Transactuations:
Where Transactions Meet the Physical World

USENIX ATC ‘19

Aritra Sengupta
(Samsung Research)

Tanakorn Leesatapornwongsa
(Microsoft Research)

Masoud Saeida Ardekani
(Uber Technologies)

Cesar A. Stuardo
(University of Chicago)

* Work done at Samsung Research America
• IoT solutions are becoming ubiquitous
• Hundreds of applications for smart homes
  • Automation
  • Security
  • Safety
• Early stage and immature
- IoT solutions are becoming ubiquitous
- Hundreds of applications for smart homes
  - Automation
  - Security
  - Safety
- Early stage and immature

Failure implication goes beyond inconvenience!
When Smart Home Is Not Smart

Inconsistent Behavior

Upset Customer A
“... More importantly, we were robbed when we were out on vacation. I had it set to armed away. The logs show the motion of the robbers, but it never sounded the alarm ... I no longer trust it to do what it is supposed to do when it is supposed to do ... “

[1] SmartThings Community: https://community.smartthings.com
function handleMotion(evt) {
  //isIntruder reads other sensors
  //and determines intrusion
  if (isIntruder(evt) && !state.alarmActive) {
    alarm.strobe();
    state.alarmActive = true;
  }
}

&
!state.alarmActive

state.alarmActive = true
(for avoiding redundant actions)
function handleMotion(evt) {
    //isIntruder reads other sensors
    //and determines intrusion
    if (isIntruder(evt) && !state.alarmActive) {
        alarm.strobe();
        state.alarmActive = true;
    }
}

(state.alarmActive = true (for avoiding redundant actions))
function handleMotion(evt) {
    // isIntruder reads other sensors
    // and determines intrusion
    if (isIntruder(evt) && !state.alarmActive) {
        alarm.strobe();
        state.alarmActive = true;
    }
}

Read sensor and app state

Actuating a device

&

!state.alarmActive

state.alarmActive = true
(for avoiding redundant actions)
Intrusion Detection Application

```javascript
function handleMotion(evt) {
    // isIntruder reads other sensors and determines intrusion
    if (isIntruder(evt) && !state.alarmActive) {
        alarm.strobe();
        state.alarmActive = true;
    }
}
```

Read sensor and app state
Actuating a device
Writing app state

&

!state.alarmActive

state.alarmActive = true
(for avoiding redundant actions)
Failure Example

What if actuation command is lost or a glitch in the alarm?
Failure Example

& !state.alarmActive

state.alarmActive = true
(for avoiding redundant actions)

What if actuation command is lost or a glitch in the alarm?

Physical state   ! =  Application state
Failure Example

The alarm is based on wireless transmissions ... can be subject to RF interference, ... cause the alarm to not operate as intended ...
Failure makes application and device states **inconsistent**
Failure makes application and device states **inconsistent**

Inherent concurrency in applications also leads to inconsistencies
How often can inconsistencies happen?

- Identified 3 classes of dependencies in application logic
- Dependencies capture semantic relationship between app and device
- These 3 dependencies are vulnerable to failures
How often can inconsistencies happen?

• Identified 3 classes of dependencies in application logic
• Dependencies capture semantic relationship between app and device
• These 3 dependencies are vulnerable to failures

By statically analyzing applications for dependencies, we can identify potential inconsistencies in smart applications
Dependency

1. Sensing $\rightarrow$ actuating
   
   \[
   c = \text{co2.value()}
   \]
   
   \[
   \text{if } (c > \text{threshold})\
   \quad \text{fans.on()}
   \]

2. Sensing $\rightarrow$ app state update
   
   \[
   t = \text{thermo.value()}
   \]
   
   \[
   \text{if } (t > 90)\
   \quad \text{setMode(“HOT“)}
   \]

3. Actuating $\rightarrow$ app state update
   
   \[
   \text{alarm.strobe()}
   \]

   active = “TRUE”
1. Sensing $\rightarrow$ actuating

\[
c = \text{co2.value}()
\]

\[
\text{if} \ (c > \text{threshold})\{
\quad \text{fans.on()}
\}
\]

2. Sensing $\rightarrow$ app state update

\[
t = \text{thermo.value}()
\]

\[
\text{if} \ (t > 90)\{
\quad \text{setMode("HOT")}
\}
\]

3. Actuating $\rightarrow$ app state update

\[
\text{alarm.strobe()}
\]

\[
\text{active} = \text{"TRUE"}
\]
Dependency

1. Sensing $\rightarrow$ actuating
   
   ```
   c = co2.value()
   if (c > threshold){
       fans.on()
   }
   ```

2. Sensing $\rightarrow$ app state update
   
   ```
   t = thermo.value()
   if (t > 90){
       setMode("HOT")
   }
   ```

3. Actuating $\rightarrow$ app state update
   
   ```
   alarm.strobe()
   active = "TRUE"
   ```
Can Transactions address the problem?

NO
Can Transactions address the problem? 

NO

- IoT devices cannot be locked
  - Users can observe intermediate value
Can Transactions address the problem?  

**NO**

- IoT devices cannot be locked  
  - Users can observe intermediate value
- Rolling back IoT devices have consequences 
  - A user observes a door locks then rolls back to unlocked 
  - Not a good user experience!
Can Transactions address the problem?

**NO**

- IoT devices cannot be locked
  - Users can observe intermediate value
- Rolling back IoT devices have consequences
  - A user observes a door locks then rolls back to unlocked
  - Not a good user experience!
- Some actuations cannot be rolled back
  - Undoing a water dispenser
Transactuation

- High level abstraction and programming model
  - Allows a developer to read/write from/to devices
  - Failure-aware association of application and device states
Transactuation

• High level abstraction and programming model
  • Allows a developer to read/write from/to devices
  • Failure-aware association of application and device states

• Atomic durability for application states
  • Actuations never roll back
Transactuation

• High level abstraction and programming model
  • Allows a developer to read/write from/to devices
  • Failure-aware association of application and device states

• Atomic durability for application states
  • Actuations never roll back

• (Internal) atomic visibility among transactuations
  • External atomic visibility cannot be guaranteed for end users!
  • Disallows several concurrency related bugs
Transactuation

• High level abstraction and programming model
  • Allows a developer to read/write from/to devices
  • Failure-aware association of application and device states

• Atomic durability for application states
  • Actuations never roll back

• (Internal) atomic visibility among transactuations
  • External atomic visibility cannot be guaranteed for end users!
  • Disallows several concurrency related bugs

• Guarantees two invariants
Transactuation

- High level abstraction and programming model
  - Allows a developer to read/write from/to devices
  - Failure-aware association of application and device states
- Atomic durability for application states
  - Actuations never roll back
- (Internal) atomic visibility among transactuations
  - External atomic visibility cannot be guaranteed for end users!
  - Disallows several concurrency related bugs
- Guarantees two invariants

**Sensing Invariant**
Governs executing a transactuation

**Actuating Invariant**
Governs committing a transactuation
Sensing Invariant
Transactuation executes only when staleness of its sensor reads is bounded, as per specified sensing policy.

Sensing policy
How much staleness is acceptable
How many failed sensors is acceptable

Example of sensing policy
at least one co2 sensor can be read within last 5 mins
Actuating Invariant
When a transactuation commits its app states, sufficient number of actuations have succeeded as per specified actuation policy.

Actuation policy
How many failed actuation is acceptable.

Example of actuation policy
At least one alarm should successfully turn on.
Simplified Example

```javascript
(sensors) => {
  let active = read('active');
  if (sensors['co2'] > threshold && !read('active')) {
    actuate('fans', 'on');
    write('active', true);
  }
  ...
}
```
Simplified Example

```javascript
let tx = new Transactuation();

tx.perform(
  (sensors) => {
    let active = read('active');
    if (sensors['co2'] > threshold && !read('active')) {
      actuate('fans', 'on');
      write('active', true);
    }
    ...
  }
);
```
let tx = new Transactuation();

Sensing policy

tx.perform(['co2'], 5m, 'sense_all'
    (sensors) => {
        let active = read('active');
        if (sensors['co2'] > threshold && !read('active')) {
            actuate('fans', 'on');
            write('active', true);
        }
        ...
    });

Application logic
Simplified Example

```javascript
let tx = new Transactuation();

tx.perform(['co2'], 5m, 'sense_all', 'act_all',
(sensors) => {
  let active = read('active');
  if (sensors['co2'] > threshold && !read('active')) {
    actuate('fans', 'on');
    write('active', true);
  }
  ...
});
```
Execution Model

1. Start if Sensing policy is satisfied

T₁

Execute app logic
defer actuations
Execution Model

1. Start if Sensing policy is satisfied
2. Speculative commit

- Find a serializable order
- Avoid rollback

Execute app logic
defer actuations

T₁
Execution Model

1. Start if Sensing policy is satisfied
2. Speculative commit
3. Final commit according to Actuating policy

Find a serializable order
Avoid rollback
Execution Model

Actuate devices

T_1

Trigger

T_2

Overlapping computation and actuation
Execution Model

Actuate devices

T1

T2

T3

Trigger

Wait

Overlapping computation and actuation
Implementation: Relacs

• Runtime called Relacs is built on Azure technology
  • Azure Functions (serverless functions)
  • Cosmos DB (Relacs store)

• Integrated to Samsung SmartThings IoT platform
Evaluation

• Programmability
• Correctness
• Runtime overhead without failures
• Runtime overhead with failures
<table>
<thead>
<tr>
<th>Application</th>
<th>Original App</th>
<th>Original App + Consistency</th>
<th>Transactuation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rise and Shine (Cn1)</td>
<td>72</td>
<td>195</td>
<td>68</td>
</tr>
<tr>
<td>Whole House Fan (Cn2)</td>
<td>29</td>
<td>176</td>
<td>26</td>
</tr>
<tr>
<td>Thermostat Auto Off (Cn3)</td>
<td>70</td>
<td>198</td>
<td>68</td>
</tr>
<tr>
<td>Auto Humidity Vent (Ee1)</td>
<td>49</td>
<td>170</td>
<td>100</td>
</tr>
<tr>
<td>Lights Off With No Motion (Ee2)</td>
<td>56</td>
<td>161</td>
<td>67</td>
</tr>
<tr>
<td>Cameras On When Away (Sc1)</td>
<td>31</td>
<td>149</td>
<td>88</td>
</tr>
<tr>
<td>Nobody Home (Sc2)</td>
<td>65</td>
<td>175</td>
<td>62</td>
</tr>
<tr>
<td>Smart Security (Sc3)</td>
<td>144</td>
<td>323</td>
<td>144</td>
</tr>
<tr>
<td>Co2 Vent (Sf1)</td>
<td>29</td>
<td>152</td>
<td>26</td>
</tr>
<tr>
<td>Lock It When I Leave (Sf2)</td>
<td>51</td>
<td>180</td>
<td>54</td>
</tr>
<tr>
<td>Application</td>
<td>Original App</td>
<td>Original App + Consistency</td>
<td>Transactuation</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>--------------</td>
<td>----------------------------</td>
<td>----------------</td>
</tr>
<tr>
<td>Rise and Shine (Cn1)</td>
<td>72</td>
<td>195</td>
<td>68</td>
</tr>
<tr>
<td>Whole House Fan (Cn2)</td>
<td>29</td>
<td>176</td>
<td>26</td>
</tr>
<tr>
<td>Thermostat Auto Off (Cn3)</td>
<td>70</td>
<td>198</td>
<td>68</td>
</tr>
<tr>
<td>Auto Humidity Vent (Ee1)</td>
<td>49</td>
<td>170</td>
<td>100</td>
</tr>
<tr>
<td>Lights Off With No Motion (Ee2)</td>
<td>56</td>
<td>161</td>
<td>67</td>
</tr>
<tr>
<td>Cameras On When Away (Sc1)</td>
<td>31</td>
<td>149</td>
<td>88</td>
</tr>
<tr>
<td>Nobody Home (Sc2)</td>
<td>65</td>
<td>175</td>
<td>62</td>
</tr>
<tr>
<td>Smart Security (Sc3)</td>
<td>144</td>
<td>323</td>
<td>144</td>
</tr>
<tr>
<td>Co2 Vent (Sf1)</td>
<td>29</td>
<td>152</td>
<td>26</td>
</tr>
<tr>
<td>Lock It When I Leave (Sf2)</td>
<td>51</td>
<td>180</td>
<td>54</td>
</tr>
</tbody>
</table>
Runtime Overhead without Failures

Execution time (s)

50% overhead with transactuations
Runtime Overhead without Failures

50% overhead with transactuations

Serverless function triggering overhead
Conclusion

• Established a critical reliability issue due to inconsistencies

• Transactuation allows a developer to program in a failure-aware way

• Demonstrated transactuation’s programmability, performance, and effectiveness
Additional Slides
Relacs

Relacs Serverless System Functions

Read Sensors

Relacs Store
Relacs

Transform to
Serverless Function

Read Sensors

Relacs Serverless
System Functions

Relacs Store
Relacs

1. Read Sensors

Transform to Serverless Function
Relacs

Transform to Serverless Function

1. Read Sensors
2. Speculative Commit

Relacs Store

Relacs Serverless System Functions

Read Sensors
Relacs

Transform to Serverless Function

1. Read Sensors

2. Speculative Commit

3. Final Commit (actuate devices)

Relacs Serverless System Functions

Read Sensors

Relacs Store