SPOT: A Smartphone-Based Platform to Tackle Heterogeneity in Smart-Home IoT Systems

Mohammad-Mahdi Moazzami†, Guoliang Xing
Michigan State University
East Lansing, MI
{moazzami, glxing}@cse.msu.edu

Daisuke Mashima†
ADSC
Singapore
daisuke.m@adsc.com.sg

Wei-Peng Chen
Fujitsu Labs of America
Sunnyvale, CA
wchen@us.fujitsu.com

Ulrich Herberg†
Independent Researcher
Santa Clara, CA
ulrich@herberg.name

† Work is done while the author is with Fujitsu Labs of America.
Heterogeneity in smart appliances
- No clear winner yet for IoT standards
- So many vendor-specific APIs
- Multiple, non-connected apps

Needs for unified control platform
- E.g. Home energy management system or Home security systems.
Heterogeneity in IoT Devices (1)

- Communication interface
  - Different communication architecture
    - Direct access, hub-based, cloud-based, …
  - Different transport/communication protocols
    - HTTP/HTTPS over WiFi, Zigbee, Bluetooth, …
  - Different messaging schema
    - JSON, XML, …
  - Different authentication mechanisms
    - UserID/Password, OAuth2, no authentication, …
Different parameter naming

- Nest Thermostat: target_temperature_f
- Radio Thermostat: t_heat or t_cool

Different message structure (e.g., nested vs. flat)

![Figure 2. Heterogeneity in programming abstraction](image)

![Figure 3. Heterogeneity in the command messages](image)
Study in Real IoT Devices

<table>
<thead>
<tr>
<th>Appliance Name</th>
<th>Vendor</th>
<th>Type</th>
<th>Protocol</th>
<th>Architecture</th>
<th>Auth.</th>
<th>Msg.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Philips HUE light[1]</td>
<td>Philips</td>
<td>light</td>
<td>RESTful</td>
<td>HUB</td>
<td>usr/pass</td>
<td>JSON</td>
</tr>
<tr>
<td>Nest thermostat[8]</td>
<td>Nest</td>
<td>thermostat</td>
<td>RESTful</td>
<td>Cloud</td>
<td>OAuth2.0</td>
<td>JSON</td>
</tr>
<tr>
<td>WeMo Switch[2]</td>
<td>WeMo</td>
<td>plug</td>
<td>SOAP</td>
<td>Local IP Add.</td>
<td>-</td>
<td>XML</td>
</tr>
<tr>
<td>Venstar thermostat[12]</td>
<td>Venstar</td>
<td>thermostat</td>
<td>RESTful</td>
<td>Local IP Add.</td>
<td>-</td>
<td>JSON</td>
</tr>
<tr>
<td>UFO Smart Plug[18]</td>
<td>UFO</td>
<td>plug</td>
<td>RESTful</td>
<td>Local IP Add.</td>
<td>-</td>
<td>JSON</td>
</tr>
<tr>
<td>IntesisHome AC Controller[19]</td>
<td>IntesisHome</td>
<td>AC controller</td>
<td>RESTful</td>
<td>Cloud</td>
<td>OAuth2.0</td>
<td>JSON</td>
</tr>
</tbody>
</table>
Design Goals

- Smartphone-based implementation
  - Take advantage of existing smartphone and tablet devices to minimize introductory cost
  - Only need commercial-off-the-shelf (COTS) smartphones w/o “rooting”

- Separation of device drivers from app’s logic
  - Properties of device (e.g., variables and configuration settings) are defined in a “driver”
  - Support dynamic driver loading to enable applications on top of the framework to seamlessly operate devices

- User/community-driven device support
  - By publishing the schema of XML device driver, anyone, including users themselves or open-source community can design and distribute drivers.
  - Do not need to wait for device vendors to publish drivers
Solution: SPOT

- Offers open, customizable, extensible platform to address heterogeneity among multi-vendor smart appliances
- Add your favorite appliances/devices to your smart-home app
Variables

- "type" indicates the class of the device
- "parent" is used to define nested structure
- "canonicalName" is the "common" name of a variable defined in SPOT

```xml
<!-- variables data unit-->
<xs:element name="variablesList">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="variable" maxOccurs="unbounded">
        <xs:complexType>
          <xs:sequence>
            <xs:element type="xs:string" name="name" min="value" max="value"/>
            <xs:element name="canonicalName"/>
            <xs:element name="parent"/>
            <xs:element type="xs:string" name="type"/>
            <xs:element type="xs:boolean" name="writePermission"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```
Read/Write action

Defines how request messages should be generated and sent

- HTTP method
- URL
- Message body format
- Etc.

```xml
<!-- read action unit-->
<xs:element name="read">
  <xs:complexType>
    <xs:sequence>
      <xs:element type="xs:string" name="httpMethod"/>
      <xs:element type="xs:anyURI" name="baseUri"/>
      <xs:element type="xs:string" name="uriExtentionPattern"/>
      <xs:element type="xs:string" name="responsePattern"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<!-- write action unit-->
<xs:element name="write">
  <xs:complexType>
    <xs:sequence>
      <xs:element type="xs:string" name="httpMethod"/>
      <xs:element type="xs:anyURI" name="baseUri"/>
      <xs:element type="xs:string" name="bodyType"/>
      <xs:element type="xs:string" name="bodyPattern"/>
      <xs:element name="httpHeaderFields">
        <xs:complexType>
          <xs:sequence>
            <xs:element type="xs:string" name="headerField"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```
<!-- variables data unit -->
<x:s:element use:intuit="false" type="xs:complexType">
  <x:s:element name="variablesList" maxOccurs="unbounded" minOccurs="0" />
  <x:s:element type="xs:complexType">
    <x:s:element name="variable" maxOccurs="unbounded" minOccurs="0" />
      <x:s:element type="xs:complexType">
        <x:s:element name="name" maxOccurs="unbounded" minOccurs="0" />
          <x:s:element type="xs:string" name="name" maxOccurs="unbounded" minOccurs="0" />
            <x:s:element type="xs:string" name="canonicalName" maxOccurs="unbounded" minOccurs="0" />
              <x:s:element type="xs:string" name="parent" maxOccurs="unbounded" minOccurs="0" />
                <x:s:element type="xs:boolean" name="writePermission" maxOccurs="unbounded" minOccurs="0" />
                  <x:s:element type="xs:string" name="showOnUi" maxOccurs="unbounded" minOccurs="0" />
                    <x:s:element type="xs:string" name="uiType" maxOccurs="unbounded" minOccurs="0" />
                      <x:s:element type="xs:string" name="uiHelpertext" maxOccurs="unbounded" minOccurs="0" />
                        <x:s:element type="xs:string" name="uiCaption" maxOccurs="unbounded" minOccurs="0" />
                          </x:s:element>
                          </x:s:complexType>
                      </x:s:element>
                    </x:s:complexType>
                  </x:s:element>
                </x:s:complexType>
              </x:s:element>
            </x:s:complexType>
          </x:s:element>
        </x:s:complexType>
      </x:s:element>
    </x:s:element>
  </x:s:complexType>
</x:s:element>

---

**RadioThermostat**

- **Temperature**
  - Current Temperature: 73.5

- **Mode**
  - Operating Mode: HEAT

- **Temporary Target Heat**
  - Sets target and mode: 62.0

- **Temporary Target Cool**
  - Sets target and mode: 0.0

- **Fan Operating Mode**
  - Current fan Mode: AUTO

**IntesisThermostat**

- **Temperature**
  - Current Temperature: 24.0

- **On/Off**
  - The On/Off State

- **Mode**
  - HVAC Mode: COOL

- **SetPoint**
  - Current AC SetPoint (In Celcius): 22.0

**VenstarThermostat**

- **Temperature**
  - Current Temperature: 82.0

- **Fan**
  - Current fan state: 0

- **Mode**
  - Set operating Mode: COOL

- **Heat temperature**
  - Set current heat to temperature: 72.0

- **Cool temperature**
  - Set current cool to temperature: 80.0

- **Fan Mode**
  - set fan mode: AUTO
How SPOT App Works – Getting Values

- The app can get values from the device using HTTP
  - Sends the HTTP request to the URI specified in the driver
  - Then “pattern matches” the payload of received response with the driver
    - The variables (indicated by “$”) stored in the app will be replaced by the received value

```xml
<get>
  <http_type>GET</http_type>
  <http_body>
    
    "sensors": [
    
    {
      "name": "$thermostat_name",
      "temp": "$current_temp"
    },

    
    ]
  </http_body>
</get>
```

GET: https://server.cloudprovider.com/query/sensors
How SPOT App Works – Setting Values

- The app sets values on the device via HTTP
- Sends the HTTP request (POST/PUT) to the URI specified in the driver
- Maps the values of the variables to the accepted format in the driver
  - Any variables (indicated by “$” will be replaced by the variable that is stored in the app).

```xml
<set>
  <http_uri>https://server.cloudprovider.com/control?mode=$mode&amp;fan=$fan&amp;
  heattemp=heating_setpoint&amp;cooltemp=cooling_setpoint</http_uri>
  <http_version>1.1</http_version>
  <http_type>POST</http_type>
  <http_header>Content-Type: application/x-www-form-urlencoded</http_header>
  <http_body></http_body>
</set>
```

POST: https://server.cloudprovider.com/control?mode=1&fan=1&
heattemp=70&cooltemp=75
Demo: Device Control

Smart Thermostat
## Evaluation

- Architectural comparison with other smart-home platforms

<table>
<thead>
<tr>
<th>Framework</th>
<th>Multi-Vendor</th>
<th>Multi-type</th>
<th>Support of devices with local WiFi API</th>
<th>Support of devices with cloud-based API</th>
<th>Third-party Extensibility</th>
<th>Unified Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>HomeOS [17]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>-</td>
<td>✓ (C#)</td>
<td>-</td>
</tr>
<tr>
<td>openHAB [5]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓ (JAVA)</td>
<td>-</td>
</tr>
<tr>
<td>IFTTT [3]</td>
<td>✓</td>
<td>✓</td>
<td>-</td>
<td>✓</td>
<td>✓</td>
<td>-</td>
</tr>
<tr>
<td>Nest thermostat [4]</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>✓</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>AllJoyn [1]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>HomeKit[2]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>SmartThings[12]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓ (XML)</td>
</tr>
<tr>
<td>SPOT</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
Evaluation

- CPU and memory footprint on smartphone
  - The CPU usage of SPOT is 0.0%: no computation in the platform
  - The memory: 49 MB during silence
  - SPOT app binary size: as small as 9.69MB

- Latency of appliance control command invocation
  - Communicate directly on local WiFi, e.g., Philips HUE light and Venstar
    - 150 ms
  - Communicate via vendor’s cloud e.g., Nest
    - 395 ms
Evaluation

- Latency of dynamically loading drivers with different number of variables
Evaluation

- The effect of polling appliances state
Conclusions and Future Works

- SPOT is a IoT device driver framework that enables:
  - Centralized control over heterogeneous IoT devices to enable truly connected smart home
  - User/community-driven device support enhancement

- Prototype app is implemented on Android platform
  - Can practically work on commodity smart phones

- We plan to apply SPOT to other IoT domains, such as vehicular systems and healthcare systems, and even to systems across multiple domains
shaping tomorrow with you
SPOT smart appliance driver generator

- **drivers**
  - **driver**
    - **driverName:**
    - **deviceType:**
    - **driverVendor:**
    - **localAccess:**
    - **cloudAccess:**
      - **common**
        - **httpHeaderFields**
          - **headerField**
            - **name:**
            - **value:**
        - **baseUrl:**
      - **read**
        - **httpMethod:**
        - **baseUrl:**
        - **uriExtensionPattern:**
        - **responsePattern:**
      - **write**
        - **httpMethod:**
        - **baseUrl:**
        - **bodyType:**
        - **bodyPattern:**
          - **headerField**
      - **variablesList**
        - **Add New Variable**
          - **variable 1**
            - **maxValue:**
            - **minValue:**
            - **description:**

[Generate XML]