APE: An Annotation Language and Middleware for Energy-Efficient Mobile Application Development

Nima Nikzad, Octav Chipara†, and William G. Griswold

nnikzad@cs.ucsd.edu

Dept. of Computer Science and Engineering, UC San Diego
Dept. of Computer Science, University of Iowa†
Growth of Continuously-Running Mobile Applications

- Runs in background, out of user's sight
- Periodic workloads
- Often delay-tolerant
  - e.g., health monitoring, sports and news feeds, social networking

**How can we run many CRM applications while minimizing impact on battery life?**
Low- and Application-level Optimizations are Both Needed

• **Low-level Optimizations** can only do so much
  • DVFS, tickless kernels, radio scheduling, batched writes to flash, etc...

• **Application-level Optimizations** are necessary
  • Workload shaping, filtering, piggy-backing radio transmission, etc...
  • **Take advantage of application specific behavior**
Thread uploadThread = new Thread(new Runnable() {
    while (true) {
        try {
            Thread.sleep(1200000);
            // Sleep 20 minutes
        } catch (InterruptedException e) {
            attemptUpload();
        }
    }
});
uploadThread.start();

Case Study: Naïve Upload in CitiSense

Alerts, Maps

CO, NO₂, O₃
Small Transmissions $\rightarrow$ Big Impact

Power

1 W

10 mW

$t$

6 sec.

12 sec.
Piggybacking is Effective for Amortizing Cost of Communication
Thread uploadThread = new Thread(new Runnable() {
    while(true) {
        try {
            Thread.sleep(1200000);
        } catch(InterruptedException e) {
            attemptUpload();
        }
    }
});
uploadThread.start();

TelephonyManager teleManager = (TelephonyManager) context.getSystemService(Context.TELEPHONY_SERVICE);
TransListener transListener = new TransListener();
teleManager.listen(transListener, PhoneStateListener.LISTEN_DATA_ACTIVITY);

private class TransListener extends PhoneStateListener {
    public void onDataActivity(int act) {
        if(act == TelephonyManager.DATA_ACTIVITY_IN ||
            act == TelephonyManager.DATA_ACTIVITY_OUT ||
            act == TelephonyManager.DATA_ACTIVITY_INOUT) {
            uploadThread.interrupt();
        }
    }
}
Building Energy-Efficient Apps is Not Simple!

• Code for power-management tends to be complex
  • Highly application specific
    • Trade-off: Energy vs Timeliness and/or Accuracy
  • Thread management, event and state change handling

• Optimizations should be postponed until application requirements are set
  • Use cases and ‘acceptable’ delay may change over time
  • But... Retrofitting → Time + Bugs
What Do Energy-Management Policies Typically Look like?

• We examined the latest research, best-practice guides, documentation, and open-source projects
  • Android Best Practice Guide, AT&T Research, [Wang09], [Nath12], [Nikzad12], [Musolesi10], Cyanogen Project

• Common thread: *Timing and Device State!*

• “Wait until _______ before executing _______”
Thread uploadThread = new Thread(new Runnable() {
    while(true) {
        Intent batt = context.registerReceiver(null, new IntentFilter(Intent.ACTION_BATTERY_CHANGED));
        int lvl = batt.getIntExtra(BatteryManager.EXTRA_LEVEL, -1);
        int scl = batt.getIntExtra(BatteryManager.EXTRA_SCALE, -1);
        float batteryPct = lvl / (float) scl;
        try {
            if(batteryPct > 70) {
                Thread.sleep(120000);
            } else {
                Thread.sleep(360000);
            }
        } catch(InterruptedException e) {
            attemptUpload();
        }
    }
}).start();

private class TransListener extends PhoneStateListener {
    public void onDataActivity(int act) {
        if(act == TelephonyManager.DATA_ACTIVITY_IN || act == TelephonyManager.DATA_ACTIVITY_OUT || act == TelephonyManager.DATA_ACTIVITY_INOUT) {
            uploadThread.interrupt();
        }
    }
}

Thread uploadThread = new Thread(new Runnable() {
    while(true) {
        @APE_If("Battery.Level > 70%")
        @APE_WaitUntil("Network.Active", MaxDelay=1200)
        @APE_Else()
        @APE_WaitUntil("Network.Active", MaxDelay=3600)
        attemptUpload();
    }
}).start();
Rest of this Talk

while(true) {
    @APE_WaitUntil("Net.Active", MaxDelay=1200)
    uploadSensorData();
}

$\text{true:Net.Activity}$

$\text{c}_{\text{G}} \geq 20 \text{ min}: \text{true}$
The APE Policy Model

Effectively Modeling Policies Using Timed Automata

- Clarify semantics
- Drive design of language and runtime
Many Policies Can Be Expressed Using Simple Timed Automata

Wait up to 20 minutes for cellular network activity
More Complicated Policies Can Fall Back to Earlier States

Wait for movement using the accelerometer, then wait up to thirty seconds for driving to be detected

\[ P = \{(\text{Accel.Move}, \infty), (\text{GPS.Drive}, 30\text{ sec})\}, \infty\)
Many Policies Can Be Expressed Using Very Simple Timed Automata

Wait up to 20 minutes for cellular network activity

\[ P = (\{(\text{Net}.\text{Active}, \infty)\}, 20 \text{ min}) \]
The APE Annotation Language

Representing Power-management Policies as Annotations

• Expressing simple timed automata using text
APE_WaitUntil: Wait For Desired Device State

Wait up to 20 minutes for cellular network activity

\[ P = \{(\text{Net.Active}, \infty)\}, \text{20 min}\]
APE_WaitUntil:
Wait For Desired Device State

Wait for movement using the accelerometer, then wait up to thirty seconds for driving to be detected

\[ P = \{(\text{Accel.Move, } \infty), (\text{GPS.Drive, } 30 \text{ sec})\}, \infty) \]

```java
while(true) {
    @APE_WaitUntil("\{(\text{accMove()}\}, \text{inf}), (\text{gpsDrive()}\}, 30)\",
    MaxDelay=\text{inf})
    downloadDriveData();
}
```
Other APE Constructs

• if-else constructs

```java
while(true) {
    @APE_If("Battery.Level > 70%")
    @APE_WaitUntil("Net.Active", MaxDelay=1800)
    @APE_Else()
    @APE_WaitUntil("WiFi.Connected", MaxDelay=3600)
    uploadSensorData();
}
```

• Creating new terms and saving policies

```java
@APE_DefineTerm("MyTerm", "Battery.Charging AND (WiFi.Connected OR Cell.4G)"
@APE_DefinePolicy("MyPolicy", "(Display.Off,inf),(MyTerm,10)"
```
The APE Middleware Service

Runtime Hardware Monitoring on Behalf of Client Apps

- Minimizing overhead associated with monitoring
Annotations are Processed at Compile-time into Runtime Requests

APE Annotated Source

APE Annotation Preprocessor

Source Including Generated APE Requests
Clients Start Up the APE Service and Send Requests at Runtime

Client Process A
Register “Net.Active” as 1

Client Process B
Bind me to the APE Service

APE Service
A.1: Net.Active
B.1: …
APE Returns Control to the Calling Thread Once the Request is Satisfied

Client Process A

Return WaitUntil(1)

APE

Tell me about changes in cellular state

Net.Active

B.1:

Data outgoing on cellular radio

Client Process B

Android System
Boolean Expressions Are Transformed into Expression Trees

WiFi.Connected OR Network.Active AND (Cell.3G OR Cell.4G)
The APE Service Performs Device State Monitoring on Behalf of Clients

APE Service

Network State Monitor

Android System
Evaluation

A Case Study of Optimizing a CRM Application Using APE

• How complicated are policies in practice?
• What are the potential benefits and overhead?
Case Study: CitiSense

```java
Thread uploadThread = new Thread(new Runnable() {
    while(true) {
        Intent batt = context.registerReceiver(null,
                                        new IntentFilter(Intent.ACTION_BATTERY_CHANGED));
        int lvl = batt.getIntExtra(BatteryManager.EXTRA_LEVEL, -1);
        int scl = batt.getIntExtra(BatteryManager.EXTRA_SCALE, -1);
        float batteryPct = lvl / (float) scl;
        try {
            if(batteryPct > 70) {
                Thread.sleep(1200000);
            } else {
                Thread.sleep(3600000);
            }
        } catch(InterruptedException e) {
            attemptUpload();
        }
    }
    uploadThread.start();
});
```

Four APE annotations versus 45 lines of Java
(thread suspension/waking, new class for handling callbacks, registering for callbacks)
Background Apps Can Repeatedly Wake Resources

Device Power Consumption

Six application instances with no piggybacking
The Simplest of Policies Can Have a Big Impact on Power-Consumption

Device Power Consumption

Six application instances with @APE_WaitUntil("Net.Active", MaxDelay=120)
The Simplest of Policies Can Have a Big Impact on Power-Consumption

Increase in power consumption after first instance

Increase of 13.49 mw (1.6%)

Increase of 551.06 mw (63.7%)
Overhead of Executing Each WaitUntil Call vs Hand-coded

Overhead of IPC measured to be 1.71 ms
Limitations and Future Work

• Assumes developer is already aware of opportunities for energy savings
  • Under Development: Automatically identifying costly operations and suggesting safe, effective policies

• User study to properly evaluate APE’s ease-of-use and sufficient expressibility of language
Conclusion

• Use of **timing is widely applicable** for power-management in mobile applications

• A simple **annotation language** and **runtime** can make this technique **conveniently** accessible to programmers

• **Timed automata** provide a **semantic model** that informs the design of language and runtime

• APE significantly **eases the implementation** of policies in CitiSense and achieves **dramatic power-savings**
Acknowledgement

• This work was supported by the National Science Foundation and the Roy J. Carver Charitable Trust

William Griswold
UC San Diego

Octav Chipara
Univ. of Iowa
Conclusion

• Use of **timing is widely applicable** for power-management in mobile applications

• A simple **annotation language** and **runtime** can make this technique **conveniently** accessible to programmers

• **Timed automata** provide a **semantic model** that informs the design of language and runtime

• APE significantly **eases the implementation** of policies in CitiSense and achieves **dramatic power-savings**
DON’T USE - Animation back up
DON’T USE - Animation back up
Challenge: Managing Energy-Consumption in CRM Apps

- **Timing** plays a significant role in energy consumption.
- **Waking the CPU**
- **Transmitting data**
- **Powering the display**

Transmit as much as possible at this time!
Case Study: CitiSense

```java
Thread uploadThread = new Thread(new Runnable() {
    while(true) {
        try {
            Intent batt = context.registerReceiver(null,
                    new IntentFilter(Intent.ACTION_BATTERY_CHANGED));
            int lvl = batt.getIntExtra(BatteryManager.EXTRA_LEVEL, -1);
            int scl = batt.getIntExtra(BatteryManager.EXTRA_SCALE, -1);
            float batteryPct = lvl / (float) scl;
            if(batteryPct > 70) { Thread.sleep(120000); }
            else { Thread.sleep(360000); }
        } catch(InterruptedException e) {
            attemptUpload();
        }
    }
    uploadThread.start();
    TelephonyManager teleManager = (TelephonyManager) context.getSystemService(Context.TELEPHONY_SERVICE);
    TransListener transListener = new TransListener();
    teleManager.listen(transListener, PhoneStateListener.LISTEN_DATA_ACTIVITY);
    private class TransListener extends PhoneStateListener {
        public void onDataActivity(int act) {
            if(act == TelephonyManager.DATA_ACTIVITY_IN ||
                    act == TelephonyManager.DATA_ACTIVITY_OUT ||
                    act == TelephonyManager.DATA_ACTIVITY_INOUT) {
                uploadThread.interrupt();
            }
        }
    }
});
```
Case Study: CitiSense

Thread uploadThread = new Thread(new Runnable() {
  while(true) {
    Intent batt = context.registerReceiver(null, new IntentFilter(Intent.ACTION_BATTERY_CHANGED));
    int lvl = batt.getIntExtra(BatteryManager.EXTRA_LEVEL, -1);
    int scl = batt.getIntExtra(BatteryManager.EXTRA_SCALE, -1);
    float batteryPct = lvl / (float) scl;
    try {
      if(batteryPct > 70) {
        Thread.sleep(120000);
      } else {
        Thread.sleep(360000);
      }
    } catch(InterruptedException e) {
      attemptUpload();
    }
  }
  uploadThread.start();
};
APE: Annotated Programming for Energy-Efficiency

• A small and declarative annotation language with a lightweight middleware runtime for Android
  • Demarcate power-hungry code using annotations
  • Execution deferred until device enters state that minimizes cost of operation
  • Abstract away details of event/state monitoring

• Abstract model based on timed automata
  • Guides design of language and middleware
Outline

• Introduction
• Approaches to Energy-Management

• APE: Annotated Programming for Energy-efficiency
  • Policy Model
  • Annotation Language
  • Runtime Service

• Evaluation
  • Case Study: CitiSense
  • Overhead and Power Savings

• Conclusion
The APE Policy Model

• Restricted form of timed automata:

\[ TA=(\Sigma, S, s0, SF, C, E) \]

Typical Automata Can Be Expressed Textually

\[ P=\{(\sigma_1,t_1), (\sigma_2,t_2)...(\sigma_n,t_n), t_{MaxDelay}\} \]

- \( \sigma_i \) is a Boolean expression consisting of events and states
- \( t_i \) is a constraint on how long to wait for \( \sigma_i \) to be true
- \( t_{MaxDelay} \) is an upper bound on the delay of \( O \)

- Linear, concise nature is easy to express in writing
  - Drove design of annotation language
A Policy is a Sequence of Boolean Expressions and Timing Constraints

• Defer the execution of an operation $O$ until a finite sequence of states holds:

$$P=\{(\sigma_1,t_1), (\sigma_2,t_2)\ldots(\sigma_n,t_n), t_{MaxDelay}\}$$

• $\sigma_i$ is a Boolean expression consisting of events and states
• $t_i$ is a constraint on how long to wait for $\sigma_i$ to be true
• $t_{MaxDelay}$ is an upper bound on the delay of $O$

• If $\sigma_i$ is satisfied before $t_i$, begin monitoring for $\sigma_{i+1}$
• Else, begin monitoring for $\sigma_{i-1}$ again
• Execute $O$ once $\sigma_n$ is satisfied or $t_{MaxDelay}$ has passed
Simplified Model: Limits Power for Conciseness

• Wait for driving to be detected after 30 seconds of continuous accelerometer movement

Not expressible in our model!
APE_If, APE_Elself, APE_Else: Conditionals for Policy Selection

Allow for dynamic selection of policies at runtime

```java
while(true) {
    @APE_If(“Battery.Level > 70%”)
        @APE_WaitUntil(“Net.Active”, MaxDelay=1800)
    @APE_Else()
        @APE_WaitUntil(“WiFi.Connected”, MaxDelay=3600)
    uploadSensorData();
}
```
APE_DefineTerm, APE_DefinePolicy: Enabling Reuse

@APE_DefineTerm("MyTerm", "Battery.Charging AND (WiFi.Connected OR Cell.4G)"")

@APE_DefinePolicy("MyPolicy", "(Display.Off, inf), (MyTerm, 10)"")

@APE_WaitUntil("MyPolicy(inf, 60), ({dataReady()}, 30)", MaxDelay=3600)
Annotations are Processed at Compile-time into Runtime Requests

```
while(true) {
  @APE_WaitUntil("WiFi.Connected", MaxDelay=1800)
  uploadSensorData();
}

void onCreate(Bundle savedInstanceState) {
  super(savedInstanceState);
  APEService.register_expr(1, "WiFi.Connected");
  ...
}
```
The APE Service Performs Device State Monitoring on Behalf of Clients
Conclusion

• A small and declarative annotation language with a lightweight middleware runtime for Android
  • Demarcate power-hungry code using annotations
  • Execution deferred until device enters state that minimizes cost of operation
  • Abstract away details of event/state monitoring

• Manage trade-off between timing and energy

• In a case study, a policy implemented with only six annotations drastically improved efficiency
  • 86% reduction in lines of power-management code
Two Categories of Approaches to Energy-Management

• **Low-level Optimizations**: DVFS, tickless kernels, radio scheduling, batched writes to flash, etc...
  - Implemented in kernel, driver, or firmware
  - Responsibility of the device and OS vendors

• **System-level Optimizations**: workload shaping, filtering, piggy-backing radio transmission, etc...
  - Implemented at application level
  - **Responsibility of the application developers**