Virginia Tech Operating System (OS) built on VOLTTRON™ for Energy Management in Buildings

July 24, 2014

Manisa Pipattanasomporn
(mpipatta@vt.edu)
Virginia Tech

Murat Kuzlu
(mkuzlu@vt.edu)
Virginia Tech
Outline

System architecture – VT OS

Software architecture – VT OS

Operating system and framework layer implemented using VOLTTRON™

Device Discovery Agent

Live Demonstration
System Architecture – VT OS

Plug Load Controllers

Power Meters

BACnet / Modbus

ZigBee mesh

HVAC Controllers

BACnet

/ZigBee

Wi-Fi

ZigBee

VT OS

Wi-Fi

Lighting Load Controllers

VT OS

Wi-Fi

VT OS
Software Architecture – VT OS

1. User interface layer
   - Web UI
   - Mobile UI

2. Application & data management layer
   - User management
     - Demand response
     - Price-based management
     - Behavior pattern analysis
     - Load management
     - Planning and scheduling
     - Alarm/notifications

3. Operating system and framework
   - Volttron – Linux-Ubuntu
     - Resource Manager
     - Authentication & Authorization
     - Communication Services
     - Directory Services

4. Connectivity layer
   - Communication Technologies (ZigBee, WiFi, Ethernet)
   - Data exchange protocols (BACnet, Modbus, SEP, ZigBee API, web)

Database
- User profile
- Preference settings
- Device information

sMAP
- Time series data, such as:
  - Device status
  - V, I, P, kW, etc.
  - Room temperature
  - Real-time pricing
  - Demand response signals

Cloud sources (e.g., OpenADR)
VT OS: Operating System and Framework Layer

VT OS is built upon VOLTRON™
Device Discovery Agent

**Step 1:** Detect the presence of a device in the building
   “I am here”

**Step 2:** Query the device to find out its address and model information
   “This is my model number”

**Step 3:** Look up the device’s API
   “This is what I can do”

**Step 4:** Initiate an agent associated with the discovered device
Step 1: Use SSDP to send a discovery message to the multicast address 239.255.255.250:1900

Code to send the multicast command:

```python
if option==1:
    message = "\n    M-SEARCH * HTTP/1.1",
    'HOST: (0):1',
    'MAN: "ssdp:discover",',
    'ST: (st)', 'MX: 3', '',''
else:
    message = "\n
socket.setdefaulttimeout(timeout)
for _ in range(retries):
    sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM, socket.IPPROTO_UDP)
    sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    sock.setsockopt(socket.IPPROTO_IP, socket.IP_MULTICAST_TTL, 2)
    if option==1:
        sock.sendto(message.format(*group, st=service, group), group)
    else:
        sock.sendto(message, group)
```

Service message for RadioThermostat:

"TYPE: WM-DISCOVER\nVERSION: 1.0\nservices: com.marvell.wm.system*\n"

A response from RadioThermostat:

"TYPE: WM-NOTIFY\nVERSION: 1.0\nSERVICE: com.marvell.wm.system:1.0\nLOCATION: http://38.68.232.113/sys/"

Response includes IP

Code for parsing responses from devices:

```python
while True:
    try:
        response = SSDPResponseLocation(sock.recv(1024))
        responses[response.location] = response
        except socket.timeout:
            break
```

```python
class SSDPResponseLocation(object):
    def __init__(self, response):
        tokens = response.split('\n')
    for token in tokens:
        if re.search('LOCATION: ', token):
            location = token.replace('LOCATION: ', '')
        break
    def __repr__(self):
        return self.location
```
Step 2: Query device to find its MAC address and model information

Code to send GET request; and retrieve UUID from JSON response:

```python
# Send GET request to device and retrieve UUID from JSON response
deviceuuidUrl = urllib2.urlopen(ipaddress)
deviceuuid=self.parseJSONResponse(deviceuuidUrl.read()).decode("utf-8"), "uuid")
```

For RadioThermostat, **MAC address** can be found at: [http://ip_address/sys/](http://ip_address/sys/)

A sample JSON format response:

```json
{"uuid":"88308a22316", "api_version":113, "fw_version":"1.04.84", "wlan_fw_version":"V10.105576"}
```

Response includes MAC address

Parsing UUID response:

```python
def parseJSONResponse(self, data, key):
    theJSON = json.loads(data)
    return theJSON[key]
```

For RadioThermostat, **device model information** can be found at: [http://ip_address/tstat/model/](http://ip_address/tstat/model/)

Code to send GET request; and retrieve model number from JSON response:

```python
deviceModelUrl = urllib2.urlopen(ipaddress.replace("/sys","/tstat/model"))
deviceModel = self.parseJSONResponse(deviceModelUrl.read()).decode("utf-8"), "model")
```

A sample JSON format response:

```json
{"model":"CT50 V1.94"}
```

Response includes model name

The agent checks the MAC ID against the database, MAC ID does not match = newly discovery device. If the MAC ID is already in the database, the device was previously discovered and an agent exists for that device.
Step 3: Look up device API

API Interface allows agents to communicate with a group of devices based on their APIs.

API interface for USNAP

API interface for ZigBee

Connectivity Layer

Agents can get device readings and send control commands using simple function calls:
- getDeviceStatus
- setDeviceStatus

Communications using device APIs
The thermostat agent is assigned its behaviors:

- Monitor
- Control
- Update UI

The thermostat agent is set up to:

- Publish its information in IEB
- Subscribe to relevant data sets

Step 4: Initiate a thermostat agent

1. Discover a CT-50 thermostat
2. Initiate a thermostat agent that communicates w/ CT-50
Once the discovery agent gets device information, device discovery status is displayed in the UI.
Live Demo for Device Discovery Process

Live Demo
Thank You

Manisa Pipattanasomporn
mpipatta@vt.edu

Murat Kuzlu
mkuzlu@vt.edu

Virginia Tech – Advanced Research Institute
Live Demo for Device Discovery Process

Step 1: Join our network
Network: ARI_Demo
PW: ARI_Demo

Step 2: Go to our UI
IP: 192.168.1.101:8000
Username: admin
PW: admin

See devices appearing on the screen as they are detected.