A mobile platform for context monitoring for PAN-scale dynamic mobile computing environments

Youngki Lee, Younghyun ju, Chulhong Min, Jihyun Yoo, Taewoo Park, Seungwoo Kang, Yoonseok Rhee, Junehwa Song

Dept. of Computer Science

KAIST
모바일 플랫폼의 진화

[íg signage]

피쳐폰

스마트폰
iPhone, Android

차세대 모바일 플랫폼

하드웨어 중심 통신 기능 집중

신호 사용자 인터페이스 감성적 만족 추구

한국 모바일 산업 위기

한국 모바일 산업 위기?
By the way, what has happened in Korea about embedded systems? why?
First Attempt: Supporting Proactive Applications
What are the requirements for true mobile App?

- Interface and interaction
  - Current status: touch, a bit of sound, a bit of motion
  - What would be good?

- Proactiveness and personalization
  - Smartphone understands me better than my wife!
  - what is required for proactive services?
  - Example:
Example

Context-aware alarm

- Identify a change in the user context
  - Essential for Personal Context-aware Applications

  - library, studying, afternoon
  - library, sleeping, afternoon

Cf. Context reporting
- Periodically report what the current user context is
- Report the user context whenever it satisfies a certain condition

Wake up!!
PAN-scale dynamic mobile computing

- A personal mobile device dynamically connected with many embedded, wearable, device-embedded sensors
- Serve a number of personal context-aware applications
  ▫ providing proactive, personalized, situation-aware services

Wearable or on-body sensors:
Accel, gyro, physiological sensors on watch, glove, shirt, belt, bracelet, ...
Continuous context monitoring

- Continuous monitoring of users’ context
  - A key building block for personal context-aware applications

- Often requires complex, multi-step, continuous processing
  - Over multiple devices
  - E.g. Running situation: sensing in three 3-axis accelerometers, FFT processing, recognition

Application logic
Context monitoring (e.g., sensing, feature extraction, recognition)
PAN-scale dynamic mobile computing platform

App logic
Context recognition
Feature extraction
Sensing
Challenges and Approach
Challenges

• A number of applications share highly scarce resources of the PAN-scale dynamic mobile computing environment
  ▫ Only a few applications can run due to resource constraint
  ▫ Potential capacity drop due to skewed resource utilization

• Significantly scarce resources
  ▫ E.g., MicaZ Motes: 8MHz CPU, 4KB RAM, ~50Kbps Bandwidth
    • A light FFT library, *kiss_fft*, requires 40KB RAM, 10 MHz CPU
  ▫ Limited battery power due to mobility

• Dynamic join/leave of heterogeneous sensors
  ▫ E.g., take off a watch sensor, enter a smart space with sharable environmental sensors

• Dynamic changes in resource demands and status
  ▫ Applications join and leave, registers and de-registers new and existing requests
  ▫ Sudden drops in BW availability due to mobility, obstacles, ...
Challenges

- Vital Sign Information
- Environmental Information
- Location/Activity Information

- Resource Management
- Energy problem
- Scalability problem

Tens/hundreds of heterogeneous sensors
Continuous data updates (Thousands of updates/sec)
Resource limitation (Battery, processor, memory)
Hundreds/thousands of long-running monitoring requests to detect context changes

- U-Secretary
- Location-based Services
- Health Monitoring
- U-Learning
- U-Trainer
- Diet diary
- U-Reminder
- Behavior correction
Applications themselves cannot solve the challenges

- Context monitoring is complexity thing to do!
  - Burdensome programming and debugging over sensor devices
  - Implementation of a range of processing modules for feature extraction and recognition
  - Repeated and time-consuming job for training (learning)

- It should handle (use and schedule) scarce resources over multiple devices
  - Awareness of the required amount of resources
  - Handling dynamic sensor availability and resource status
    - A variety of different resource conditions

- It should coordinate the resource use of other (multiple) applications
  - Should make applications communicate and negotiate with each other
  - However, it is almost impossible to make it
A new mobile platform for PAN-scale dynamic mobile computing environment

PCAs

U-Secretary

Diet diary

U-Trainer

API

Scalable and Energy-efficient Processing

Context Specification/Learning

Active Resource Orchestration

Privacy/Security Control

Collaborative Monitoring

Communication Management

Device Abstraction

MobiCon

Sensors

BVP/GSR

Accelerometers

GPS

Diet diary

Collaborative Monitoring

Context Specification/Learning

Privacy/Security Control

Attendance Tracking

U-Secretary

ACM
Current Status

- Active Resource Orchestration: ActraMon (PerCom 2010, TMC)
- Producer-oriented Execution of Sensing Flows: FastFlux (percom 2012)
- Orchestration of Multiple Sensing Applications on a Smartphone: SymPhoney (sensys 2012)
- Collaborative Context Monitoring: CoMon (MobiSys 2012)
Scalable and energy-efficient context monitoring
Computation & Energy Efficient Context Processing

- Bidirectional Processing Pipeline
- Transformation-based Approach
- Shared, Incremental Processing
- Producer-Oriented Context Processing
- Essential Sensor Set
General processing model for context/activity

- Requires **costly** operations for
  - Continuous **data updates** from sensors
  - Continuous **context processing**
    - Complex feature extraction and context recognition
  - Continuous **change detection**
    - Repeated examination of numerous monitoring requests
Our approach: bidirectional processing

- **Early detection** of context changes
  - Remove processing cost for continuous context recognition
- **Utilize the locality of feature data** in change detection
  - Reduce processing cost by evaluating queries in an incremental manner
- **Turn off unnecessary sensors** for monitoring results
  - Reduce energy consumption for wireless data transmission
Context Monitoring Query (CMQ)

- Simple and intuitive query language
  - Free developers from the complexity of continuous context monitoring
  - Support the semantics to catch the context change

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

- Example query
  - "Let me know if the user starts to run in a hot and humid weather"

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

Context elements
CMQ translation

- Avoid context recognition processing and enable feature data-level change detection

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

---

Feature data-level CMQ

---

**CONTEXT** (accel_1_y_energy > 52) AND (accel_3_x_dc < 500)
AND (accel_3_x_energy < 263) AND (temp > 86)
AND (humidity > 80)
ALARM F → T
DURATION 1 month

---

**Context Translation Map**

<table>
<thead>
<tr>
<th>Context-level semantic</th>
<th>Feature1</th>
<th>Feature2</th>
<th>Feature2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Value</td>
<td>ID</td>
<td>Low</td>
</tr>
<tr>
<td>activity</td>
<td>running</td>
<td>acceler_1_y_energy</td>
<td>52</td>
</tr>
<tr>
<td>temp</td>
<td>hot</td>
<td>temp</td>
<td>86</td>
</tr>
<tr>
<td>humidity</td>
<td>wet</td>
<td>humidity</td>
<td>80%</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

---

**Feature data-level CMQ**

<table>
<thead>
<tr>
<th>Feature1</th>
<th>Feature2</th>
<th>Feature2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Feature1</td>
<td>Feature2</td>
<td>Feature2</td>
</tr>
<tr>
<td>ID</td>
<td>Low</td>
<td>High</td>
</tr>
<tr>
<td>Low</td>
<td>High</td>
<td>ID</td>
</tr>
<tr>
<td>Low</td>
<td>High</td>
<td>ID</td>
</tr>
<tr>
<td>Low</td>
<td>High</td>
<td>ID</td>
</tr>
</tbody>
</table>

---

**CONTEX** (accel_1_y_energy > 52) AND (accel_3_x_energy < 263)
AND (temp > 86) AND (humidity > 80)
ALARM F → T
DURATION 1 month

---

20
Shared and Incremental Context Processing

- **Incremental** processing by exploiting the locality of context
  - Locality of context $\Rightarrow$ locality of feature data
- **Shared** processing on all registered CMQs

Continuous feature data show gradual changes

Query results often remain the same

Temperature data:
- 80, 81, 82, 82, 83

Humidity data:
- 83, 84, 86, 86, 85

CMQ Processor

- Q1: $\text{humidity} > 80$ AND $\text{temp} > 86$
- Q2: $\text{humidity} < 40$ AND $\text{accel}_3\_x\_energy < 263$
- Q3: $\text{temp} > 87$ AND $\text{accel}_3\_x\_dc \geq 500$

Q₁, Q₂, Q₃: False
Essential Sensor Set: Energy-efficient context monitoring

- Avoid unnecessary data transmission from wireless sensors
- Identify and deactivate unnecessary sensors
  - **Example query:** "Is the weather hot and humid?"
    - If it is already cool, no need to see humidity.

Feature data values

- Hygrometer
  - humidity: 85
- Thermometer
  - temp: 75
- Accelerometer3
  - energy: 105
  - dc: 280

Q1: (humidity > 80) AND (temp > 86)  
  - F AND T  
  - F

Q2: (humidity < 40) AND (accel_3_x_energy < 263)  
  - F AND T  
  - F

Q3: (temp > 87) AND (accel_3_x_dc < 500)  
  - F AND T  
  - F
Fast-Flux: Producer-oriented Execution of Sensing Flows
**Approach: Producer-oriented Execution**

**Consumer-oriented model:** Consumers collect, manage, and process their input data

- Frequent messaging and scheduling overhead
- Redundant management of the same data
- Repetitive memory allocation and de-allocation

**Producer-oriented model:** Producers manage their output data in an integrated fashion

- *Lazy data delivery:* reduces the number of data pass and operator scheduling operations
- *Integrated data management:* eliminates the redundancy in the separate management
- *In-place buffer write:* avoids repetitive memory allocation and de-allocation
DataBank

- Execution container for an operator that realizes the producer-oriented model
  - Serves as the basic unit in the execution of a dataflow graph
  - Takes charge of the management and delivery of the output data
  - Separates the pure processing logic from the logic for dataflow execution and data management

Flow table

<table>
<thead>
<tr>
<th>cons_id</th>
<th>wsize</th>
<th>sl_wsize</th>
<th>trig_idx</th>
<th>cons_ptr</th>
</tr>
</thead>
<tbody>
<tr>
<td>op_m</td>
<td>512</td>
<td>512</td>
<td>512</td>
<td></td>
</tr>
<tr>
<td>op_n</td>
<td>128</td>
<td>64</td>
<td>192</td>
<td></td>
</tr>
<tr>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
</tr>
</tbody>
</table>

Shared output buffer

DataBank of $op_i$
Execution Network

- A network of DataBanks for the execution of a dataflow graph

- DataBanks are connected via the pointers in internal flow tables
  - A DataBank passes data to next DataBanks through direct function call via the pointer
    - Low overhead in inter-operator communication

- DataBanks of sensing operators are connected to their consumers’ DataBanks via sensing data queue
  - Due to the asynchronous nature of sensing operators
Active Resource Orchestration
Active resource use orchestration

**active resource use orchestration** vs. **passive resource use mngmnt**

(e.g., in conventional resource management systems in mobile systems, sensor systems)

- High-level context monitoring request
  - E.g. Context == Running

- System-wide holistic view of applications and resources

- Flexible system-driven resource binding

- **Has limited view**, i.e., the resource requests

- **Low-level resource allocation request**
  - E.g. 5MHz CPU, 5KB RAM, 10kbps BW for a watch sensor

- **Static app.-driven resource binding**
Key features

- **Alternative resource usage plans**
  - Context $\rightarrow$ a variety of processing methods $\rightarrow$ diverse resource usages (device, task distribution)
- **Runtime decision** of the **best** use (by system-wide policy)
Operation example

Applications

- Plan A-1
- Plan A-2
- Plan B-1
- Plan B-2
- Plan C-1
- Plan C-2
- Plan C-3

Resources (RAM: Resource Availability Matrix)

<table>
<thead>
<tr>
<th>CPU(%)</th>
<th>Mem(KB)</th>
<th>BW(pkt/s)</th>
<th>Energy(J)</th>
</tr>
</thead>
<tbody>
<tr>
<td>63.22</td>
<td>56285</td>
<td>25.5</td>
<td>9256</td>
</tr>
<tr>
<td>97.83</td>
<td>1.064</td>
<td>25.5</td>
<td>8253</td>
</tr>
<tr>
<td>85.25</td>
<td>1.925</td>
<td>25.5</td>
<td>10258</td>
</tr>
</tbody>
</table>

RDM: Resource Demand Matrix

<table>
<thead>
<tr>
<th>CPU(%)</th>
<th>Mem(KB)</th>
<th>BW(pkt/s)</th>
<th>Energy(mJ/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mobile</td>
<td>0.845</td>
<td>57344</td>
<td>0.78 5.68</td>
</tr>
<tr>
<td>Accl.</td>
<td>2.17</td>
<td>1152</td>
<td>0.78 1.08</td>
</tr>
</tbody>
</table>

Plan B-1

- Accelerometer attached to sleeve

Plan B-2

- Accelerometer attached to belt

Plan C-2

- Accelerometer attached to belt

Sensor-side: Frequency domain feature extractor

Mobile-side: Decision Tree
SymPhoney:
Orchestration of Multiple Sensing Applications on a Smartphone
Concurrent Sensing Applications

- **CPU contention**
  - *Concurrent* sensing applications
  - *Continuous and heavy* CPU consumption (e.g., 5% ~ 20%)
  - *Resource-limited* smartphone
  - Users won’t want to use 100% of their CPU for BG jobs
CPU Contention Results in ...

- Sensing apps can’t process continuous sensing data on time !! → processing delay and data drop
  - Long service delay
  - Abrupt service vacancy
  - Wrong context inference, which are critical for timely, situation-aware services

- Performance degradation of foreground apps
  - e.g., frame drop in games, increase of web loading time
Current Mobile OS

- Allocate most of the CPU time to an activated FG app
  - e.g., more than 90% in Android
  - Multiple BG sensing apps should share the remaining small portion

![Graph showing CPU utilization and unprocessed sensing data over time.](image)
Current Mobile OS

• Hard to determine the proper amount of CPU time for each sensing app
  ▫ Sensing apps are dealt with just as black-box processes
  ▫ No application-level information such as
    • the CPU time required to process continuous sensing data
    • the required QoS level

→ Could result in unfair and inefficient resource use
SymPhoney

- A sensing flow execution engine for concurrent sensing apps
  - Coordinate the resource use of **contending applications**
    - Maximize their utilities under given resource conditions
  - Adapt to the **fluctuating resource availability** from FG apps
    - Minimize the performance degradation of the interactive apps
Flow-Aware Coordination

• Leverage a series of sensing and processing operations to produce a context result as a basic unit of resource allocation and scheduling
  ▫ Inspect dataflow graphs, and identify the unit of continuous sensing data to produce a context result for each graph: c-frame

• Best utilize system resources under contentious situations without unnecessary waste
  ▫ Ensure the data integrity inside assigned c-frames and the CPU time to process the frames
    • Do not compromise the accuracy of processing results and prevent prolonged delay
    • c.f., in-frame data drop in duty cycling or downsampling
  ▫ Eliminate unnecessary sensing and processing outside the c-frames.

• Facilitate the system to satisfy applications’ service quality in the coordination process
  ▫ Direct mapping between and resource allocation service provision
Observation

- Look into regular processing structure of sensing applications

**Flat** sensing data stream

**Well-structured** video stream
Frame Externalization

- Externalize semantic structures embedded in sensing data stream

Frame externalization provides useful hints for sensing flow coordination.
Multi-layered Frame Structure

feature frame (f-frame)  context frame (c-frame)

Layer 3
512 x 20 x 3 samples

Layer 2
512 x 20 samples

Layer 1
512 samples

sensing stream

[SoundSense, MobiSys’09]
How to Adjust Sensing Apps’ Resource Use?

Collecting and Processing a meaningful frame is critical to generate proper results!!
**c-frame-based Flow Coordination**

- **c-frame** as the basic unit of resource allocation

  A c-frame = A result
  → Easy to reflect application requirements

Contest results

Sensing stream

---

Ensure the deserved amount of resources for an allocated c-frame → Correct results
**c-frame-based Flow Coordination: Multiple Flows**

- **Processing time**: Available CPU utility functions of (e.g., 500ms/c-frame) (e.g., 20%) flow 0 and flow 1.

- **Contestion**
  - Context results
  - Sensing stream
  - Flow 0
  - Flow 1

- Monitoring interval: 5, 20

- **Utility functions**
  - Utility 0
  - Utility 1

- **α, c-frame interval of flow 0**

- **β, c-frame interval of flow 1**
Architecture

- **urgency-based flow scheduling**
- **Efficient flow execution: execution container**
- **c-frame-based flow coordination**
- Resource reservation for BG sensing apps

SymPhoney

- Resource Monitor
- Flow Scheduler
- Flow Execution Planner
- Flow Executor

Applications

Execution results

Efficient flow execution:
- execution container

- **CPU quota**

Mobile OS

- CPU Scheduler
- Sensor
  - Accel
  - Gyro
  - Mic.

Resource reservation for BG sensing apps
Implementation

- Implemented on Android in Java

- Run as a Android service and provide XML-based service interface

- Provide 50+ types of built-in operators commonly used in mobile sensing apps

<table>
<thead>
<tr>
<th>Operator types</th>
<th>SymPhoney built-in operators</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sensing operators</td>
<td>Sound, Accel., Gyro., GPS, ...</td>
</tr>
<tr>
<td>Feature extractors</td>
<td>FFT, MFCC, RMS, Correlation, Energy, Average, Entropy, ...</td>
</tr>
<tr>
<td>Classifiers</td>
<td>GMM, HMM, Decision tree, ...</td>
</tr>
</tbody>
</table>

- CPU quota allocation is implemented on Android CFS scheduler
  - No kernel modification
SymPhoney effectively coordinate sensing apps’ resource use in changing resource situations
SymPhoney provides applications with high utilities considering their resource demand and requirements.
Conclusion (SymPhoney)

- Newly address the CPU contention problem between continuous sensing apps and with other mobile apps

- SymPhoney – a mobile sensing **flow execution engine for concurrent sensing applications**
  - Propose a flow-aware coordination approach
    - Develop frame-based coordination and execution methods
  - Maintain sensing apps’ utilities at a reasonable level even under high contention

- Frame externalization can provide valuable hints to address various system challenges for mobile sensing applications
CoMon
We Travel Together. Why Everyone Sense?
Expected Power Savings

I Travel Alone

≈ 440 mW

≈ 315 mW

I Meet Young

≈ 80 mW

I Meet Brian

≈ 110 mW

≈ 315 mW
Enough Opportunities in Daily Lives!

- 8.5 hours of collocation with acquaintances (ATUS).
  - American Time Use Survey (http://www.bls.gov/tus/):
    - Survey over 10,000 participants about what they do, for how long, with whom per day
- 47% meetings continues >1 hour. 65% do >30 Min

### Time Use Per Day

- 8.5 hours with acquaintances
- 5.8 hours Alone/with strangers
- 9.8 hours Sleeping/private activities
CoMon Overview

Context-level negotiation: benefit-aware negotiation

Monitoring a context in turn
Air Quality (I: 5 Min, You: 5 Min)
Conclusion

• PAN-scale dynamic mobile computing environments
  ▫ new personal computing environments enabling
    • continuous context monitoring and
    • personal context-aware applications

• Platform support for proactive mobile applications
  ▫ First step toward UX-oriented mobile apps
  ▫ A new mobile platform for the emerging sensor-rich mobile computing environments
    • Energy efficiency
    • Scalability
    • Active resource use orchestration
  ▫ support a number of concurrent applications with highly scarce and dynamic resources
  ▫ active coordination between multiple applications and multiple underlying resources
Thanks a lot!  Questions

Juehwa Song

Dept. of CS
KAIST

junesong@cs.kaist.ac.kr
http://nclab.kaist.ac.kr