive Computing and Communications* (Mannheim, Germany, Mar. 29–Apr. 2). IEEE Computer Society, Washington, D.C., 2010, 135–144.
8. Kang, S. et al. SeeMon: Scalable and energy-efficient context-monitoring framework for sensor-rich mobile environments. In *Proceedings of the International Conference on Mobile Systems, Applications, and Services* (Breckenridge, CO, June 17–20). ACM Press, New York, 2008, 267–280.
9. KISS FFT, <http://kissfft.sourceforge.net/>
10. Lachenmann, A., Marrón, P.J., Minder, D., and Rothmel, K. Meeting lifetime goals with energy levels. In *Proceedings of the ACM Conference on Embedded Networked Sensor Systems* (Sydney, Nov. 6–9). ACM Press, New York, 2007, 131–144.
11. Lee, J. et al. BMQ-processor: A high-performance border-crossing event-detection framework for large-scale monitoring applications. *IEEE Transactions on Knowledge and Data Engineering* 21, 2 (Feb. 2009), 234–252.
12. Li, Q. et al. Accurate, fast fall detection using gyroscopes and accelerometer-derived posture information. In *Proceedings of the International Workshop on Wearable and Implantable Body Sensor Networks* (Berkeley, CA, June 3–5). IEEE Computer Society Press, Washington, D.C., 2009, 138–143.
13. Liu, X., Shenoy, P., and Corner, M.D. Chameleon: Application-level power management. *IEEE Transactions on Mobile Computing* 7, 8 (Aug. 2008), 995–1010.
14. Lorincz, K. et al. Mercury: A wearable sensor network platform for high-fidelity motion analysis. In *Proceedings of the ACM Conference on Embedded Networked Sensor Systems* (Berkeley, CA, Nov. 4–6). ACM Press, New York, 2009, 183–196.
15. Lorincz, K., Chen, B., Waterman, J., Allen, G.W., and Welsh, M. Resource-aware programming in the Pixie OS. In *Proceedings of the ACM Conference on Embedded Networked Sensor Systems* (Raleigh, NC, Nov. 5–7). ACM Press, New York, 2008, 211–224.
16. Noble, B.D. et al. Agile application-aware adaptation for mobility. In *Proceedings of the ACM Symposium on Operating Systems Principles* (Saint-Malo, France, Oct. 5–8). ACM Press, New York, 1997, 276–287.
17. Park, T., Yoo, C., Choe, S.P., Park, B., and Song, J. Transforming solitary exercises into social exergames. In *Proceedings of the ACM Conference on Computer Supported Cooperative Work* (Seattle, Feb. 11–15). ACM Press, New York, 2012.
18. Park, T., Lee, J., Hwang, I., Yoo, C., Nachman, L., and Song, J. E-Gesture: A collaborative architecture for energy-efficient gesture recognition with hand-worn sensor and mobile devices. In *Proceedings of the ACM Conference on Embedded Networked Sensor Systems* (Seattle, Nov. 1–4). ACM Press, New York, 2011, 260–273.
19. Raffa, G. et al. Don't slow me down: Bringing energy efficiency to continuous gesture recognition. In *Proceedings of the International Symposium on Wearable Computers* (Seoul, Oct. 10–13). IEEE Computer Society Press, Washington, D.C., 2010, 1–8.
20. Riva, O. Contory: A middleware for the provisioning of context information on smart phones. In *Proceedings of the International Middleware Conference* (Melbourne, Australia, Nov. 27–Dec. 1). Springer, 2006, 219–239.
21. Sadler, C.M. et al. Data-compression algorithms for energy-constrained devices in delay-tolerant networks. In *Proceedings of the ACM Conference on Embedded Networked Sensor Systems* (Boulder, CO, Oct. 31–Nov. 3). ACM Press, New York, 2006, 265–278.
22. Sorber, J. et al. Eon: A language and runtime system for perpetual systems. In *Proceedings of the ACM Conference on Embedded Networked Sensor Systems* (Sydney, Nov. 6–9). ACM Press, New York, 2007, 161–174.
23. Sorber, J. et al. Turducken: Hierarchical power management for mobile devices. In *Proceedings of the International Conference on Mobile Systems, Applications, and Services* (Seattle, June 6–8). ACM Press, New York, 2005, 261–274.
24. Vert, G., Iyengar, S.S., and Phoha, V. *Introduction to Contextual Processing: Theory and Applications*. CRC Press, Boca Raton, FL, Fall 2010.
25. Wang, Y. et al. A framework of energy-efficient mobile sensing for automatic user state recognition. In *Proceedings of the International Conference on Mobile Systems, Applications, and Services* (Krakow, Poland, June 22–25). ACM Press, New York, 2009, 179–192.
26. Zeng, H., Fan, X., Ellis, C.S., Lebeck, A., and Vahdat, A. ECOSystem: Managing energy as a first-class operating system resource. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems* (San Jose, CA, Oct. 5–9). ACM Press, New York, 2002, 123–132.

Youngki Lee (youngki@nclab.kaist.ac.kr) is a Ph.D. student in the Department of Computer Science of the Korea Advanced Institute of Science and Technology, Daejeon, Korea.

S.S. Iyengar (iyengar@csc.lsu.edu) is a professor in the Department of Computer Science of Louisiana State University, Baton Rouge, LA.

Chulhong Min (chulhong@nclab.kaist.ac.kr) is a Ph.D. student in the Department of Computer Science of the Korea Advanced Institute of Science and Technology, Daejeon, Korea.

Younghyun Ju (yju@nclab.kaist.ac.kr) is a Ph.D. student in the Department of Computer Science of the Korea Advanced Institute of Science and Technology, Daejeon, Korea.

Seungwoo Kang (swkang@nclab.kaist.ac.kr) is a post-doctoral researcher in the Department of Computer Science of the Korea Advanced Institute of Science and Technology, Daejeon, Korea.

Taiwoo Park (twpark@nclab.kaist.ac.kr) is a Ph.D. student in the Department of Computer Science of the Korea Advanced Institute of Science and Technology, Daejeon, Korea.

Jinwon Lee (jircle@nclab.kaist.ac.kr) is a researcher at Qualcomm Research, Santa Clara, CA.

Yunseok Rhee (rhees@hufs.ac.kr) is a professor in the Department of Electronics and Information Engineering of Hankuk University of Foreign Studies, Yongin, Korea.

Junehwa Song (junesong@kaist.ac.kr) is a professor in the Department of Computer Science of the Korea Advanced Institute of Science and Technology, Daejeon, Korea.