Programming abstractions for the Internet of Things
From Macroprogramming to Virtual Resources

Andrea Azzarà – a.azzara@sssup.it

November the 19th, 2015
Introduction

The Internet of Things
The Internet of Things
The next revolution of the Internet

• Common objects will be able to communicate and cooperate with each other to realize innovative applications
  - A huge number of systems embedded in the environment
  - Wireless Sensor Networks and RFID

• The IoT is expected to have a significant impact on the world economy and on the life of people

• A fundamental driver for the development of Smart cities
  - Building automation
  - Intelligent Transportation Systems
  - Health-care
Wireless Sensor Networks
A building block of the IoT

• Sensors-Actuators networks
  – Small, low cost, resource-constrained embedded devices (motes)
  – Self-organizing topology and multi-hop communication
  – Focus on data collection

• Main challenges
  – Energy saving
    • In-network processing
    • Duty cycling
    • Energy harvesting
  – Resource Constrains
    • Low memory and CPU power
    • Low Bandwidth, short ranges, small payloads
The Unique Features of IoT Networks

• Scalability
  – Unprecedented pervasiveness
  – Need for network architectures and protocols

• Shared infrastructures/Multiple applications

• Part of a larger computing infrastructure (the Cloud)

• Communication through standard protocols
IoT Enabling Technologies

- Global Internet Addressing
  - IPv6/6LoWPAN

- Energy-efficient routing
  - RPL (Low-power lossy networks)

- RESTful interaction and Resource abstraction
  - CoAP (Constrained Application Protocol)
    - HTTP-like (lightweight, UDP-based)
    - URI support
      - coap://[aaaa::202:2:2:1]/led
    - Built-in resource discovery
      - coap://[aaaa::202:2:2:1]/.well-known/core
    - Asynchronous notification model
      - Observe

- Wireless Sensors Networks become inter-operable systems connected to the Internet
Motivation and Contribution
Motivation
Programming the IoT is difficult

• The complexity of developing applications still limits the growth of the IoT
• IoT networks are inherently difficult to program
  – coordination and cooperation among the nodes
• IoT programming requires a knowledge in different domains
  – electronics, networking and low-level system architectures
Motivation
Programming the IoT is difficult

• Need for novel abstraction and tools designed to simplify the development of applications for the IoT
  – Hide distribution and communication details
  – Deal with the limited resources (computational power, battery, network bandwidth, etc.)

• Existing solutions lack of generality
  – they are often targeted to isolated systems (based on proprietary protocols) → difficult to interoperate
  – designed for a specific application → difficult to generalize
Motivation
Cloud-centric IoT

- *Cloud-centric IoT* is the emerging architecture
- Resource-limited sensors and actuators with RESTful interfaces at one end
- Full-fledged Cloud-hosted applications at the opposite end
- The application logic resides entirely in the Cloud
Motivation
Cloud-centric IoT

• **Pros**
  – Applications benefit from virtually-unlimited computing resources
  – Wire range of development tools and integrated services

• **Cons**
  – Application logic is unlikely interested in the individual readings
  – Increases the network traffic within the resource-constrained network
  – Higher energy consumption
  – Increased Latencies
  – No in-network processing
Contribution
Novel tools and abstractions

– PyoT
  • Macroprogramming system facilitating the development of applications for the IoT
  • Supports dynamic configuration of the processing of IoT nodes
  • Suitable for real-world implementation (Smart Plant Monitoring and safety enhancement systems, Multimedia Sensor networks for Intelligent Trasportation)

– Virtual Resources
  • An alternative to Cloud-based model
  • Push slices of the application logic down to the IoT network
The Virtual Resource Abstraction for the Internet of Things
Application Scenario: building automation

- Smart control electrical appliances such as lights and HVAC
  - Reduce energy expenditures
  - Retain the user comfort
- Sensed data are not used in raw form: aggregation based on
  - Location (public spaces vs. private offices)
  - Type (not critical appliances safely dimmed or switched off)
Virtual Resources

• An architectural solution for IoT applications that contrasts the separation between physical devices and Cloud-hosted applications

• Developers can relocate slices of the application logic to any intermediate IoT device using Virtual Sensors and Virtual Actuators
Virtual Resources

• With Virtual Sensors developers can
  – acquire data from a set of physical sensors
  – process the data according to a programmer-provided processing function
• In the building automation scenario
  – A “total-energy-public-areas” sensor provides the sum of the readings of all power meters in public areas
• Virtual Actuators
  – A single entry point for the Cloud-hosted application to control the lights in public areas at once
• RESTful interface akin or even identical to a physical device
Virtual Resources: Design

• Virtual Resources are hosted in a **Virtual Resource Directory** and are defined by
  
  – **Templates**
    • Abstractly specify the services offered by a virtual resource
    • Foster re-use
  
  – **Instances**
    • derived from templates
    • every instance is a sub-resource of the template it is derived from
  
  – **Configuration resources**

<table>
<thead>
<tr>
<th>Virtual Resource Directory</th>
</tr>
</thead>
<tbody>
<tr>
<td>VS Template</td>
</tr>
<tr>
<td>VS Instance Configuration</td>
</tr>
<tr>
<td>VA Template</td>
</tr>
<tr>
<td>VA Instance Configuration</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>
Virtual Resources: Design

• Templates specify three aspects
  – The resources of interest
  – The structure of the corresponding instances
    • The interface offered by a Virtual Resource and its configuration sub-resources
  – The distributed behavior of the virtual resource
    • E.g., Push/Pull model

<table>
<thead>
<tr>
<th>Virtual Resource Directory</th>
</tr>
</thead>
<tbody>
<tr>
<td>VS Template</td>
</tr>
<tr>
<td>VS Instance</td>
</tr>
<tr>
<td>Configuration</td>
</tr>
<tr>
<td>VA Template</td>
</tr>
<tr>
<td>VA Instance</td>
</tr>
<tr>
<td>Configuration</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>
From Templates to Instances

• The Templates are published in the Directory
  – Allow the creation of Instances
  – When a new Instance is created the configuration sub-resources are automatically created

• Instances can be created
  – Using the RESTful interface of the Directory (CoAP)
  – Programmatically (PyoT)

• A virtual resource may use another virtual resource as input/output
Virtual Resources: Programming

• Publish Templates on the Directory

```python
metersPublic = resource_set(Type='power_meter', location='public_area')
VsPeriodic(Input=metersPublic, name='per_power_meter_public')
lightsPublic = resource_set(Type='light_control', location='public_area')
VaSimple(Output=lightsPublic, name='lights_control_public')
```

• Create and configure Instances

```python
metersTempl = Resource.get(name='per_power_meter_public')
vs = metersTempl.POST('vs')
vs.proc.PUT(sumCode)
vs.period.PUT('60')
lightsTempl = Resource.get(name='lights_control_public')
va = lightsTempl.POST('va')
va.proc.PUT(translationCode)
```
Virtual Resources: Programming

• Processing Code

```python
Actuators = get_actuator_list()
setpoint = get_setpoint()
if type(setpoint) == Power:
    setpoint = power2lux(setpoint)
if len(Actuators) > 0:
    newSetpoint = setpoint / len(Actuators)
    set_actuator(newSetpoint)
```

• Cloud-based Application

```python
def control_app():
    while True:
        meter_values = vs.GET()
        new_output = control_fun(meter_values)
        va.PUT(new_output)
```
Implementation
Run-time support

• Virtual Resource Directory
  – An advanced CoAP Resource Directory

• PyoT
  – Extended to support Virtual Resource Templates and Instances
  – Takes care of the smart deployment of the Virtual Resources in the IoT network

• T-Res (Daniele Alessandrelli 2013)
  – Provides the “processing” feature of the Virtual Resources
  – Extended to support virtual actuators, and new interaction modes with I/O resources

• The Prototype
  – WisMote devices (MSP430, 16 kB of RAM, 256 kB of ROM)
  – Contiki OS
  – CoAP, 6LoWPAN, RPL, IEEE 802.15.4
Smart Instance Placement

• The performance of the distributed processing is influenced by RPL and depends on what device a Virtual Resource is allocated on

• RPL creates a DAG
  – Message travel along the links in the DAG
  – optimized when most of the traffic goes through the root
  – If a common ancestor exists
    • Path shortcut (A ↔ B)
  – Otherwise
    • Messages travel up to the root (C ↔ D)
Smart Instance Placement

- Virtual Resources generate peer-to-peer traffic among arbitrary nodes
- We developed a simple heuristic to drive the placement of Virtual Resources
  1. VRs are placed on the first common ancestor (walking the graph from the I/O resources to the root)
  2. If multiple ancestors exist, we take the one farthest from the root
  3. If common ancestors do not exist, the VR is placed on an available node closest to the root
Evaluation Setup

• Two Control Applications
  – Cloud-Centric vs. Virtual Resources
  – Virtual Sensor
    • A sliding-window-sensor template (computes an average over a sliding window of sensor values asynchronously collected from the physical resources)
  – Virtual Actuator
    • Command translation

• Qualitative comparison and Simulation (Cooja/MSPsim)
Evaluation

• 21 nodes
  – A border router (RPL root)
  – Physical sensors and actuators
  – Intermediate nodes (message routers)
  – A control node modelling the Cloud-hosted application

• RDC vs. no RDC

• Control loop period = 5 seconds

• Controlled Topologies vs. Randomly-generated

• We measured
  – The control loop latency
  – The inter-actuator latency
  – The energy consumption
Evaluation: Controlled Topologies

- Input/output of virtual resources physically co-located
- Varying
  - the number of physical sensors and actuators
  - the hop-distance between the RPL root and the virtual resources

Baseline

Virtual Resources
Evaluation: Controlled Topologies

- No radio duty-cycling
- Control loop and inter-actuator latencies decrease using Virtual Resources
- The system scales more gracefully.

**Control Loop Latency**

**Inter Actuator Delay**

![Graphs showing latency and delay for different configurations.](Image)
Evaluation: Controlled Topologies

- RDC: ContikiMAC
- The gains in control loop latency amplify in favor of Virtual Resources
- Energy savings for message routers
Evaluation: Random Topologies

• More than 70 randomly-generated topologies
• The placement of virtual and physical resources is initially random
• Then we apply the heuristic to place the Virtual Resources
Evaluation: Random Topologies

- Virtual Resources perform better even when considering the fully random placement.
- Further improvements after applying the heuristic.

Control Loop Latency

Inter Actuator Delay
Evaluation: Random Topologies

- Lower traffic load reduces energy spent by routers
- Physical sensors and actuators may become message forwarders → Reduced consumption with Virtual Resources
Virtual Resources: Final Remarks

• Network resources are better utilized
  – the amount of data coming from sensors is reduced
  – Reducing network traffic prolongs the system lifetime
  – Reduced congestion in IoT network → reduced latencies

• The Cloud-hosted application becomes simpler
  – Driving a single actuator for all lights in public areas is easier than a varying number of individual actuators
  – the Cloud-hosted application does not need to change when the inputs (outputs) set changes

• Better separation of concerns
  – The company manufacturing the IoT devices may also provide a library of virtual resources
  – Domain experts implement the high-level application logic at the Cloud
Conclusions

• New tools and abstractions designed to simplify the development of IoT applications
• PyoT is a macroprogramming framework facilitating the development of distributed applications for IoT networks
• Virtual Resources allow for a better utilization of network resources, prolonging the system lifetime, and promoting the separation of concerns in the development of cloud-hosted applications

Thanks for your attention! Questions?
List of Publications


List of Publications


