How do we program the Internet of Things at scale?

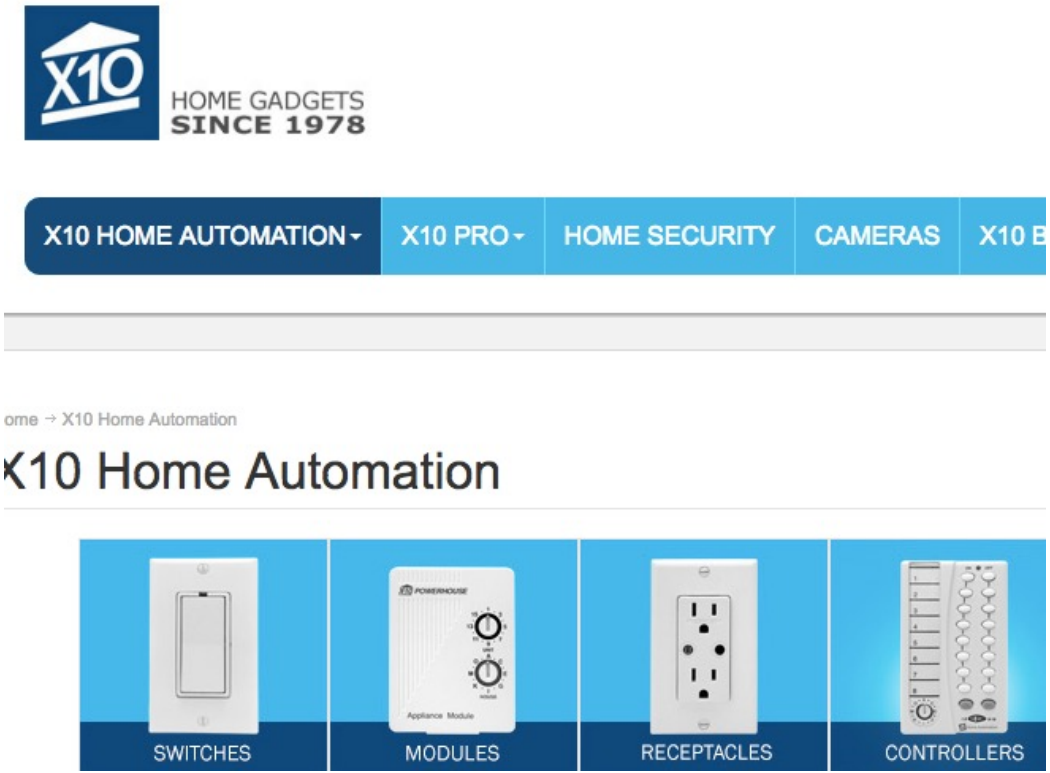
Jan Janak, Luoyao Hao and Henning Schulzrinne

(Columbia University)

September 2021 – RIoT

This material is based upon work supported by the National Science Foundation under Grant No. (CNS-1932418). Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

IoT is not exactly new (1978)



IoT – an idea older than the web (1985)

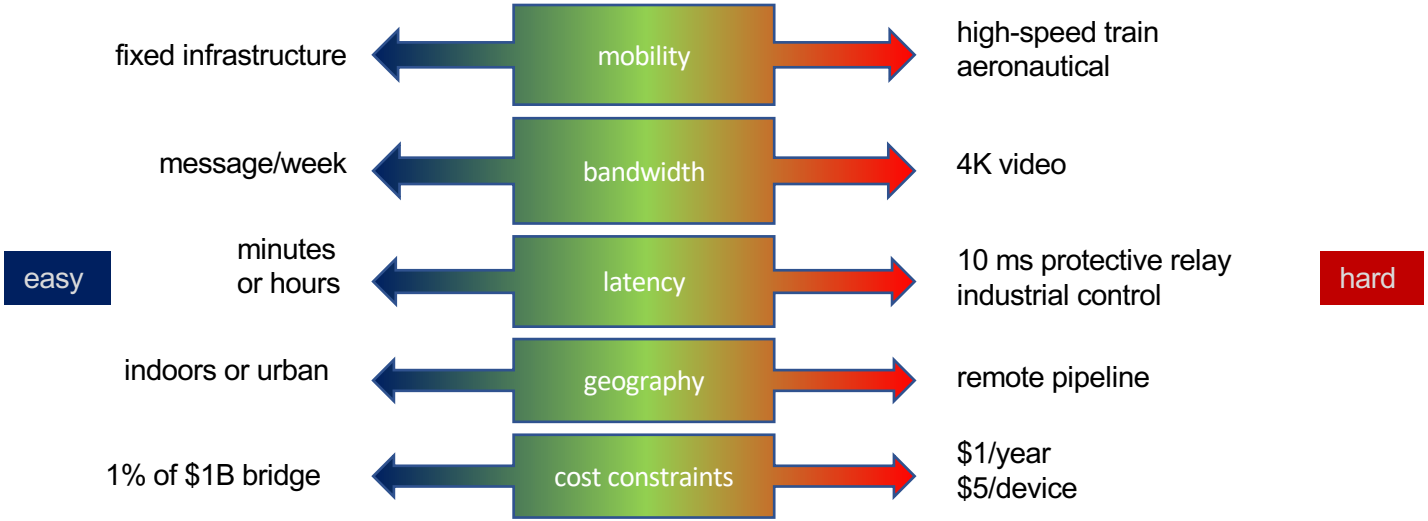
Peter Lewis (panel discussion 1985)

By connecting devices such as traffic signal control boxes, underground gas station tanks and home refrigerators to supervisory control systems, modems, auto-dialers and cellular phones, we can transmit status of these devices to cell sites, then pipe that data through the Internet and address it to people near and far that need that information. I predict that not only humans, but machines and other things will interactively communicate via the Internet. ***The Internet of Things, or IoT, is the integration of people, processes and technology with connectable devices and sensors to enable remote monitoring, status, manipulation and evaluation of trends of such devices****. When all these technologies and voluminous amounts of Things are interfaced together -- namely, devices/machines, supervisory controllers, cellular and the Internet, there is nothing we cannot connect to and communicate with. What I am calling the Internet of Things will be far reaching.*

From Chetan Sharma Consulting 2016

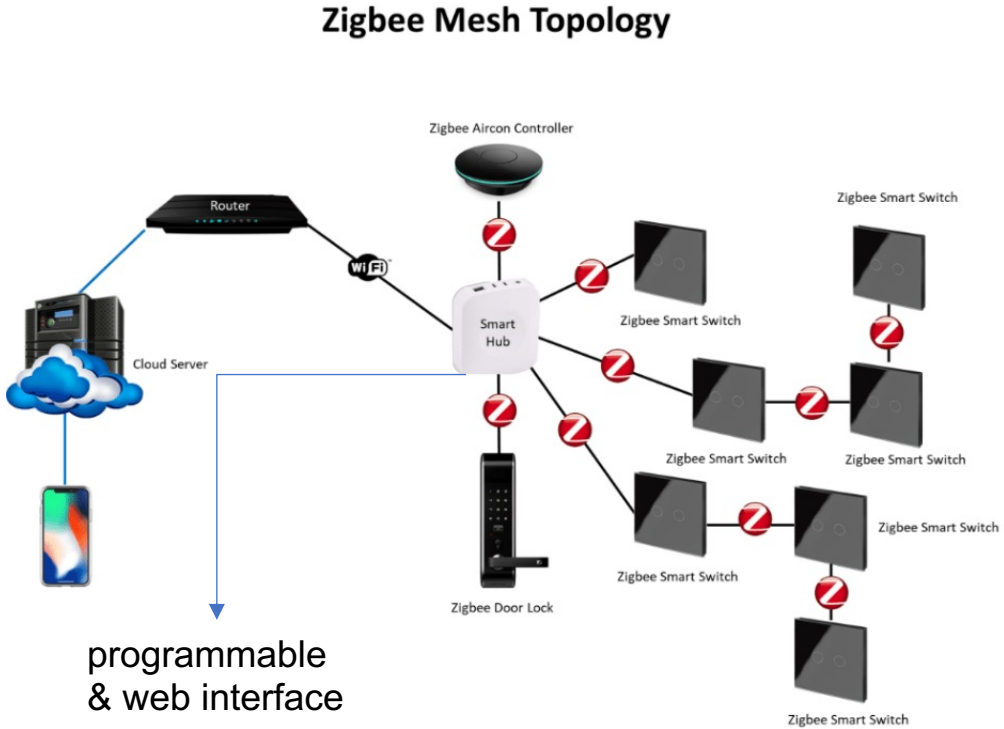
IoT is not a helpful term

- The only common thread is what doesn't matter: absence of a human
- Otherwise, spans every dimension of networking



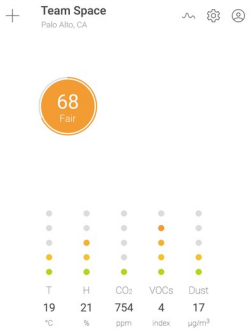
Smart home IoT

Original vision of home network

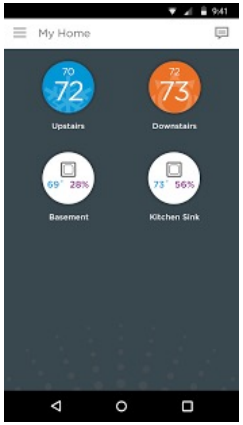


One Thing, one app

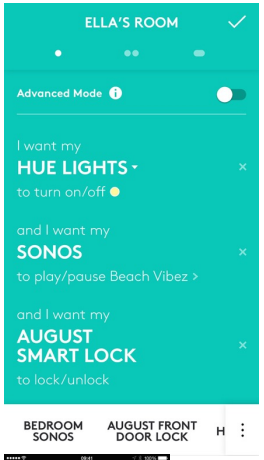
Awair



Honeywell

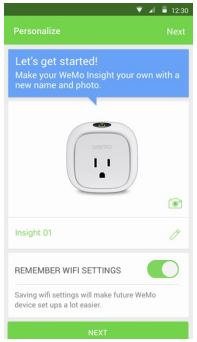


Logitech

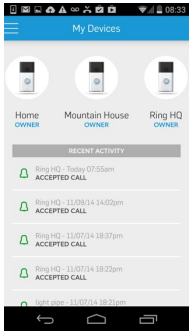


SATIS

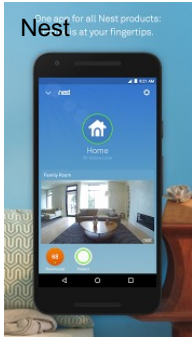
WeMo



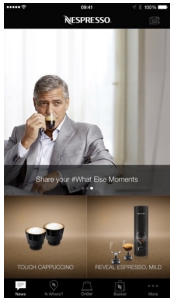
Ring



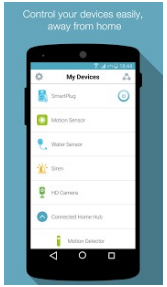
Nest



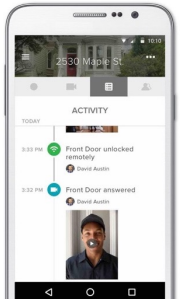
Nespresso



D-link



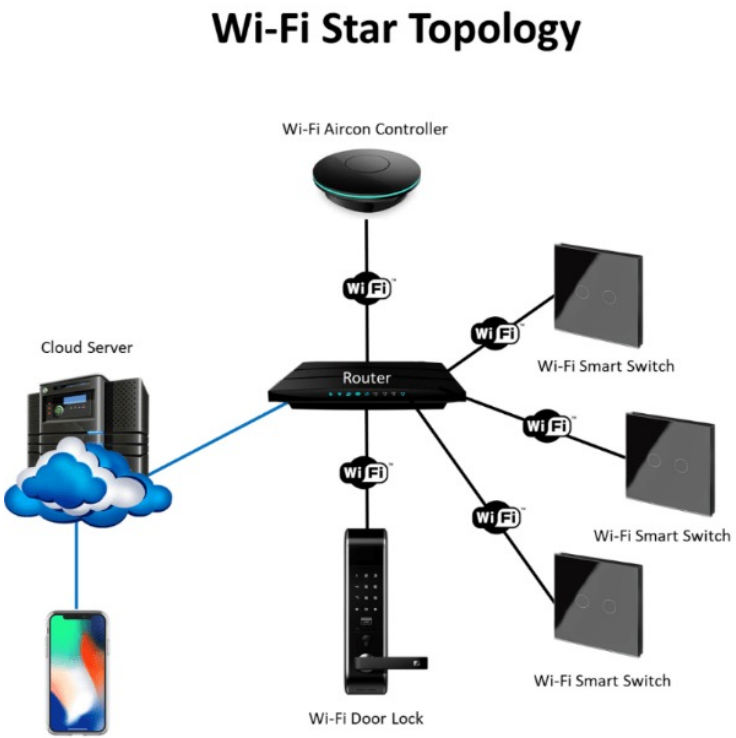
August



Common reality

IFTTT

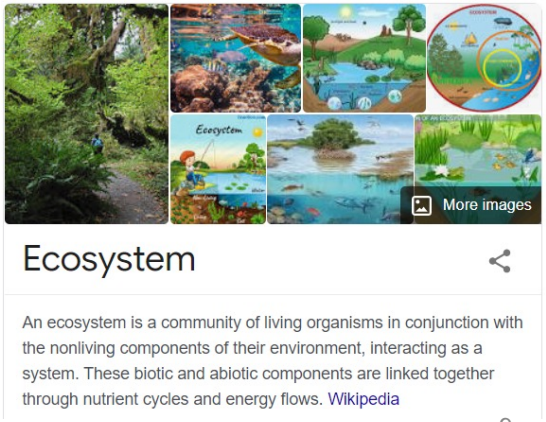
Works with Apple HomeKit



Challenges for Smart City

- Not a city-scale smart home
  - Strangers are everywhere
  - Huge number of sensors and actuators
  - Multiple sectors and administrations
- No longer application-specific or vendor-specific
  - Applications and interactions everywhere
  - Decouple IoT functions & software from device identities
- What makes an “IoT ecosystem”?
  - “living organisms”: active devices, users, applications
  - “nonliving components”: entities, attributes
  - “interacting as a system”: all-level communications

Source: https://www.arcweb.com/industries/smart-cities



What is the scale of outdoor and distributed applications?

one per household or business (e.g., 136M housing units in US)

31,100 in US
but often connected to city fiber network

268.8M (US) cars, trucks, …

5 million (US)
62,000 in NYC

26.5M (44M) in US
$2B energy cost / year
Boston: 64k street lights
but: often connected to fiber (5G!)

Goals (not necessarily at the same time)

- Robust naming of logical ”things”, observations, …
  - “turn off kitchen lights” as user interface, but doesn’t scale beyond single home
- Edge computing, reduced: move computation to the outer edge
  - reduce privacy risks
  - reduce network transmissions (= energy)
    - for IoT, most of the time, nobody cares about the individual measurement
    - ➔ aggregate & trigger
  - support small programming languages rather than full Docker stack
- Familiar programming interfaces
- Cover architectures from Arduino-class to Raspberry Pi
- Span multiple administrative domains

Current models: HTTP, CoAP and MQTT

- HTTP: REST model
  - largely device-centric (possibly aggregated via gateway)
  - polling, mostly
- CoAP: REST model
  - polling, mostly
- MQTT: event model
  - but only simple named events

There are only two hard things in Computer Science: cache invalidation and naming things. (Phil Karlton)

Geospatial IoT Naming

(in progress)

Protocols & networks matter, but programmability matters more

- Nobody wants to program raw protocols
- Most significant network application creation advances:
  - 1983: socket API → abstract data stream or datagram
  - 1998: Java network API → mostly names, HTTP, threads
  - 1998: PHP → network input as script variables
  - 2005: Ruby on Rails → simplify common patterns
- Many fine protocols and frameworks failed the programmer hate test
  - e.g., JAIN for VoIP, SOAP for RPC
- Most IoT programmers and smart city specialists will not be computer scientists (and won't have a telecom background)

Programs and people need names

- Addressing the wrong device could cause significant harm
- These do not work well (even if they are common…):
  - *IP address:* changes with DHCP and may be NAT (which 192.168.1.1?)
  - *MAC address*: changes with hardware swap-out
  - *domain name:* tempting, but risky – could be based on IP address, random, or some internal convention ("device17.local.net").
- Requirements:
  - no hardware or network topology dependence
  - work across administrative domains
  - reflect semantics meaningful to programmers → fewer errors
    - we don't program in memory addresses and disk blocks, either
  - allow resolution to one or more devices
  - failure is better than wrong device
  - where possible, allow devices to self-identify → location, "attached to"
  - clear semantics: address hardware unit or logical function – need both!

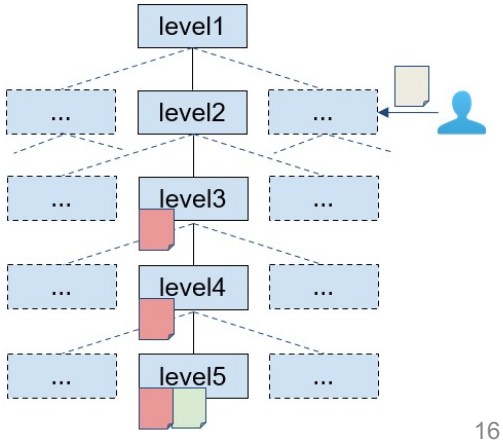
Directory-based approach

- Why directories?
  - Manage growth of data
  - Facilitate resource discovery
  - Provide naming resolution – hide changes
  - Encourage data sharing
  - Foster cross-domain services
- Model and architecture
  - Location-based hierarchical and distributed
  - Some sense of administration is needed
  - Access control for both lookups and access

```
my_location = my.location()
temperature_sensors = db.sensors.find({
    "type": "temperature",
    "location": $geoWithin(my_location, 5 miles),
    "interval": time.now()
    # ("2020-11-03 09:00:00", "2020-11-03 09:10:00")
    })

local_temperature = average (temperature_sensors.data)
```

A program to measure nearby temperature



Geospatial IoT device naming

- IoT device names are primarily for people (not machines)
  - including programmers!
  - think of program function and variable names
- People often refer to physical objects by their location
  - “Light switch in the **living room**”
  - “Security camera at First & Elm St”
- Test: “Please turn NAME on/off”, where NAME is a device name
  - NAME should be unambiguous and intuitive to another person
- Intuitive IoT device names often have a geospatial component
  - intuitive: easy for people to use and understand
- Missing building block: database of named geospatial objects

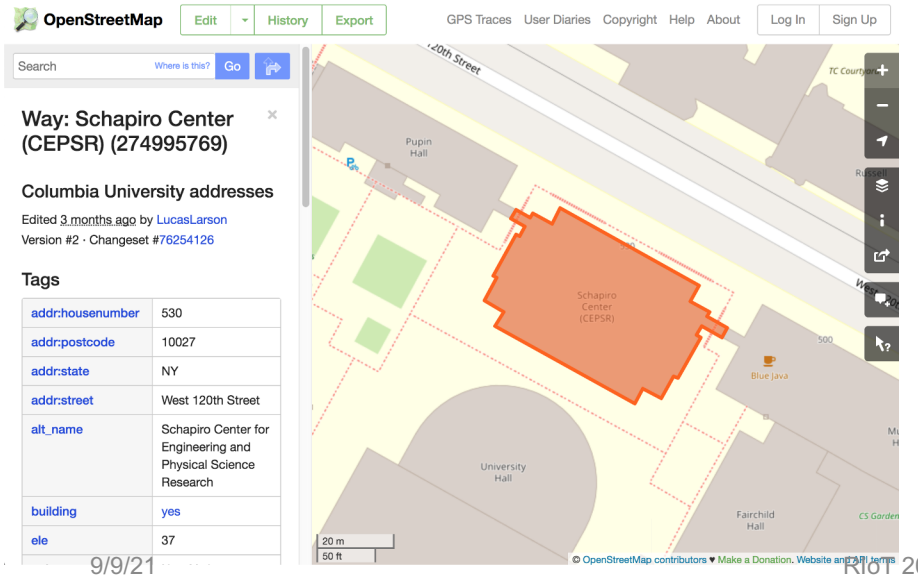
Sources of geospatial information

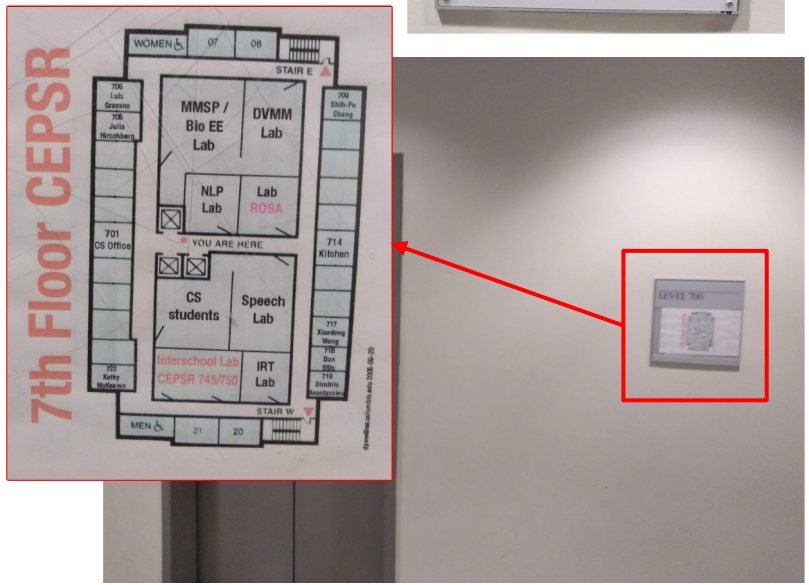
	Google Maps	OpenStreetMap	BIM	Floorplans
User Access	No	Yes	No	No
User Editable	No	Yes	Yes	No
Accurate	Yes (?)	No	Yes (?)	No
Public spaces	Yes	Yes	No	No
Private spaces	?	No	Yes	Yes
Dimensionality	3D	2D	4D (time)	2D
Standardized	No	De facto	Yes	No
(Reverse) geocoding	Yes	Yes	No	No
Offline gccess	No	Yes	Yes	Yes
Aerial imagery	Yes	No	No	No

Database of named geospatial objects

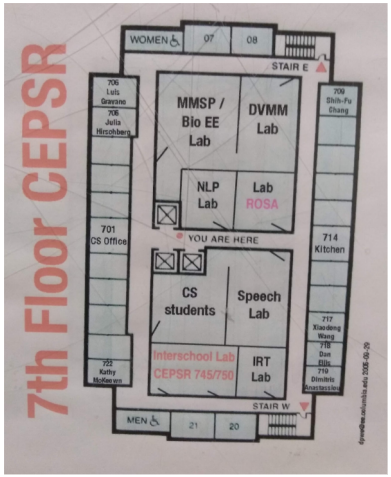
Combine user-accessible data sources:

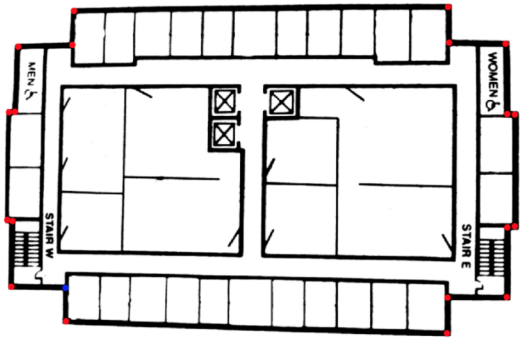
- OpenStreetMap building data (polygon in GeoJSON format)
- Fire escape floor plans (scanned images with floor/room info)
- Below: CEPSR 7th floor on Columbia University campus

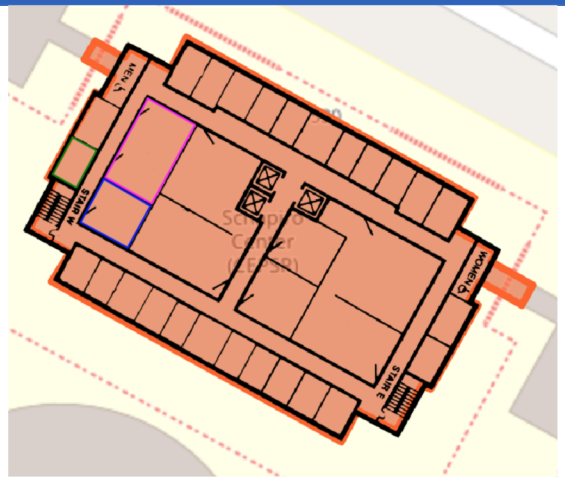




Converting raw data to geo database







- Download building polygon from OpenStreetMap in GeoJSON format
- Digitize printed fire escape floor plan & adjust (contrast, colors, transparency)
- Align floor plan with building polygon via reference coordinates (find best transformation using least-squares method)
- Define rooms by drawing polygons over floor plan
- Assign names to rooms

What's the result?

- Database of GeoJSON objects for rooms, sections, floors, and buildings
  - All points in WGS84 (longitude, latitude)
  - 2.5D model: GeoJSON augmented with altitude range (floor information)
  - Spatial indexes (MongoDB Geospatial or PostGIS)
  - GeoJSON objects associated with name sets
- Database of IoT devices
  - Each device has <longitude,latitude> information
- API
  - Select enclosing GeoJSON for <longitude,latitude> (reverse geocoding)
  - Select all devices inside a GeoJSON object

Programming model 1: directory + object

```
for d in devices("a1 = 'PA', a3 = 'Philadelphia', nam = 'Four Seasons Total Landscaping', type = 'CO'"):
  print(d.concentration)
```

```
SQL:
dlist = devices(SELECT device FROM device WHERE type='light'
  AND NAM = 'Four Seasons Total Landscaping')

for d in dlist:
   d.switch = true // translate to HTTP or CoAP in getter/setter
```

civic address data type
PIDF-LO (RFC 4776)

CAtype	label	description
1	A1	national subdivisions (state, canton, region, province, prefecture)
2	A2	county, parish, gun (JP), district (IN)
3	A3	city, township, shi (JP)
4	A4	city division, borough, city district, ward, chou (JP)
5	A5	neighborhood, block
6	A6	group of streets below the neighborhood level

Programming model 2: “pure” SQL

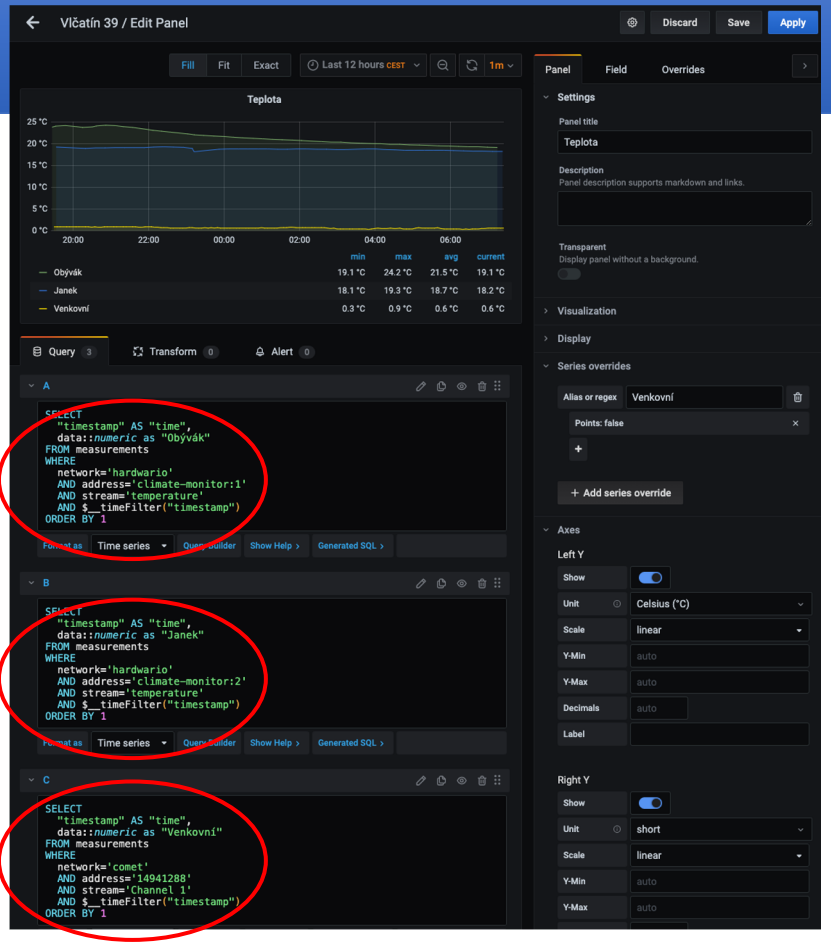
```
SELECT temperature, measured
FROM sensors
WHERE type = ‘light’
    AND a1 = ‘PA’
    AND a3 = ‘Philadelphia’
    AND nam = ‘Four Seasons Total Landscaping’
ORDER BY measured DESC
LIMIT 1
```

SenSQL

Storage and Data Processing Architecture
for Distributed Cyber-Physical Systems

Approach

- Present SQL interface to IoT applications
  - Standard, high-level, declarative, familiar, widely supported
- Proxy SQL queries to *relevant IoT storage nodes*
  - Design mapping service to discover storage nodes
- Aggregate response data for IoT application
  - Present unified view of spatio-temporal sensor data



SenSQL design goals

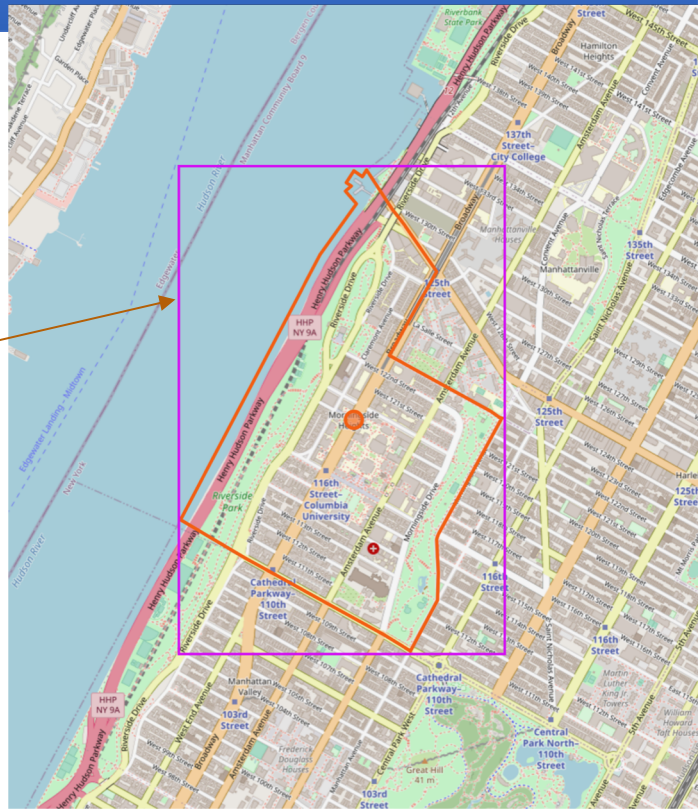
- Decentralized
  - Shards of sensor data stored at nodes near sensor devices
  - No single point of failure, e.g., common cloud infrastructure
- Federated
  - Storage nodes operated by independent administrative entities
  - Present unified view of spatio-temporal sensor data
- Query language (SQL) for applications
  - Standardized, high-level, declarative, familiar, widely-supported
  - Distributed query execution

Types of queries supported by SenSQL

- Find yesterday's maximum PM2.5 value in Manhattan
- Get daily minimum/maximum temperature near <latitude,longitude>
- Discover road-level temperature sensors along the road from A to B
- Find overhead light actuator in a CEPSR corridor but not in rooms
- Calculate daily average temperature in IRT lab (room)
- Find noise level sensor at W 120th Street & Amsterdam Avenue

Example: PM2.5 sensors in Morningside Heights

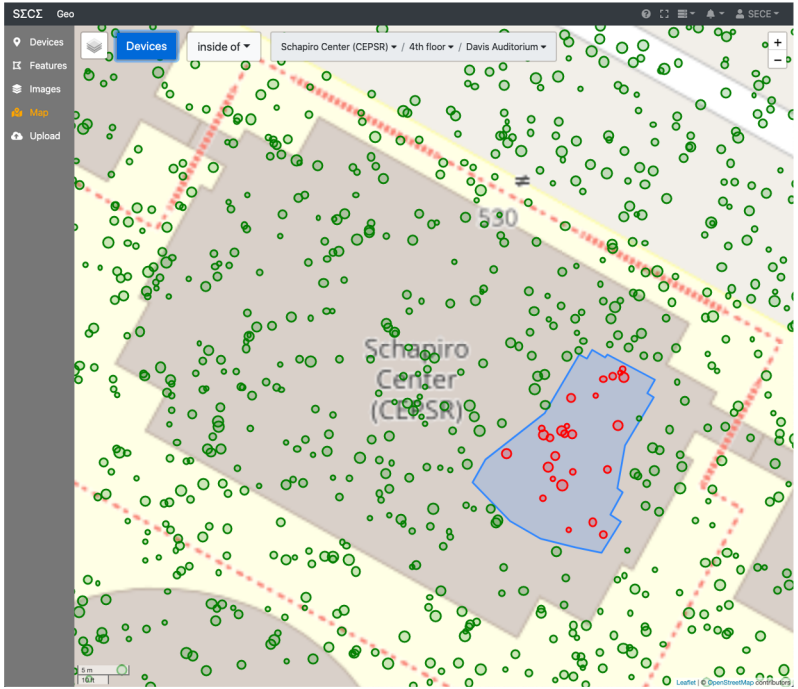
```
SELECT
   measurements.timestamp     AS ‘Time’,
   measurements.data::numeric AS ‘PM2.5'
FROM
   measurements, features
WHERE
   ST_Contains(features.bounds, measurements.location)
   AND features.name          = 'Morningside Heights'
   AND measurements.quantity  = ‘PM2.5'
   AND measurements.timestamp > "2020-01-01'
ORDER BY timestamp
```



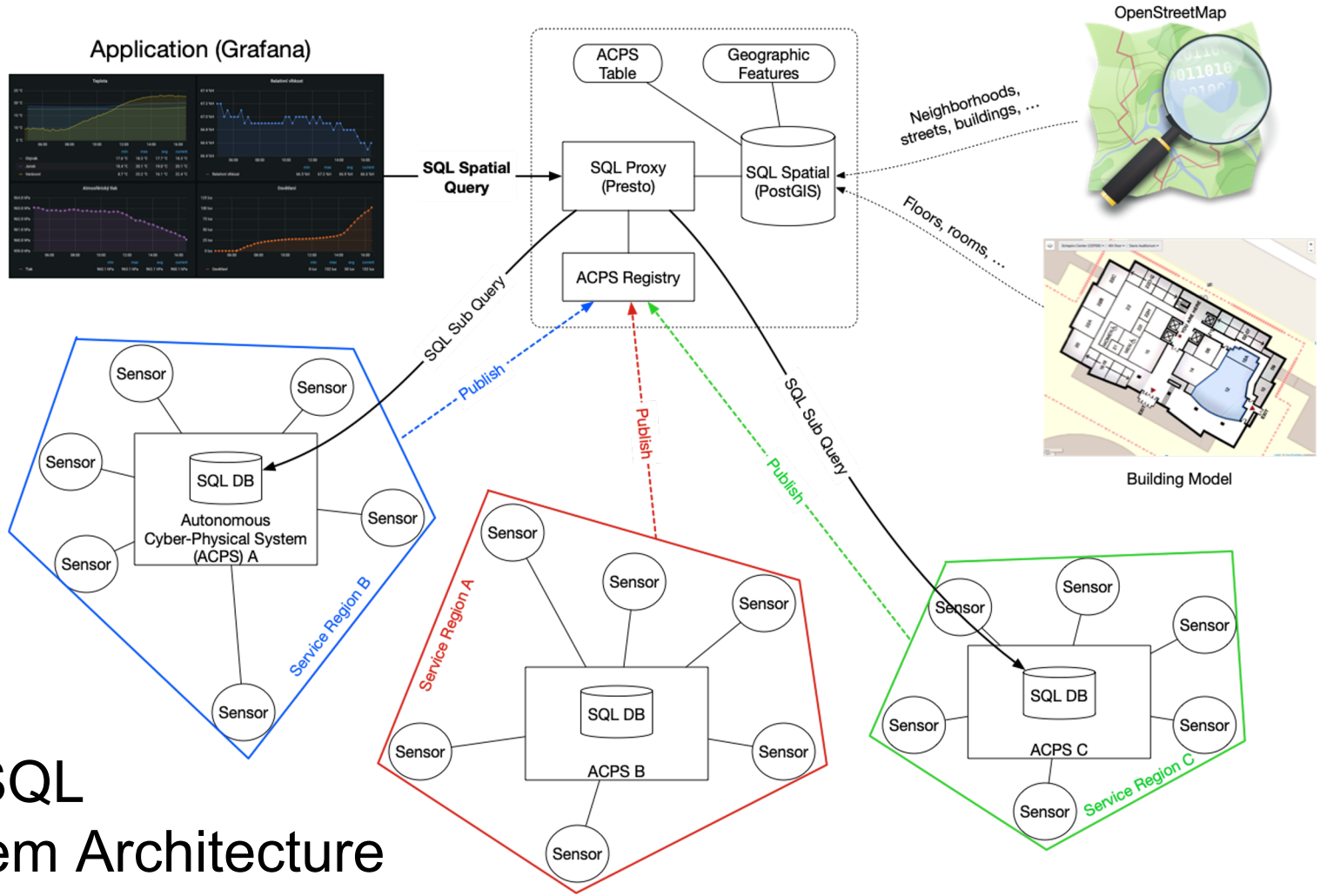
“Return the measurements ordered from January 1st, 2020 until now from all PM2.5 sensors within the Morningside Heights neighborhood.”

Example: CO_2 Sensors in Davis Auditorium

```
SELECT
   measurements.timestamp     AS 'Time',
   measurements.data::numeric AS 'CO2'
FROM
   measurements, features
WHERE
   ST_Contains(features.bounds, measurements.location)
   AND features.name        = 'Davis Auditorium'
   AND measurements.quantity  = 'CO2'
```



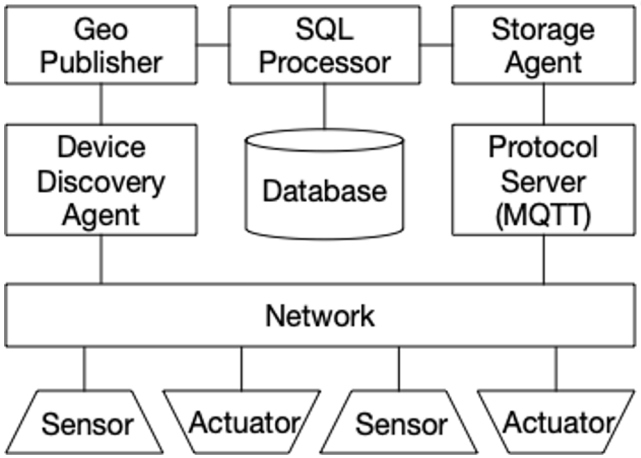
"Return the measurements from all CO2 sensors inside Davis Auditorium on Columbia University campus."



SenSQL
System Architecture

Autonomous Cyber-Physical System (ACPS)

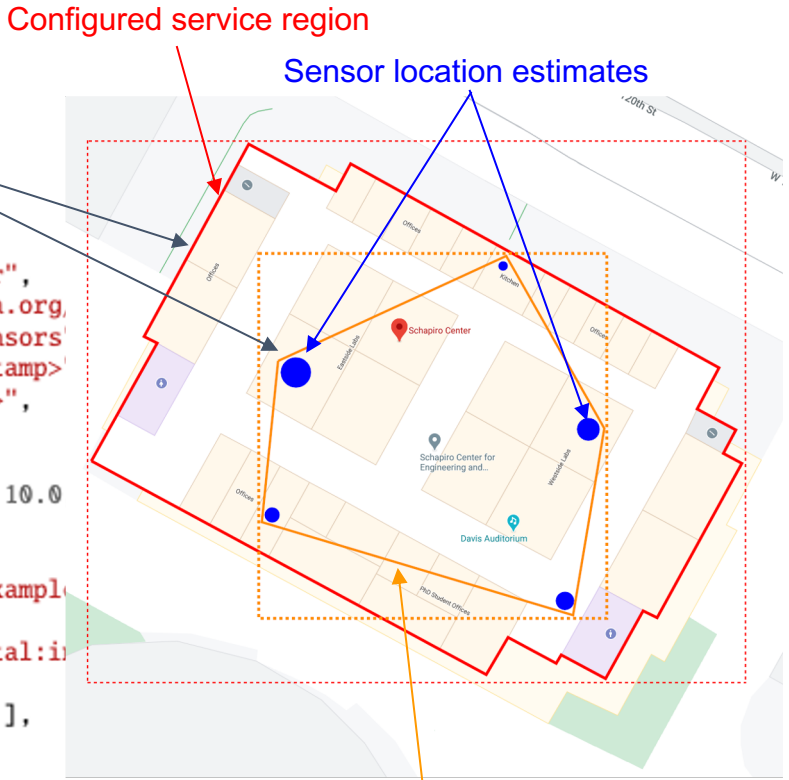
- Sensor & actuator infrastructure operated by a common administrative entity
- Geographically bounded by a **service region**
- Described by a **service descriptor**



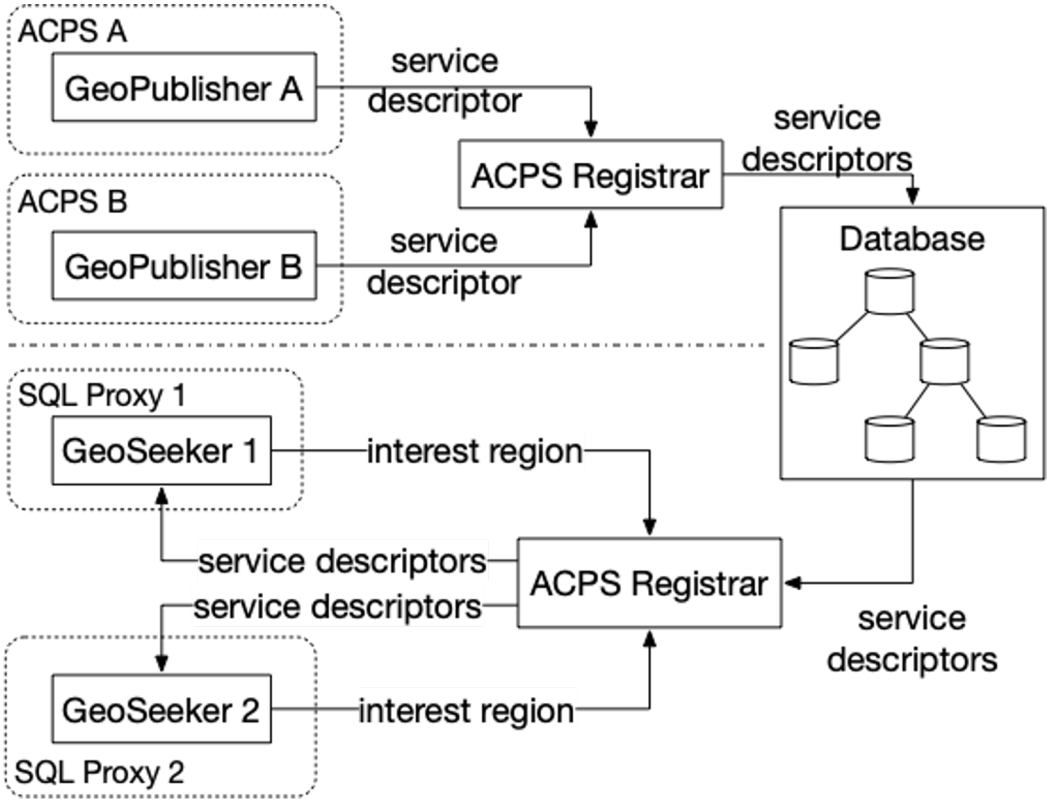
Internal architecture

```
{ "@type": "ServiceDescriptor",
"@context": "https://schema.org,
"displayName": "IRT Lab Sensors
"lastUpdated": "<ISO timestamp>
"expires": "<ISO timestamp>",
"coverageRegion": {
    "type": "Polygon",
    "bbox": [-10.0, -10.0, 10.0
    "coordinates": [] },
"private": false,
"contact": "wss://as-123.exampl
"sensorTypes": [
    "urn:sensor:environmental:i
    "urn:sensor:security",
    "urn:sensor:occupancy" ],
"signature": {} }
```

Service descriptor



ACPS registry & discovery service



Prototype demo with two ACPS nodes

```
presto> select * from measurements, map where map.name = 'Morningside Heights' and ST_contains(map.bounds, measurements.location) and type='zigbee';
        timestamp        |  type  | device
-------------------------+--------+--------
 2021-06-09 06:04:39.528 | zigbee | lock-2
 2021-06-09 06:04:39.529 | zigbee | lock-1
(2 rows)

Query 20210724_231606_00000_zuc2v, FINISHED, 1 node
Splits: 18 total, 18 done (100.00%)
0:07 [2 rows, 0B] [0 rows/s, 0B/s]
```

```
presto> select * from measurements, map where map.name = 'Butler Library' and ST_contains(map.bounds, measurements.location) and type='zigbee';
        timestamp        |  type  | device
-------------------------+--------+--------
 2021-06-09 06:04:39.529 | zigbee | lock-1
(1 row)

Query 20210724_231716_00001_zuc2v, FINISHED, 1 node
Splits: 17 total, 17 done (100.00%)
0:03 [1 rows, 0B] [0 rows/s, 0B/s]
```

```
presto> select * from measurements, map where ST_contains(map.bounds, measurements.location) and map.name = 'Lerner Hall' and type='zigbee';
        timestamp        |  type  | device
-------------------------+--------+--------
 2021-06-09 06:04:39.528 | zigbee | lock-2
(1 row)

Query 20210724_231911_00002_zuc2v, FINISHED, 1 node
Splits: 17 total, 17 done (100.00%)
0:03 [1 rows, 0B] [0 rows/s, 0B/s]
```

Future SQL opportunities

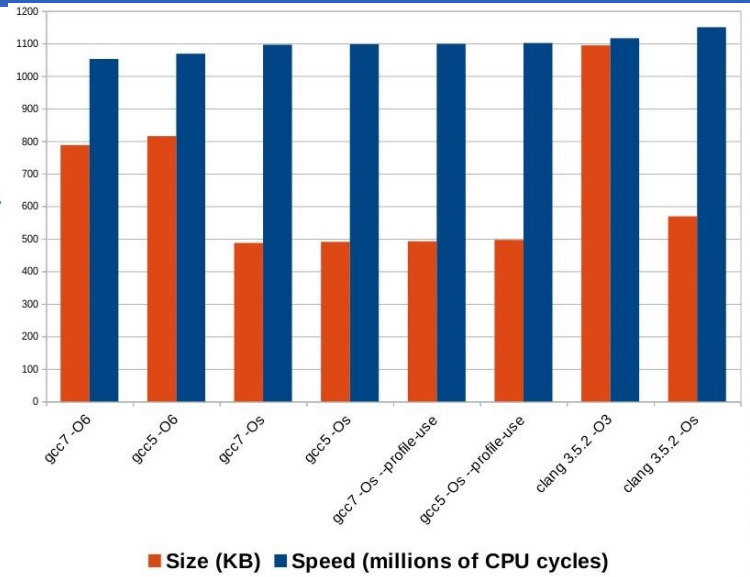
- ”REST” model for actuators: UPDATE and INSERT into (time) tables
  - “set actuator to 21C for 2021-09-09 11:59”
    - INSERT INTO actuators SET temp=21, action=“2021-09-09 11:59”,device=“name”
  - “update all actuators in region to X”
- Triggers → MQTT events?
- Time-series SQL databases (TimescaleDB, Druid, …)

SQL for sensor networks is not new

- Example: TinyDB had SELECTs in 3 KB (early 2000's)

Target Devices		Mobile DBMSs
Extremely Small Devices with Low Computing Power	Sensors	TinyDB
	Smartcards	PicoDBMS
Small Devices with High Computing Power	Cell Phones, PDAs, Car Navigators, Ultra Books	Sybase SQL Anywhere, Oracle Lite, MS SQL Server CE, SQLite IBM DB2 Everyplace

But SQL for embedded systems?



Sqlite µLogger for Arduino

Sqlite µLogger is a Fast and Lean database logger that can log data into Sqlite databases even with SRAM as low as 2kb as in an Arduino Uno. The source code can be ported to use with any Microcontroller having at least 2kb RAM.

Development environment

Development language	DeviceSQL language (Industry standard Oracle PL / SQL compliant) C / C ++ (Embedded SQL API / Native API)
Tool	DeviceSQL Compiler Sample application (StarterSamples) DeviceSQL SQLProbe
Development host OS	Windows 7/8/10 Linux

Target environment (Runtime)

Footprint	50KB ~. Depends on configuration and compiler type
DB type	Library type
Data model	Relational Data stream
Maximum database size	4TB (4096GB)

Work in progress

- Unified sensor data model
- Event notification
- Architecture of ACPS registry & discovery service
- Distributed SQL scheduler algorithm
- Prototype evaluation (performance & scalability)
- Topology change notifications

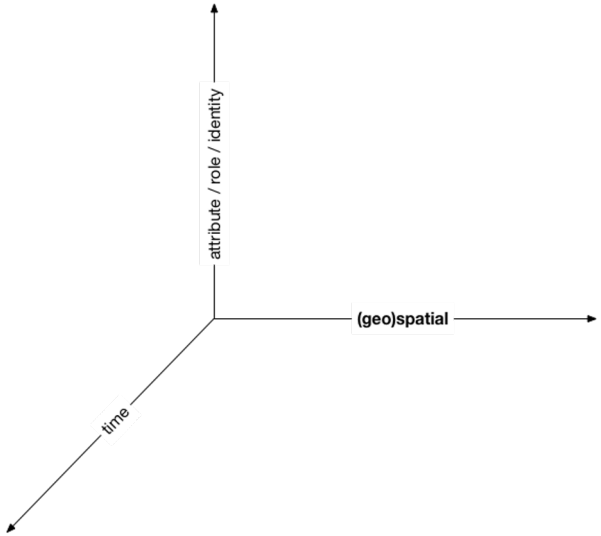
Geospatial Access Control+

Early Idea

IoT access control is more complicated

- Not just “user X has read access to file”
- ”Room occupant can set temperature to 65 to 80 degrees; facilities personnel can set it to any temperature”
- “Staff can read temperature for their office *from anywhere*, but only set temperature when *in that room*”
- “Public can read *day-averaged* occupancy sensor value, but not current occupancy”

Access control problem for IoT

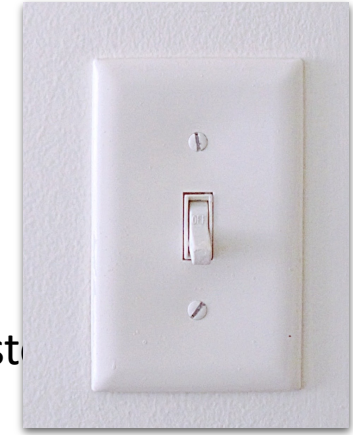


- attribute, role & identity
  - RBAC, ABAC
  - challenge: who provides attributes?
- time
  - calendar-like functionality
  - also frequency of access
- (geo)spatial
  - not well understood
  - rooted in the physical world
  - intuitive

Real-world IoT access control policies will be a combination of all three
Example: access control on university campus

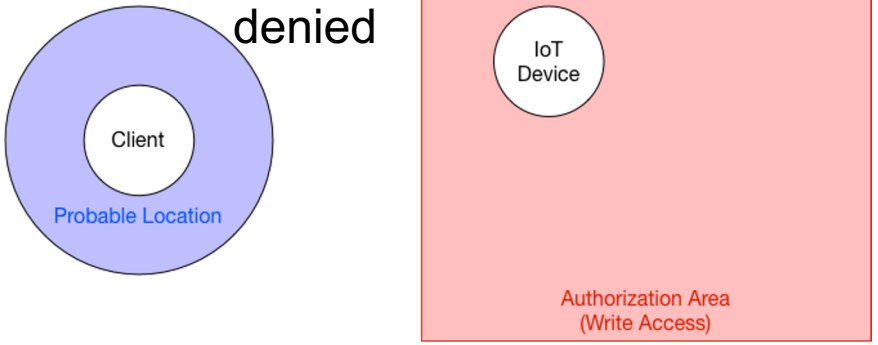
Access control based on geospatial relationships

- Access control policies based on geospatial relationships, e.g.,
  - ”Write access if ‘near’, otherwise read-only access”
  - ”Write access if in the same room”
  - "Discoverable on campus”
- Complement role and attribute access control mechanisms
  - Use identity & role access control if identity is known
  - Fallback to geospatial access control if identity/role unknown or untrust
- Access control rules inspired by the physical world
  - People in a room can implicitly control light switches in the room
  - Access determined by client location relative to the controlled device

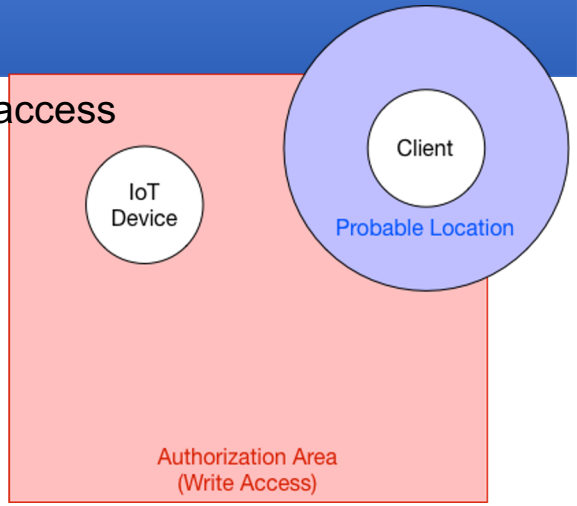


Probable location overlaps determines access

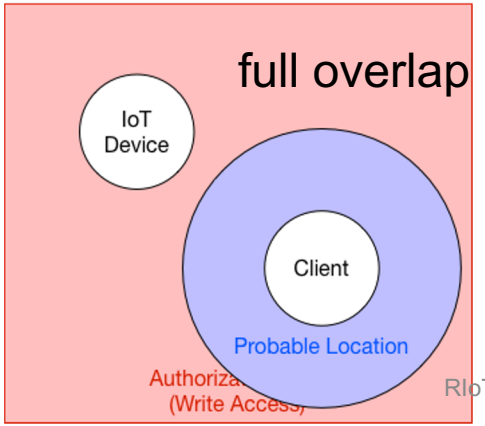
no overlap → access denied



partial overlap → access denied



full overlap → granted

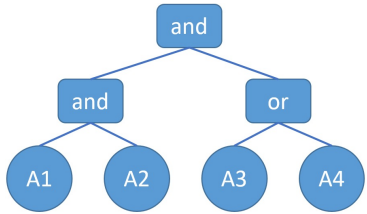


Access Control for IoT

- Commercial solutions
  - Per-device granularity (identity-based)
  - All-or-nothing access right
  - Up to Role-Based Access Control (RBAC)
- Academic solutions
  - Per-capability granularity
  - Attribute-Based Access Control (ABAC)
  - Capability-Based Access Control (CapBAC)

Access control for IoT

- ABAC
  - Take multi-dimensional characteristics
    - subjects, objects, environments
  - Policy: set of attributes-value pairs
    - e.g., {sbj.department=CS, sbj.title=professor, obj.type=sensor, obj.location=campus, action=read, freq=3/hour}



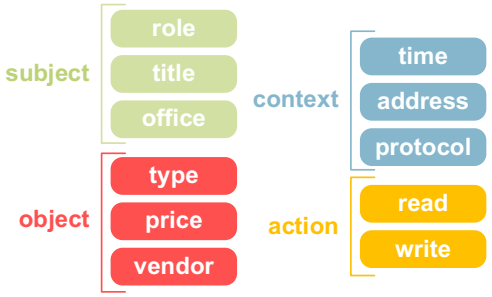
- CapBAC
  - A communicable, unforgeable token assigned to user
  - Address “role explosion” in RBAC
  - Claimed to be “finest grained”

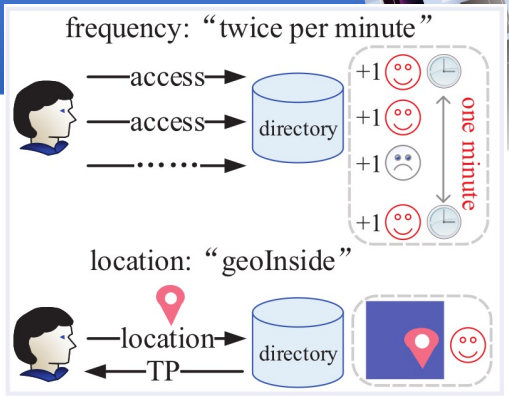
Design principles

- Fine-grained access support
  - provide sufficient system-wide supports to ensure that the access control remains fine-grained in practice
- Least-attribute principle
  - inspired by least-privilege principle
  - retrieve the smallest (or least-sensitive) set of attributes to complete an authorization
- Privacy preserving
  - when and what sensitive attributes are used is controlled by the user
- Local maintenance
  - subject and object information is maintained by distributed systems
- Resource-constrained device support
  - support but not burden devices with limited resources

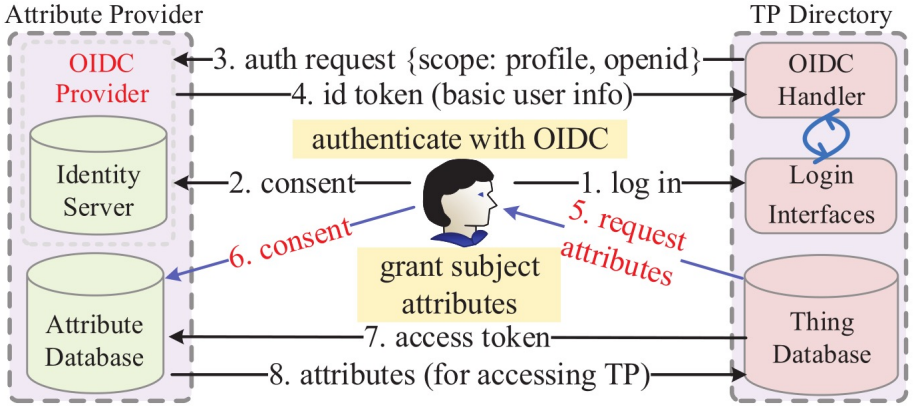
DBAC: Phase one

- Classify and retrieve attributes
  - Subject attributes
    - locally stored or OpenID (OIDC)
  - Object attributes
    - attributes in TDs
  - Environment attributes
    - attributes from third-parties
  - Stateful attributes
    - access frequency





Access frequency and geoInside condition



Cross-domain subject attribute retrieval

There’s work to do…

- Think beyond CoAP, HTTP, and MQTT (and SPARQL)
- What abstractions are easiest to use for programmers who don’t know protocols?
- Where should this functionality reside?
- Where should the data reside?
- How should we handle more advanced functionality such as object recognition and other local ML-enabled functions?
- How can we make access control less confusing and avoid surprises?