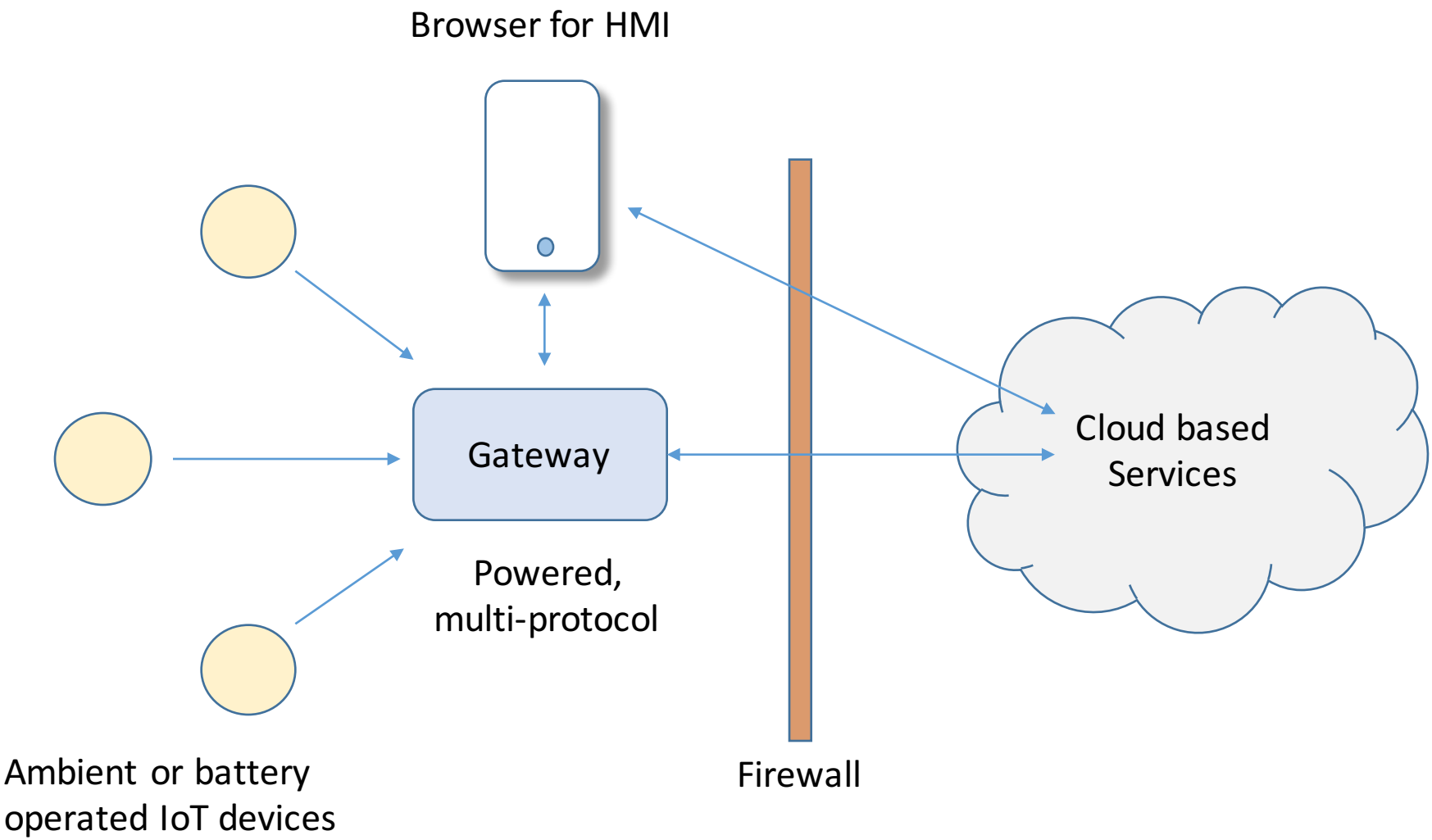
Implementation work on open source web of things servers and gateways

Dave Raggett, W3C
<dsr@w3.org>

Monday, 11 April 2016

Reference Architecture

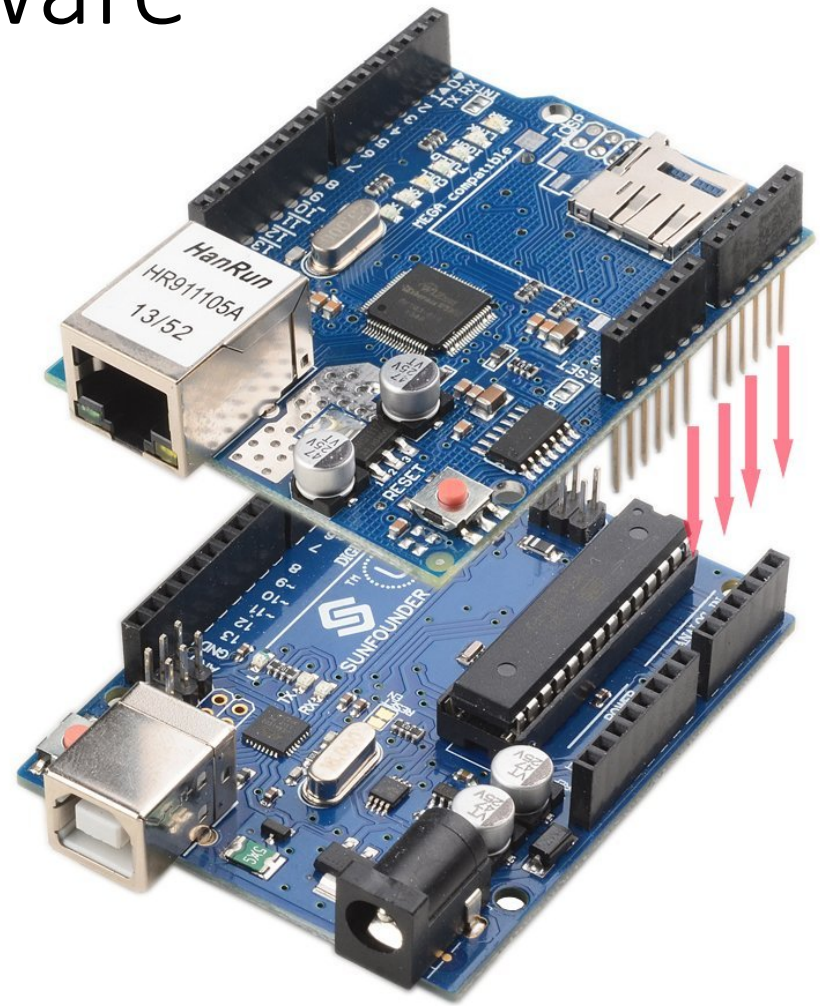


Introduction

- I am working on two open source Web of Things server & gateway projects
  - NodeJS
    - For relatively power devices, e.g. running Linux
  - C++ for microcontrollers
    - For Arduino Uno and Wiznet based Ethernet Shield
      - Highly constrained low cost microcontroller
      - Future plans for extension to more powerful MCUs
      - And other kinds of communications technologies
- Why?
  - Nothing like working code for better understanding
  - Huge potential market for low-end IoT devices

Required Hardware

- Arduino Ethernet Shield
  - 16 KB RAM
  - MicroSD card slot
  - Controlled through SPI bus
  - Polling or H/W interrupt
  - Cost: 4.75 GBP on eBay
- Arduino Uno with ATmega328P MCU
  - 2 KB RAM
  - 1 KB EEPROM
  - 32 KB FLASH
  - Lots of I/O pins
  - Cost: 2.33 GBP on eBay



EEPROM useful for storing settings and error logs, etc.

Required Software

- Arduino IDE
  - Free from https://www.arduino.cc/en/Main/Software
- Serial driver for CH340 USB to serial chip
  - Required for cheap Arduino clones
    - A bit of a pain to find and set up ☹

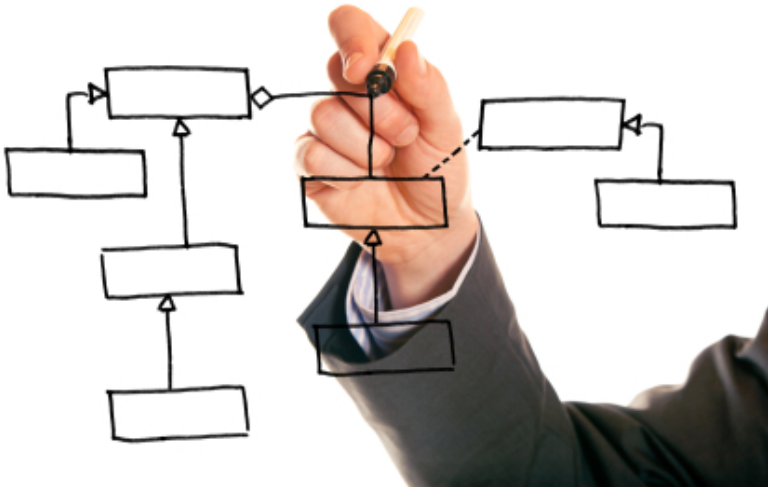
Components

- Arduino Sketch with application code
  - This is where you write your code to interface to sensors and actuators using the Arduino libraries
    - GPIO pins
    - A2D and PWM
    - SPI and I2C buses
    - Timers
- Web of Things Library for the Arduino IDE
  - Installable from GitHub
  - Small footprint with custom networking module
- Wiznet W5100 with native IP support
  - Multicast for discovery
  - UDP and TCP
  - ARP, ICMP, IGMP etc.

Configuration

- Use static IP address or DHCP for dynamic address
- Uses mDNS to discover Web of Things gateway
  - Service type: _wot._tcp.local (unregistered)
  - My MacBook's firewall blocks all other multicast sockets
- Registers remote proxy on gateway for local thing on the Arduino
- Uses TCP for messaging with gateway
  - Less code due to native IP support on W5100 chip
  - Very similar to my Web Socket implementation for NodeJS web of things server
  - CoAP would take more space and be less capable
- Note the choice of protocol depends on the choice of hardware, which in this case is an Ethernet controller

Software architecture



- Target is <28KB for the Library code
  - 2KB for loader, >2KB for application code
- 2KB for RAM
- Avoid malloc and free for robust operation
  - Fragmentation and long term instability
- Static node pool
  - JSON for numbers, booleans, objects, arrays etc.
  - Average Length Binary trees (objects and arrays)
  - Mark/Sweep garbage collector
  - 6 bytes per node on the ATmega328P
- JSON object property names replaced by symbols
  - To reduce RAM footprint to 2KB is very challenging
- Efficient binary message encoding/decoding
  - Relies on deterministic assignment of symbols
  - Designed with short packet technologies in mind e.g. Nordic nRF24L01+ with 32 byte payloads

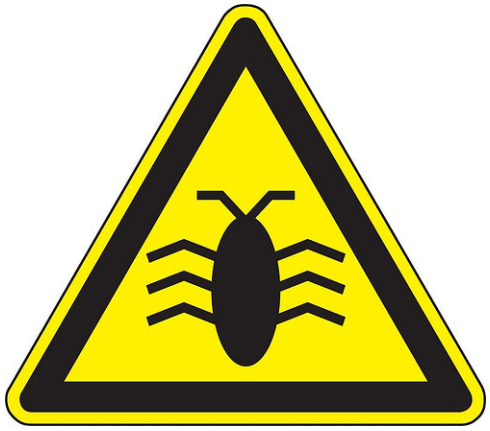
RAM vs FLASH memory

- Some MCUs like the ATmega328P use the Harvard memory architecture
  - This has separate data and code address spaces
- String literals are stored in FLASH and copied to RAM just before entering a function
  - This is obviously bad if you're short of RAM
- First step: tell compiler to keep literals in FLASH
  - F("string literal") vs "string literal"
  - You then need to use system calls to access them
- Second step: use abstraction layer to hide the difference between RAM and FLASH
  - Signatures: const __FlashStringHelper * vs const char *
  - Merge address spaces through adding 0x8000 to FLASH pointers and use string functions that know how to deal with them

Interrupts and Events

- Hardware interrupts are valuable for handling sensors and actuators etc.
- But the interrupt service routine (ISR) must be quick to avoid blocking other interrupts
- Event driven behaviour is very convenient, but event handlers can take time to execute
- Solution: ISR pushes event onto queue, which is dispatched from within Arduino loop() method
- Same single threading execution model as for Web browsers – event handlers are executed sequentially
- Avoids app developers from having to deal with complex synchronisation issues

Life after Death



- Eventually something bad *will* happen …
- Memory issues
  - Running out of RAM for the program stack
  - Running out of free nodes for use with JSON
  - Overflow of critical queues
- Program bugs
  - You can never be 100% confident that there are no bugs
- Bad data
  - Violation of design assumptions
- Software restart on detection of fatal errors
- Hardware watchdog timer forces restart if not reset in time
  - This allows the device to recover after being stuck in a loop
- Gateways and other devices must be resilient to such restarts

Things and Proxy chains

- Things are software objects with properties, actions, events and metadata
- If you want to interact with a Thing on another device you will need a Proxy object for it
- Things Layer keep Proxies in sync with their Things
- Proxies can be chained to form Proxy **trees** rooted in the Thing they stand in for
- Updating a property on a proxy results in messages that travel towards the root and leaves of the tree

Locally Scoped Identifiers

- Each device in the proxy tree generates locally scoped IDs for the proxies/things it hosts
- These are used to match messages to the objects they are targeted at
- Messages are distinguished by whether they are sent to a parent or child of a node in the proxy tree
- Proxy objects hold their ID for this device as well as their parent's ID for the corresponding Thing/Proxy on the parent device
- Message handler sorts this all out to route messages between the desired software objects

Clean Separation of Layers

Application Developer
(WoT focus)

Platform Developer
(IoT focus)

Application	Scripts that define thing behaviour in terms of their properties, actions and events, using APIs for control of sensor and actuator hardware Focus on data types, APIs and error handling
Things	Software objects that hold their state Abstract thing to thing messages Semantics and Metadata, Data models and Data
Transfer	Bindings of abstract messages to mechanisms provided by each protocol, including choice of communication pattern, e.g. pull, push, pub-sub, peer to peer, etc.
Transport	REST based protocols, e.g. HTTP, CoAP Pub-Sub protocols, e.g. MQTT, XMPP Others, including non IP transports, e.g. Bluetooth
Network	Underlying communication technology with support for exchange of simple messages (packets) Many technologies designed for different requirements

Implementing the Communications Stack

- Design the software architecture to allow for use of multiple communications technologies
- Thing Layer that hooks into the software objects for things and handles corresponding abstract messages
- Transfer layer maps these to communication patterns subject to communications metadata
- Transport Layer for specific protocols implemented as pluggable modules for the transfer layer
- The choice of transport is indicated by the connection information for each parent and child
- Low end device likely to support only one protocol

Thing Data Models

- Things may have properties, actions, events and metadata
- Events have a type and a value
- Properties have values
- Actions may pass a value when they are invoked
- Actions may return a sequence of values as responses
  - zero, one, two or more responses as appropriate
- Values have types*
  - null, true, false, integers, floats, strings
  - Arrays and objects (sets of name/value pairs)
  - Values can be compound, e.g. nested arrays and objects
  - Things and Streams as first class types

* Should enumerations be allowed as core types?

Lifecycle Considerations

- Things can have other things as their properties*
- Things referenced as properties can be local or remote
- For local things, the referenced thing may not have been created yet
- For remote things, server needs to set up proxies
  - Proxies take time to set up due to the need to get the Thing data model description
- Solution: push the property to a pending queue and set the property value when the referenced thing/proxy is ready

* Except for very constrained edge devices

Cyclic Dependencies

- With things as properties there is the potential for cyclic dependencies between things
  - Thing A has Thing B as a property
  - Thing B has Thing C as a property
  - Thing C has Thing A as a property
- This could involve a mix of local and remote things
- If we start such Things when all of their properties have been started then we will get a deadlock!
- Solution: start a Thing/Proxy when all of its properties are resolved, and for each Thing/Proxy hold messages to that object in a queue until it has been started

Late Binding

- Values whose full type is only given at run-time
  - The type may be partially specified at compile time
- Can apply to properties, actions and events
- Variant data types with tag that determines which variant applies to this value (e.g. C++ unions)
- Things as late bound values
  - You need to retrieve the data model to instantiate the software object for the thing you've been passed
  - The value includes a reference to the data model
- Thing Layer manages this automatically

Pointers

- Say a Thing has a property which is an array
- A script updates the 3rd item in an array
- This requires a means for update messages to identify a reference to the 3rd item in the array
- In other words some kind of pointer that operates on the Thing's data model
- Simple expression within JSON messages, e.g.
  "path":foo[3] and "path":foo[3].bar
- For CoAP a URI based syntax would be appropriate

Refinement of JSON-LD

- Used for expressing Thing data models
- @context for binding short names to RDF URIs
  - Implicit context for the most common names
  - Linked context for domain specific names
    - Used for semantic search and service compositions
    - Can be safely ignored by edge devices
- Top level object with “properties”, “actions” & “events”
- Values either declared as *name: type*
  - e.g. “on”:“boolean”, or “level”:“float”
- Or declared as *name: { annotations }*
  - e.g. “temperature” : { “type” : “integer”, “min” : -20, “max” : 100, “units” : “celcius”}
  - Needed for nested properties*
- Actions declared with “in” and “out” for the values that are passed to them, and for the responses the action generates

* Discussion topics: how to apply this to nested arrays and objects

Integrity Constraints*

- Enable detection of bad data for increased resilience in the presence of faults and attacks
- Good programming practice: assert(*expression*)
- Basic integrity constraints on a single value
  - Upper and lower bounds for numbers
  - Closed set of values for strings
- Integrity constraints across multiple values
  - Relationships between properties of a single Thing
  - Relationships across multiple Things
- Cardinality constraints
  - e.g. For every X you must have at least two Y
- Metadata constraints
  - e.g. for application versioning
- Need for an expression language for defining constraints

* Temporal constraints can be expressed in domain models

Integrity Constraints in JSON-LD

- Simple value constraints
  - As annotations on the type, e.g. min and max
- Complex constraints involving expressions*
  - Operators and Functions
    - Numeric comparisons, String comparisons, Boolean operators
    - Extensible set of functions
- Complex constraints involving pointers
  - Pointers as references to other values in this Thing
    - Other properties, which can include other things
    - Other parts of the data in an event, or passed to, or returned by an action
- Too expensive for low end devices with limited RAM
  - I plan to implement this on the gateway server

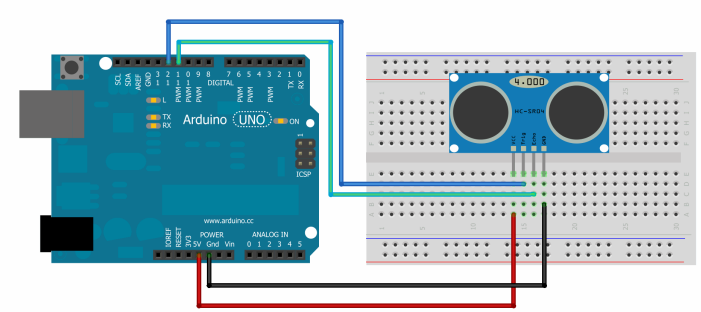
* Can we generalise JSON to allow easy to understand expressions with operator precedence?

Domain Models

- Rich ontology based descriptions
- Can express semantic constraints
  - If x is a temperature sensor then x must define its physical units from the set {Kelvin, Celsius, Fahrenheit}
  - Temporal constraints, e.g. on sequences of actions
- Useful for semantic search and service compositions
  - Verify compatibility of services
- Requires support for OWL, SPARQL, etc.
- Not needed for resource constrained edge devices

Demo Projects

- Aim to provide some examples for how to use the library with commonly available sensors and actuators, e.g.
  - Temperature & humidity
  - Ultrasonic range sensor
  - Multicolour LEDs
  - Servo motor



- Need to provide sketch, wiring diagram and notes
- Better yet to also provide a video
- Looking for volunteers to help with this effort!
  - Starting in June 2016

Project site

- The Arduino project is on GitHub
  - https://github.com/w3c/wot-arduino
- Still a work in progress
  - Having to do this as a background activity due to lots of competing tasks ☹
  - Currently working on Thing Layer following recent work on garbage collection and discovery
  - Re-using virtually all the code for gateway
    - Will enable low cost microcontroller based gateways
  - Plan to update NodeJS project* to match
- Goal to have fully documented project with examples in time for Beijing meeting in July 2016

* https://github.com/w3c/web-of-things-framework

Where Next?

- More powerful processors
  - Arduino Mega
    - 8KB RAM, 4KB EEPROM, 256KB FLASH
  - ARM based MCUs
    - More RAM, but no EEPROM
- Work on Streams, e.g. for electrocardiograms
- Other communication technologies
  - Sensor networks with Nordic nRF24L01+
  - Bluetooth Smart with XBee Arduino shield
- Sleepy devices
  - Using timer to wake up every few minutes
  - Mainly relevant to wireless technologies
    - Note that Arduino boards are power hungry due to LEDs and linear voltage regulators
- Synchronisation across clusters of devices
  - Synchronised control of robot joints
- Strong hardware based security
- Domain models
- Crowd sourced informal semantics