Open Source Projects for a suite of Web of Things Servers

Sunnyvale F2F

Dave Raggett <dsr@w3.org>

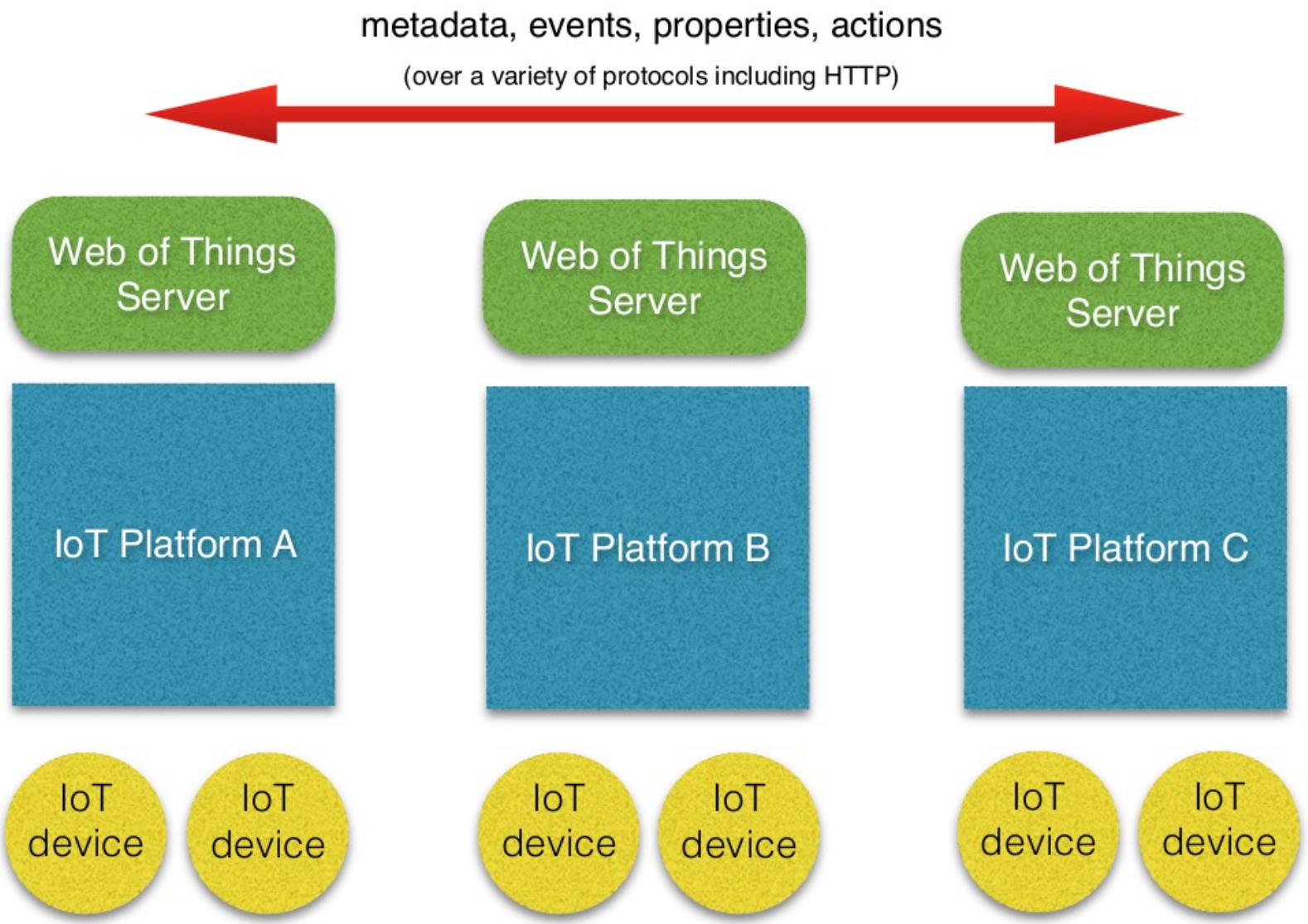
29th July 2015

This work is supported by the European Union's 7th Research Framework Programme (FP7/ 2013-2015) under grant agreement n°317862 - Compose

Web of Things

- Web of Things is a distributed platform of platforms
  - Designed as an abstraction layer to connect different IoT platforms
    - Connecting the silos to enable the benefits of the network effect
    - Things as virtual representation of physical or abstract entities
    - Things as proxies for things on other servers
    - Things are bound to domain semantics using linked data
      - As basis for discovery and interoperable combinations of services
    - Things as virtual objects with events, properties and actions
      - For easy scripting in many languages
    - Object model decoupled from the transport protocols
      - Different protocols are needed in different contexts
      - Protocol bindings map objects to data streams
    - Different kinds of inter-server messages
      - Events, property updates, action invocations, action results, metadata updates

The Web as the Global Data Bus



Web Servers at Many Scales

Web of Things servers can be realised at many scales from microcontrollers to clouds

Home Hub: home/office server for access to smart home and wearables, running behind firewall

Micro-controller: resource constrained, IoT devices or gateways, CoAP, running behind firewall

Smart Phone: personal server for access to smart home and wearables

Cloud-Based: highly scalable server for many users, devices and working with big data

Distributed System

It is all in the metadata!

- Different kinds of sensors have very different requirements
  - Smart meters vs Security Cameras vs ECG vs ...
    - Small amounts of data that isn't time critical
    - Large amounts of data that is needed in real-time
    - Privacy sensitive data e.g. health sensors
  - Sensor streams where you need a log of readings over time
    - Ability to query data for a specific time or time window
    - Interpolation between readings for smoothly varying properties
  - Multiplexing data from sensor networks
- Pushing Interpretation to the Network Edge
  - Upload scripts to Web of Things server (hubs)
  - Reduces amount of data to be sent over network
- Pushing control to the Network Edge
  - Clock synchronisation across group of controllers
    - Coordinated control of actuators, e.g. traffic lights, factory floor
    - Programming path of robot hand via smooth control of its joints
- The need to collect representative use cases

Accessing Things from Web Browsers

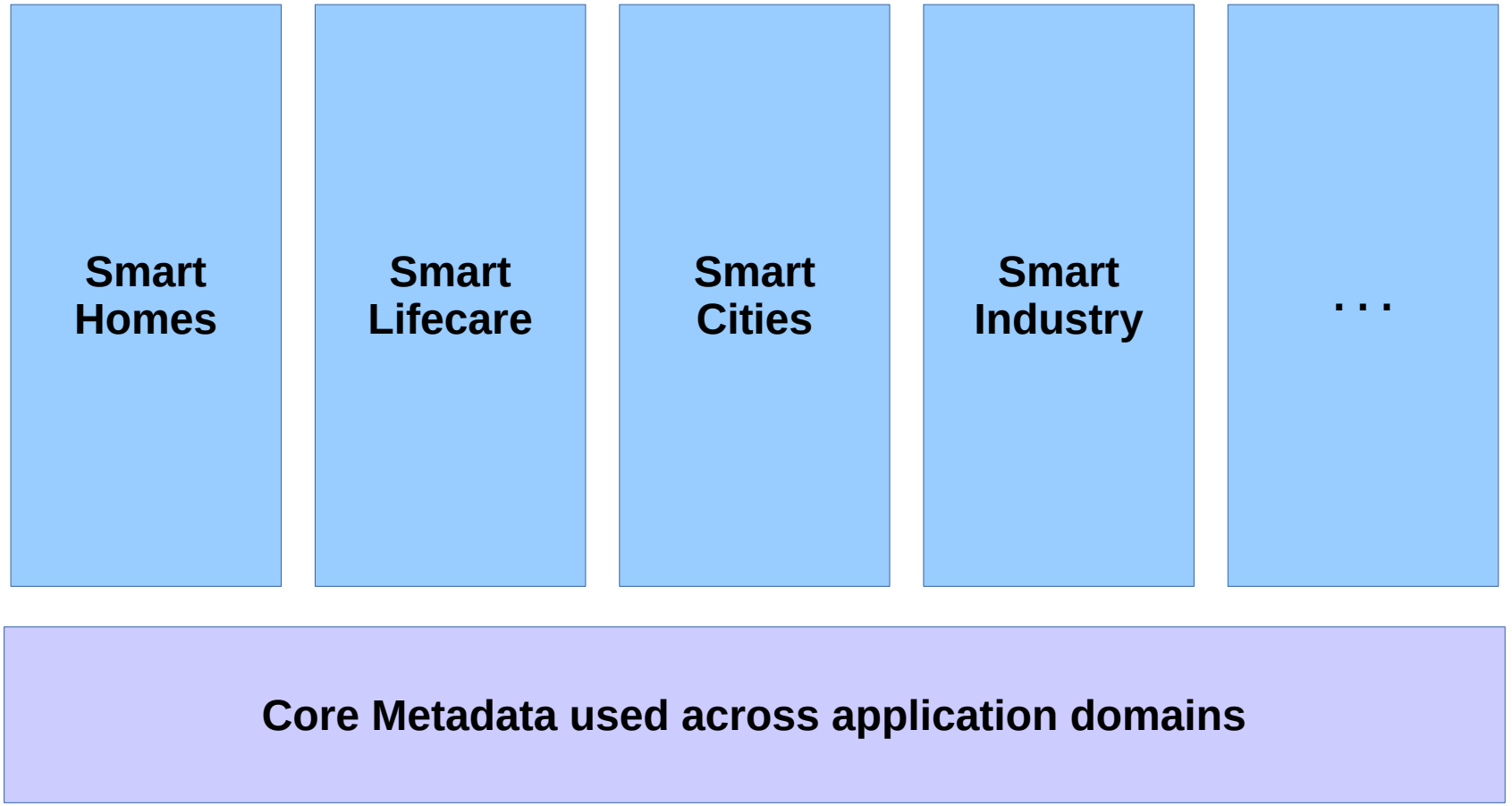
- Web page scripting library for accessing “things”
  - Library creates local object in web page script execution environment as proxy for remote thing
- Web browsers support APIs for a limited set of protocols **
  - HTTP – Working Draft 30 January 2014
  - Web Sockets – W3C Candidate Recommendation 20 September 2012
  - Server sent events – W3C Recommendation 03 February 2015
  - Push API – Working Draft 27 April 2015
  - WebRTC – Working Draft 10 February 2015
    - Real-time communications between Web Browsers
    - In principle, WebRTC data channel could be applied to proxying things
- Browser web security model
  - Single origin policy limits scripts to same server as was used to load the web page
  - Server could support HTTP and Web Sockets
    - Use HTTP to access thing data models
    - Use WebSockets for messaging
  - CORS as a way of relaxing single origin policy – W3C Recommendation 16 January 2014
    - Thing server uses CORS header to indicate which web origins it authorises browsers to use it from

** Further protocols could be supported via browser specific add-ons, see e.g. Firefox Copper

Open Source WoT Servers

- Why do we need this?
  - Open source servers played a vital role in early growth of the Web
  - Rough consensus and running code vital for success of standardisation work
- Suite of GitHub projects at various stages of progression
  - NodeJS – https://github.com/w3c/web-of-things-framework
    - HTTP for accessing models,
    - WebSockets for inter-server messaging, other protocols to be added later
    - Browser library for access from Web page scripts
  - Go language
    - Just at the beginning
    - Focus on scalable high performance server
  - Arduino – https://github.com/w3c/wot-arduino
    - Focus on minimal memory footprint, expect to support CoAP and MQTT-SN
    - After we get this to work, extending to other microcontrollers will be easy!
  - ESP8266
    - Just at the beginning – will be clone of Arduino server + small mods
  - Miscellaneous Microcontrollers (to be started)
    - ARM, Intel, Texas Instruments, etc.

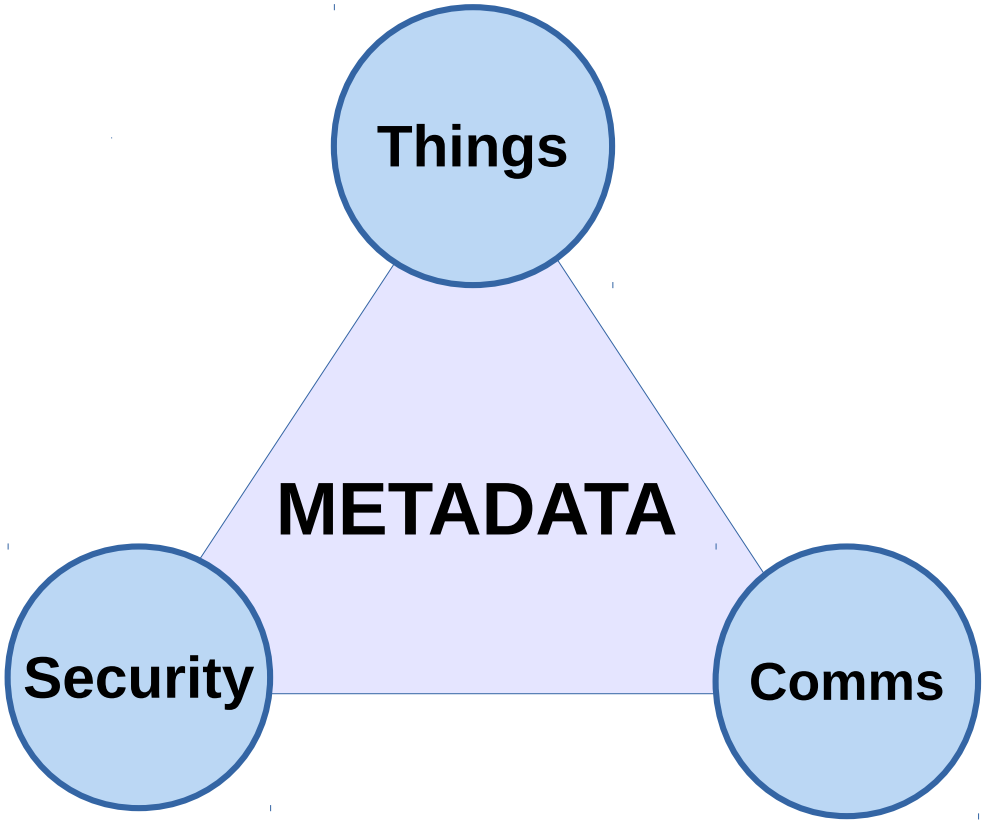
Horizontal & Vertical Metadata



Industry specific groups are in best position to define metadata for each vertical

Focus of W3C Contribution

Core metadata applicable across application domains



- **Thing descriptions**
  - Links to thing semantics
  - **Data models & relationships between things**
  - Dependencies and version management
  - Discovery and provisioning
  - Bindings to APIs and protocols
- **Security related metadata**
  - Security practices
  - Mutual authentication
  - Access control
  - Terms & conditions
    - Relationship to “Liability”
  - Payments
  - Trust and Identity Verification
  - Privacy and Provenance
  - Resilience
- **Communication related metadata**
  - Protocols and ports
  - Data formats & encodings
  - Multiplexing and buffering of data
  - Efficient use of protocols
  - Devices which sleep most of the time

Data Models

- A standard vocabulary for thing data models will enable servers to construct virtual objects for things for use by application scripts
- Enabling developers to interact with things without needing to know about the underlying transport protocols
- The aim is to enable the description of data models, not to standardise those models, which are likely to be domain specific
- To be able to link to domain specific semantic models for interoperability based upon shared semantics.

Data Models

- Properties values ranging from basic to complex types
- Basic types including null, boolean, number, string, enumeration
- Complex types including arrays and objects (a mapping from names to values)
- Events can carry data
- Actions can be passed data when invoked, and may asynchronously yield data as a result
- Streams as a first class data type, e.g. ECG sensor stream
  - Sequence of data points
    - Simple values vs complex values with named properties
    - Buffered for access to historic values
  - Metadata, e.g. time stamps or sampling interval
  - Access methods

Example Data Models

- Let's consider a example for a hotel room
  - Door has a card reader and a bell
  - Room has a light
- We want to unlock the door and turn on the room's light when the correct card is presented
- Describe things using JSON-LD
  - Serialisation of RDF in JSON
  - W3C Recommendation Jan 2014
    - http://www.w3.org/TR/json-ld/
  - Bridges cultural gap between web developers and the Semantic Web

Thing Descriptions

Server uses URI for a thing to download its description
and create a local proxy object for use by scripts,
this applies recursively for properties that are things

- Door

```
{
    “@events” : {
        “bell”: null,
        “key”: {
            “valid” : “boolean”
        }
    },
    “@properties” : {
        “is_open” : “boolean”
    },
    “@actions” : {
        “unlock” : null
    }
}
```

- Light switch

```
{
    “@properties” : {
        “on” : {
            “type” : “boolean”,
            “writable” : true
        }
    },
}
```

TDL's default JSON-LD context defines bindings of core vocabulary to URIs
Data models may be defined explicitly or by reference to an external definition

Thing as Agent

- Thing description

```
{
    “@properties” : {
        “door” : {
            “type” : “thing”,
            “uri” : “door12”,
        },
        “light” : {
            “type” : “thing”,
            “uri” : “switch12”
        }
    }
}
```

- It's behaviour

```
// invoked when service starts

function start () {
    door.observe(“key”, unlock);
}

function unlock(key) {
    if (key.valid) {
        door.unlock();
        light.on = true;
    }
}
```

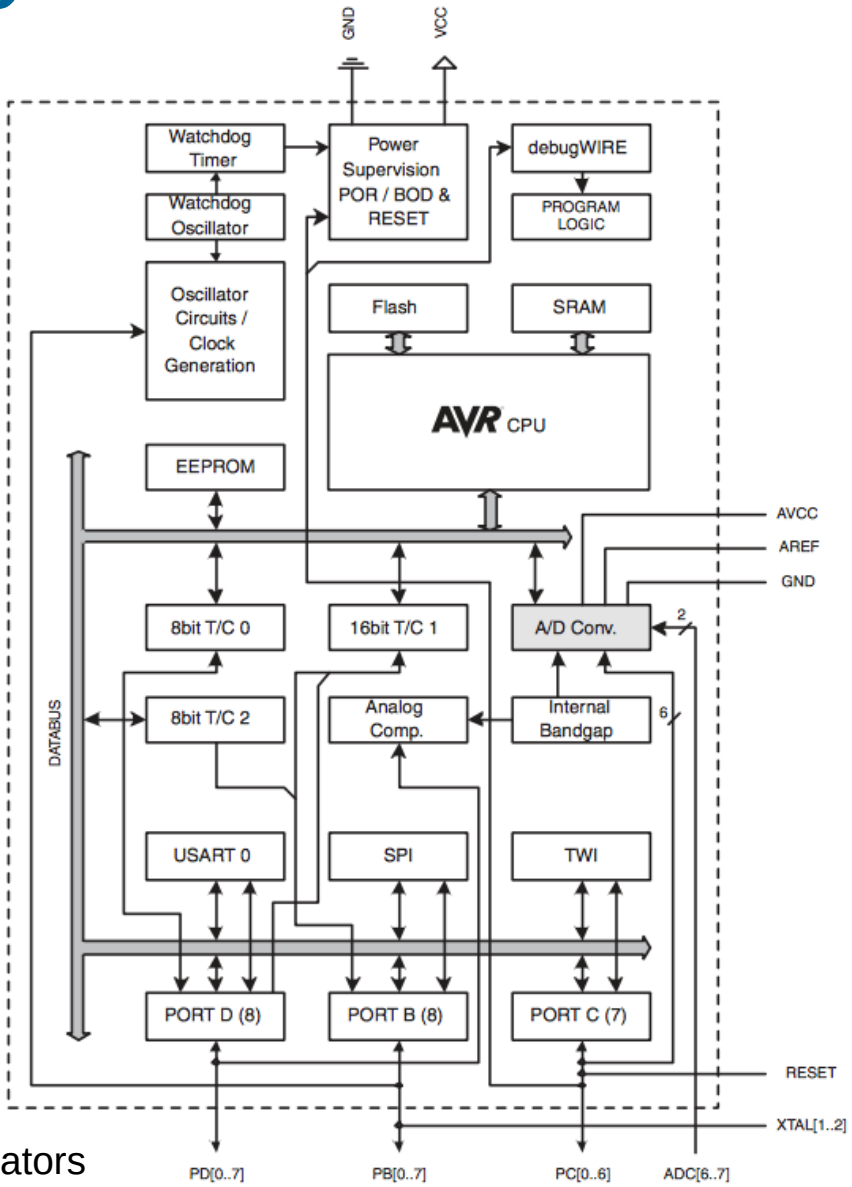
This “thing” is an agent that is bound to a specific door and light switch.
It unlocks the door and turns on the light when a valid key is presented.

Designing a Web of Things Server

- Starting point is the data model for a thing
  - What are its events, properties and actions?
- You can register a thing along with its data model and implementation
  - Server creates virtual object based upon the thing's data model and binds it to the implementation and network protocols
  - Server publishes thing's data model for other servers to retrieve
- You can register a proxy for a thing on another server
  - Server downloads thing's data model and creates virtual object with bindings to network protocol
    - You can script the proxy object as if it were local
    - Server transparently manages the messaging to make that work
  - You can create chains of proxies if required
- Different servers may support different protocols and encodings
  - Server publishes metadata describing which protocols etc. it supports
    - At a well known location, e.g. /.well-known/protocols

Embedded Systems

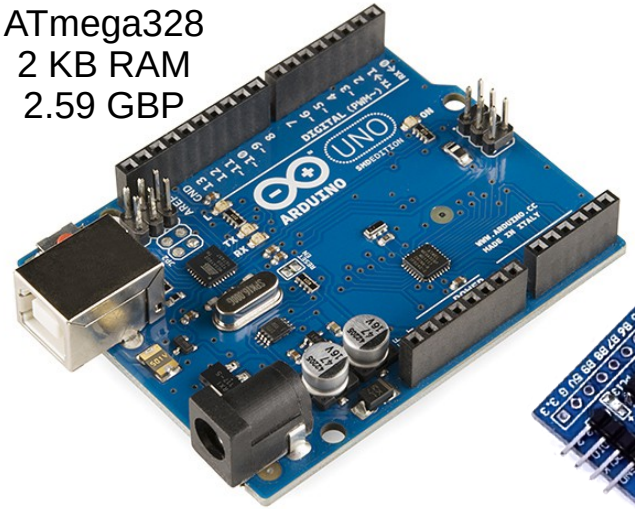
- IoT devices are typically embedded systems
  - Microcontroller plus sensors and actuators
  - Often battery operated and designed to work for months or years
    - Variety of power saving modes
      - If RAM state is not preserved, need fast-reboot
- Resource constrained
  - RAM with Kilo bytes not Giga bytes!
    - Arduino Uno uses ATmega328 which has 2 Kbytes RAM
  - Flash for program code and static data
  - EEPROM for configuration data
    - Limited number of update cycles
- Harvard vs Von Neumann CPU architecture
  - Harvard architecture has separate paths for data and code
- Interrupts, Timers and Real-Time Clocks
- Data bus between chips
  - I2C, SPI and CAN
    - Access to Flash, EEPROM, and other microcontrollers (e.g. in a car)
    - Access to sensors, e.g. MPL3115A2 barometric pressure & temperature
  - USART for serial connection to host computer
- GPIO, ADC, PWM pins for low level interfacing to sensors/actuators
  - Turn on a LED, control a servo, read analog value for ECG



Building Momentum through the Maker Community

- Open hardware and open source software are a huge opportunity for a bottom up approach to growing the Web of Things
  - *Let's have lots of hands on fun!*

Arduino Uno
ATmega328
2 KB RAM
2.59 GBP



ARM STM32
20 KB RAM
64KB Flash
3.03 GBP

ESP8266 96 KB
RAM, 512KB Flash
32 bit MCU + WiFi
1.5 GBP

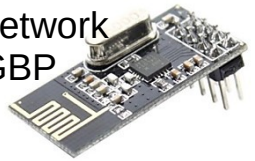
CoAP: REST for IoT devices
MicroCoAP: 16 KB including the Ethernet library, for more see:
https://github.com/1248/microcoap

MQTT as a lightweight binary Pub-sub protocol with brokers, see:
https://github.com/knolleary/pubsubclient

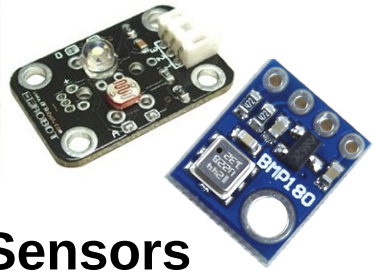
NodeJS based Web of Things server with many libraries available for IoT (run on Raspberry Pi as Home Hub)

ATECC108
ECC Crypto

nRF24L01 2.4 GHz
Sensor Network
1.34 GBP



CC2530: 8051 MCU + IEEE 802.15.4
Up to 256 KB flash + 8 KB RAM
Available for 6 USD

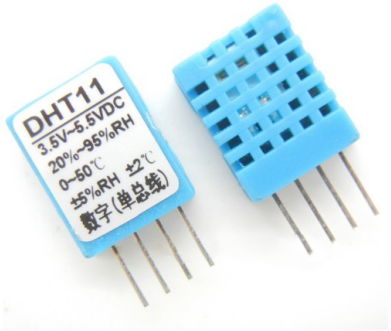


Sensors

C++ & Arduino IDE
Lua & NodeMCU
MicroPython
RIOT OS

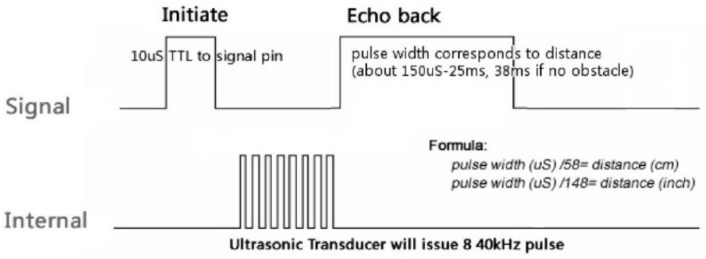
Demo Ideas

Should have these working for TPAC 2015

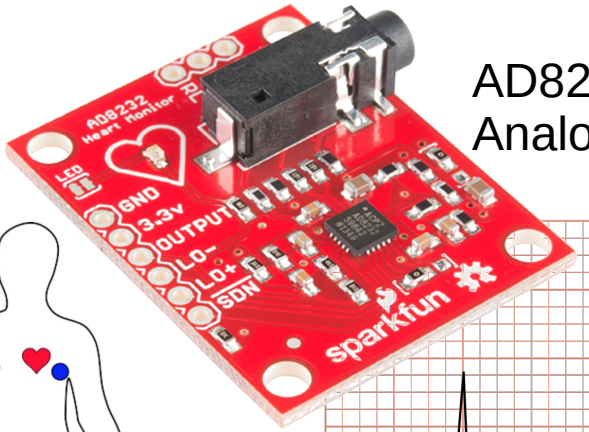


DHT11 temperature & humidity
Ad hoc one wire interface

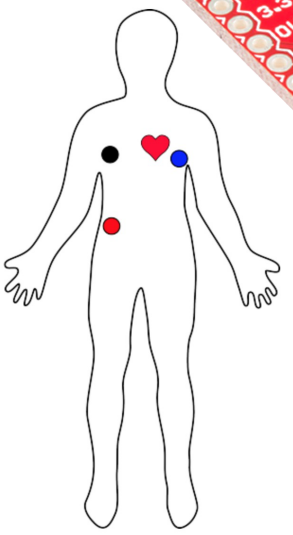
HC-SR04 proximity
2cm to 4m range

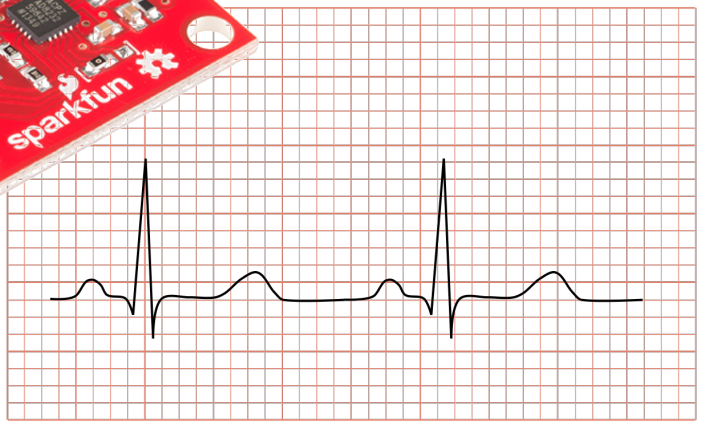


BMP180 temperature & pressure
I2C and SPI bus interface



AD8232 ECG
Analog output





Hundreds of samples a second

Arduino Server

https://github.com/w3c/wot-arduino

- A work in progress — goal is to enable WoT hackathons in 2016
- Stretch challenge: can we create a Web of Things server that will work with the 2 Kbytes RAM in an Arduino Uno?
  - Statically allocate memory pool for JSON nodes
    - true, false, strings, numbers, objects, arrays and null
    - 6 bytes per node on ATmega328 *and* on 32 bit MCU's
      - Nodes can be formed into linked lists if needed with no extra memory
        - Assuming pool of up to 4095 nodes and a extra list node for each string
  - AVL trees for representing objects and arrays
    - Approximately balanced binary tree with 6 bytes per node
      - Assumes limit of 255 properties per object and items per array, and pool of 65535 nodes
        - Or perhaps 1023 object properties/array items and pool of 16383 nodes
    - Shares node pool with JSON
- Map names to numeric symbols when parsing a thing's data model
  - Saves memory and enables compact messages
    - Single byte for JSON tags and 200 different symbols
  - Can pre-parse models on larger servers
- Statically typed versus dynamically typed languages
  - More cumbersome to work with, but not too bad
    - C++ not nearly as nice as Lua or JavaScript

Arduino* Sketch

- C/C++ environment for Microcontrollers
- Extensive suite of libraries
- Your app is written as a “sketch”
- Compiled and copied to MCU's Flash
- USB serial connection for debug messages

```
// the setup function runs once when you press reset or power the board

#define LED 13

void setup() {
    pinMode(LED, OUTPUT);  // initialize digital pin 13 as an output
}

// the loop function runs over and over again forever

void loop() {
    digitalWrite(LED, HIGH);     // turn the LED on (HIGH is the voltage level)
    delay(1000);                 // wait for a second
    digitalWrite(LED, LOW);      // turn the LED off by making the voltage LOW
    delay(1000);                 // wait for a second
}
```

* Named after the Italian king “Arduin” who reined from 1002 – 1014

Agent using C++

- The agent's model declares the door and light as properties
- The server downloads the models for the door and light, and creates proxies for them
- It then calls the agent's initialisation code
- The dictionary of names to symbols is then discarded
- The sketch uses global variables to keep track of things and symbols
- Door and Light use similar code along with hardware interrupts and GPIO pins to interface to the sensors and actuators
- Server supports single threading model to avoid complications with code execution in interrupt handlers

```
Thing *agent, *door, *light;
Symbol unlock_sym, on_sym;

void setup() {
    RegisterThing(agent_model, init_agent);
}

void init_agent(Thing *thing, Names *names)
{
    agent = thing;
    door = agent->get_thing(names, F("door"));
    light = agent->get_thing(names, F("light"));
    unlock_sym = names->symbol(F("unlock"));
    on_sym = names->symbol(F("on"));
    agent->observe(names, F("key"), unlock);
}

void unlock(JSON *valid)
{
    if (JSON_BOOLEAN(valid)) {
        door->invoke(unlock_sym);
        light->set_property(on_sym, JSON_TRUE);
    }
}

void loop() {
    DispatchEvents();
}
```

Note: PROGREM and F() macro direct compiler to save strings in program memory

Event Driven Software

- Avoid MCU blocking on waiting for an incoming network packet as this precludes working on other tasks
- Voltage change on MCU pin triggers hardware interrupt which is handled by Interrupt service handler (ISR)
  - ISR pushes event onto event queue and exits
- Main loop calls event dispatcher which retrieves events from the queue and dispatches the corresponding event handlers
- If nothing to do can put MCU into a sleep state until next interrupt
  - Programmable timers for timer interrupts as a basis for timeouts or behavior at programmed intervals

Dependencies across Things

- One thing may depend upon another
  - Agent example which depends on door and light
- The things may be on different servers
- When you're setting up a thing, the things it depends upon may not be available right now even it is on the same server
  - This requires a means to wait for them to become ready
- Cyclic dependencies
  - A depends upon B which depends upon C which depends upon A
- Server's hold messages for things until they are ready
- I've got this working on the NodeJS server

Strong Security

- Needed to secure communications and enable secure software updates, including fixes for security flaws
  - Motivated by numerous reports of security flaws with current IoT devices
  - NIST Systems Security Engineering – An Integrated Approach to Building Trustworthy Resilient Systems (May 2014)
    - Develop more penetration-resistant, trustworthy, and resilient systems that are capable of supporting critical missions and business operations with a level of assurance or confidence that is consistent with the risk tolerance of the organization
- Microcontrollers may lack processing power and memory to implement the desired security algorithms
- One solution is to use a more powerful MCU, e.g. ARM STM32F417IG
  - Hardware acceleration for AES 128, 192, 256, Triple DES, HASH (MD5, SHA-1), and HMAC, crypto quality random numbers, CRC calculation unit and unique ID
- Another, potentially complementary, solution is to use a dedicated crypto chip, e.g. ATECC108 which connects via I2C bus
  - NIST Standard P256, B283, and K283 Elliptic Curve support, SHA-256, unique ID, crypto quality random numbers, tamper proof EEPROM for keys, certificates and other data
- Looking for people to help with implementing strong security for open source web of thing servers

Please Help

We're looking for offers of help – either people who want to contribute to the open source projects, or for resources to support the work.