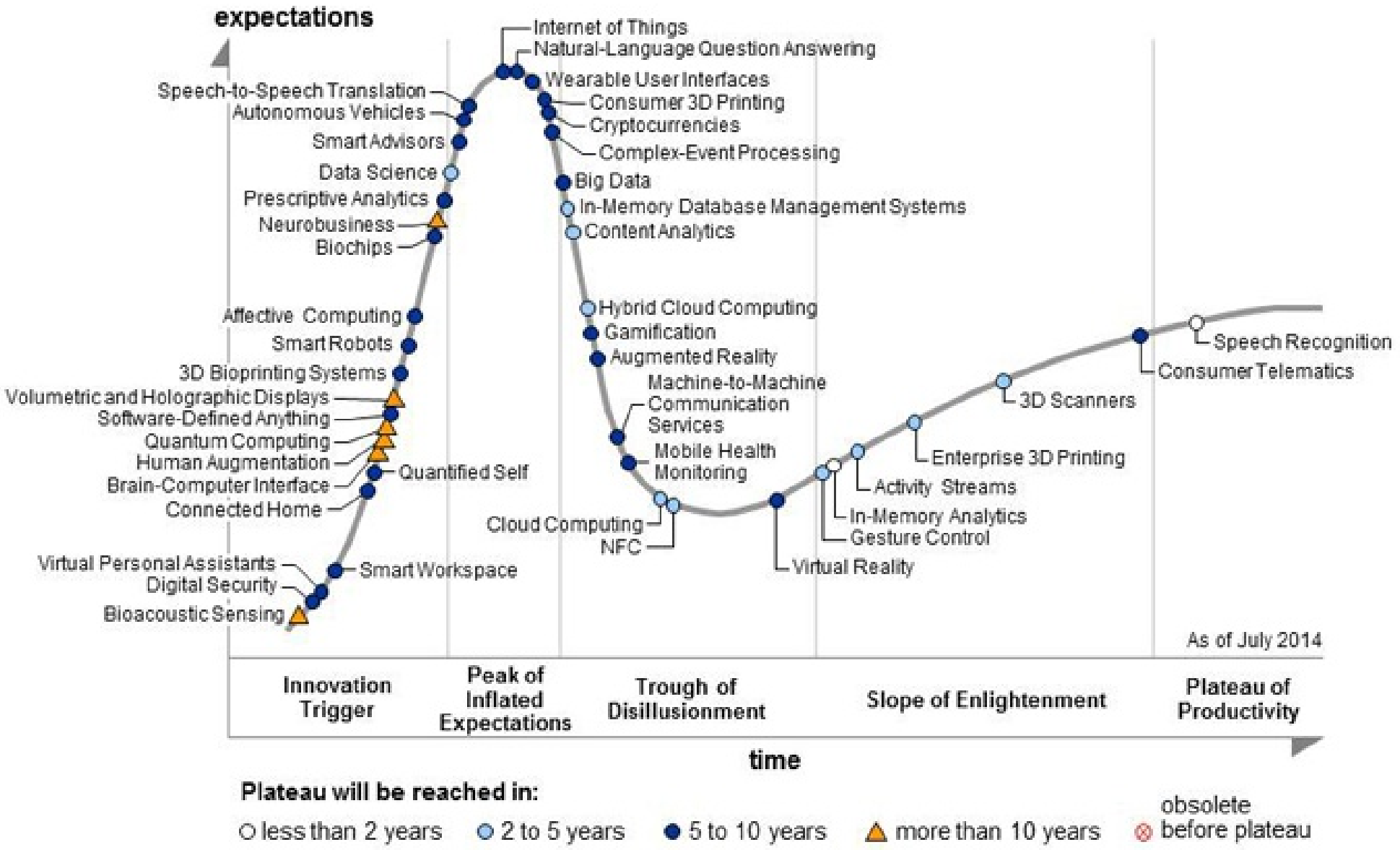
The Web of Things

Dave Raggett <dsr@w3.org>

Presentation to OIC

Thursday, 5 Nov 2015

IoT still at the top of the hype cycle*



*From Gartner's hype cycle for emerging technologies – August 2014

IoT – over-hyped and fragmented

- The IoT is still very immature, but has huge potential
- There is currently a lot of fragmentation, which
  - drives up development costs
  - increases the risk for investors
  - reduces the market size for solutions
- There are many Industry Alliances and SDO's
  - W3C is seeking to establish collaboration agreements and work together to realise the potential and unlock the network effect

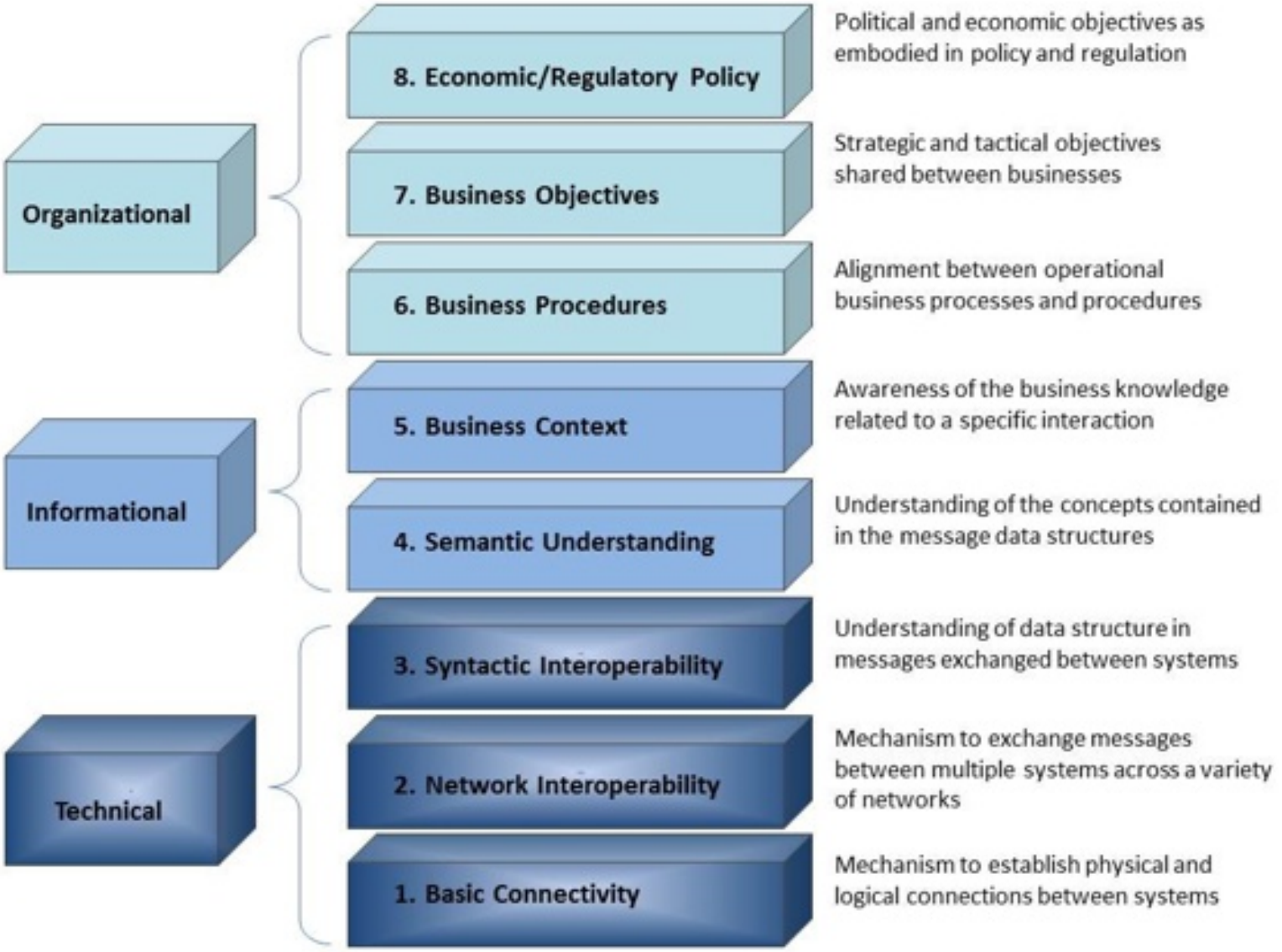
The World Wide Web Consortium

- W3C is a member funded international organisation focusing on developing standards for Web & semantic technologies
  - e.g. HTML, scripting APIs, the Semantic Web and Linked Data
- We are addressing the fragmentation for the IoT with an abstraction layer that spans devices, platforms and application domains
- A focus on simplifying application development by decoupling services from the underlying protocols

Interoperability implies . . .

- Exchange of meaningful, actionable information between two or more systems across organizational boundaries
- A shared understanding of the exchanged information
- An agreed expectation for the response to the information exchange A requisite quality of service: reliability, fidelity, and security.
- From: **GridWise Architecture Council**, March 2008 “Interoperability Context-Setting Framework”
  - See: http://www.gridwiseac.org/pdfs/

Interoperability Framework



Challenges Related to Interoperability

- Scaling on multiple dimensions
  - Scaling across **devices** from microcontrollers to massive cloud based server farms
  - Scaling across **platform and services** from different vendors and built upon different standards
  - Scaling across application **domains**
- The inevitability of evolutionary change in complex ecosystems
  - Weakly coupled communities will evolve independently
  - How to support “trade” across these communities
- Discovery of services
  - The benefits of a *lingua franca*, and its limitations
- Composition of services
  - From different vendors for an open market of services

Why do we care?

- The potential value for IoT applications grows rapidly with the ability to flexibly combine services
- There are huge potential benefits across many application domains
- For example, a car navigation system could interact with the local city information system to locate the best parking option, given the current destination
- Even better, if the city offers parking reservation services then the navigation system could make the reservation in advance, with payment to be made by a smart phone transaction once the car was parked

Semantics & Metadata

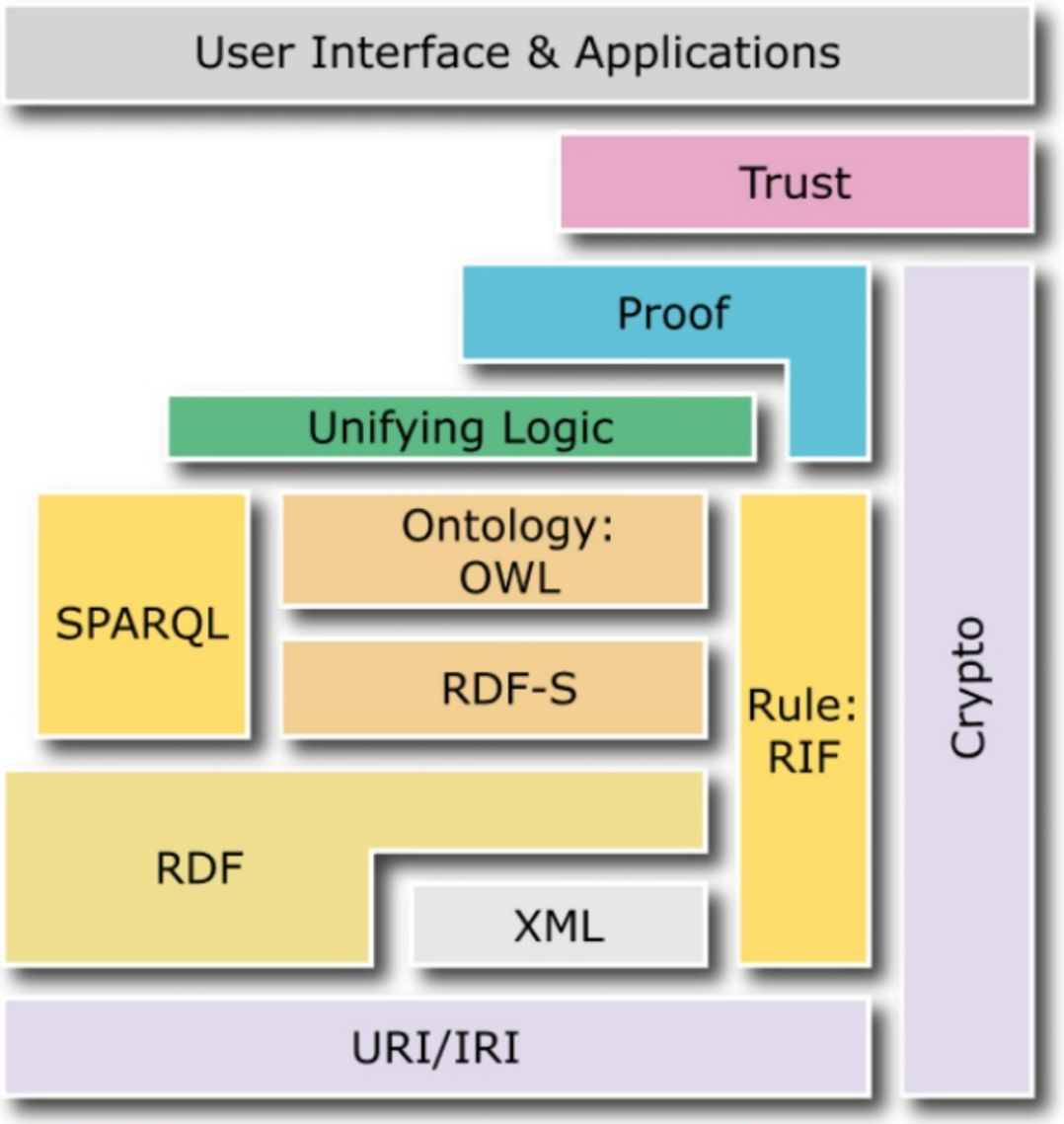
Semantics – a very quick recap

- Semantics is the study of meaning
  - The relationship between words and what they stand for
- Some things we can talk about
  - **People**, e.g. Robert Schuman
  - **Places**, e.g. Brussels
  - **Events**, e.g. the formation of the European Coal and Steel Community (ECSC)
  - **Dates**, e.g. 9 May 1950
  - **Documents**, e.g. the Schuman Declaration
- In other words, a mix of physical and abstract entities

Relationships Between Entities

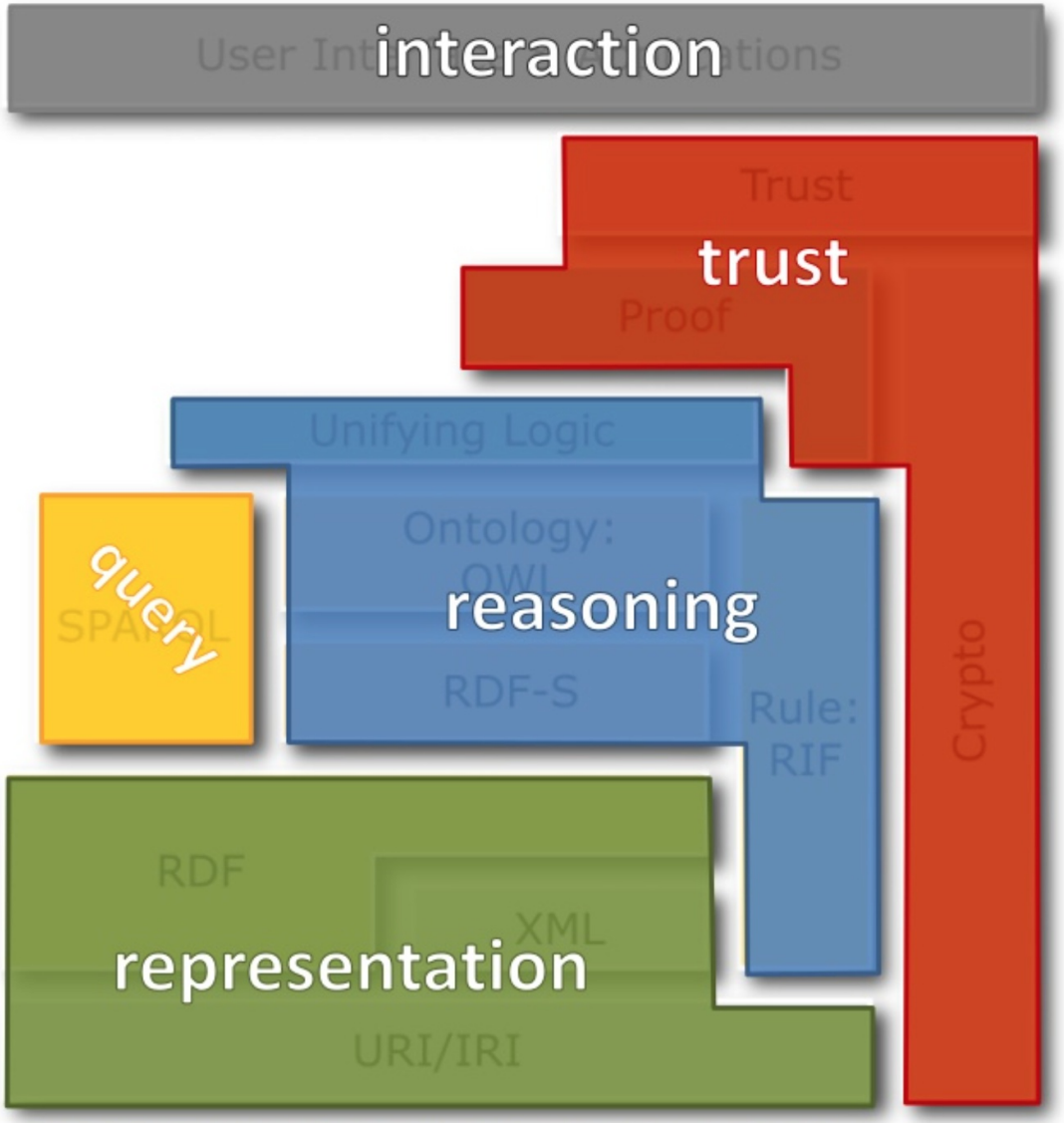
- Named relationships between entities
  - **Brussels** is the **capital city** of **Belgium**
  - Subject: “Brussels”
  - Predicate “capital city”
  - Object: “Belgium”
- W3C's **Resource Description Framework**
  - Subject, Predicate and Object as Web addresses (URLs)
  - These URLs act as globally unique identifiers
  - The URLs can be dereferenced to obtain further information (hence the term “Linked Data”)
  - RDF has many serialisations, e.g. XML, Turtle, JSON-LD

Semantic Web Stack

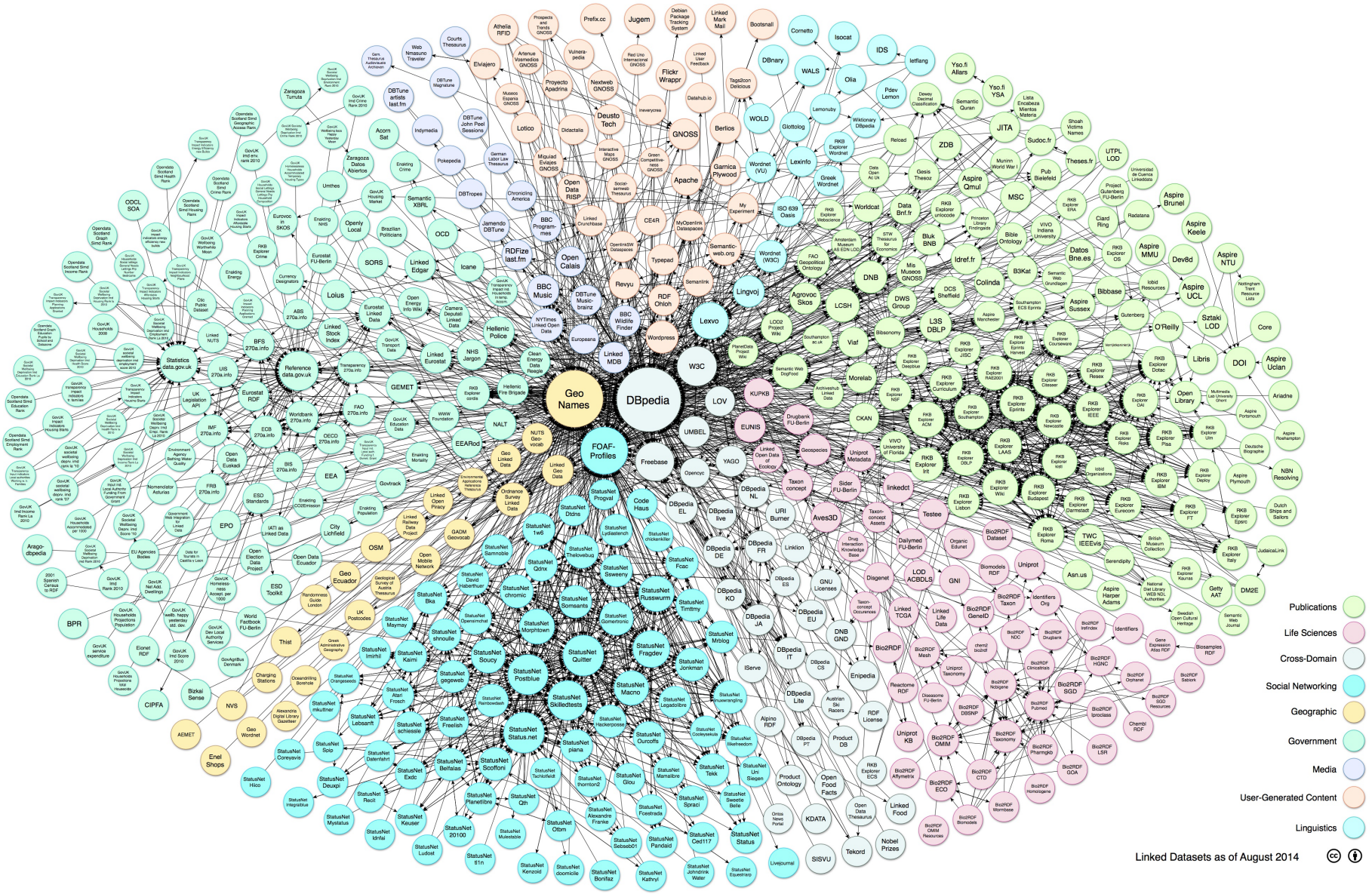


With thanks to Fabien Gandon

Semantic Web Stack



A Growing Cloud of Linked Data



The Value of Semantics

- **Interoperability**: Semantic approaches enable dynamic and easier integration and cross- domain interoperability on multiple scales and levels, bypassing the current practices of pre-deployment determination and configuration.
- **Discovery**: Semantic approaches enable dynamic discovery of IoT services, data and resources. This supports application composition and service reuse.
- **Consistency**: Through semantic approaches, services, data, and resources can consistently refer to the same meaning within and across IoT domains.
- **Scalability**: Semantic approaches (such as semantic annotation) provide support for component independence and decentralized management and act as scalability enablers.
- **Efficiency**: Semantic technologies enable the efficient description of data, resources and services, so that machines and humans can have a common understanding of resources, data and services in the IoT. This benefits automatic operations, analysis and processing activities and enhances human-machine interaction

Semantics and the IoT

- What is the relevance to the Internet of Things?
  - Shared vocabularies for entities and their relationships
  - Describing the software objects that stand for “things”
  - Verifying that a data source and sink are compatible and have the same semantics
    - Floating point number representing a temperature value expressed in Kelvins
  - When searching for services with a given semantics
    - Show me all temperature sensors in a radius of 100m
  - To facilitate the design of service compositions
  - To enable simulation prior to deploying changes to cyber-physical systems

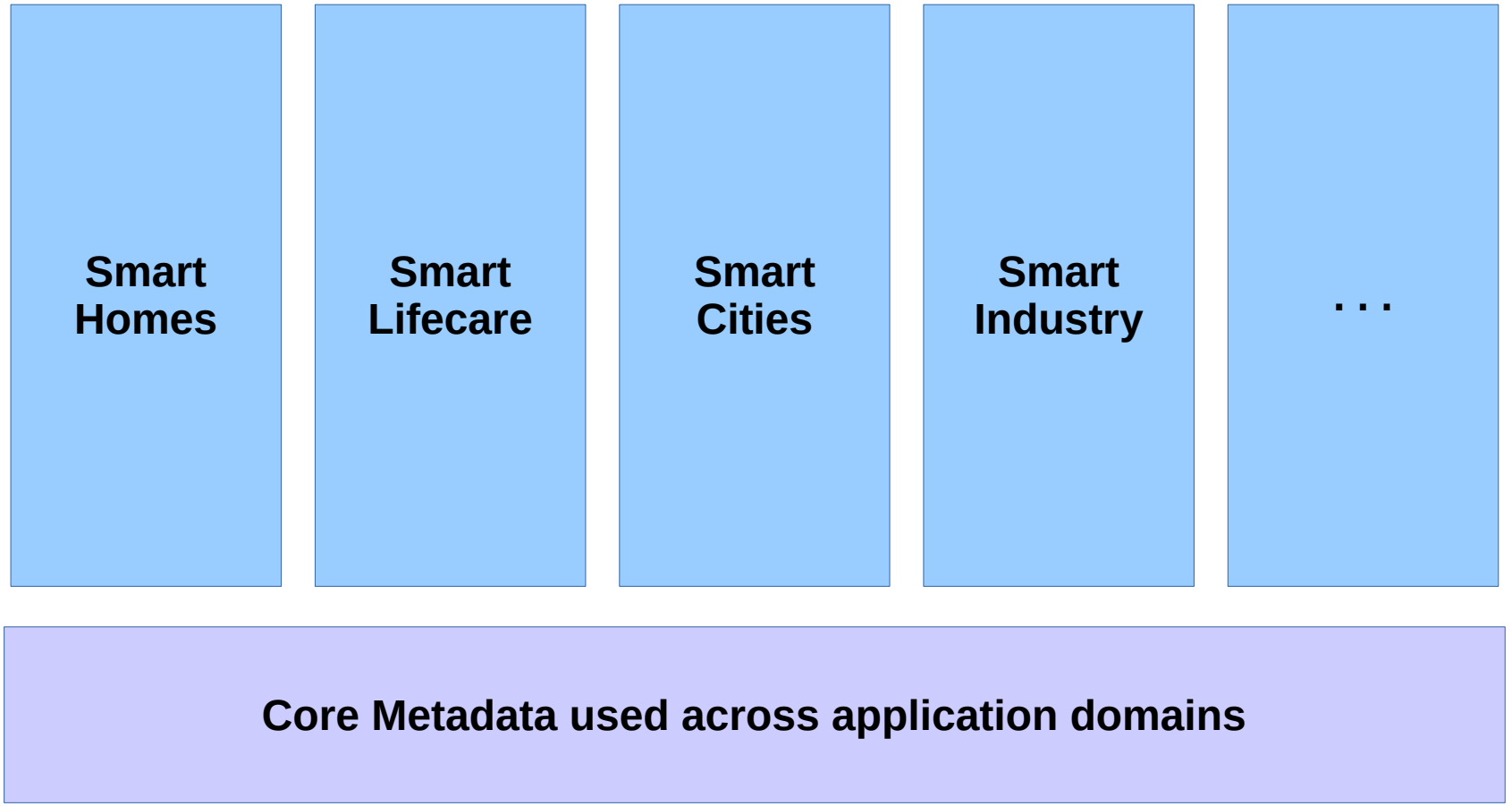
Uses of Semantics and Metadata

- There are many opportunities for exploiting semantics and metadata
  - Discovery
  - Service composition
  - Planning and simulation
  - S/W & H/W Dependencies and interoperability
  - Provisioning and set up
  - Monitoring and detection of problems
  - Causal models for fault diagnosis
  - Security, safety, compliance and resilience
  - Control at multiple levels of abstraction
  - *etc.*

From the Web of Pages to the Web of Things

- The fundamental elements of Web architecture
  - Addresses, Resources, Protocols
- For the Web of Pages
  - URLs, HTML and HTTP
- For the Web of Things
  - URIs, Thing descriptions, a suite of protocols
    - Different protocols are suited to different contexts
- Declarative formats for resources allow search engines to spider the Web and create indexes
  - HTML – Hypertext Links
  - Thing descriptions – Links between dependent “things”

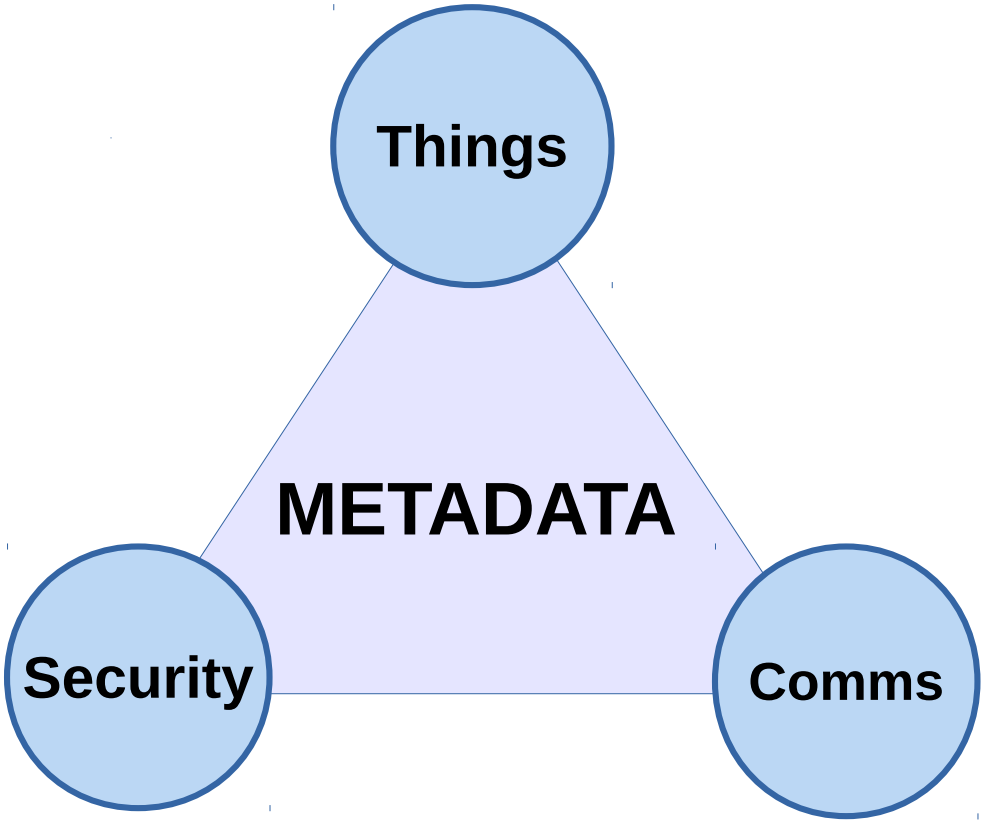
Horizontal and Vertical Metadata



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

W3C view of Horizontal Metadata

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
  - Safety, Compliance and 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

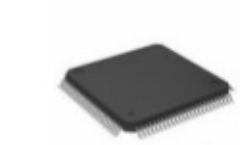
The Web of Things

Web of Things

- Making life easier for application developers by providing a simple scripting model
  - Things standing for physical and abstract entities
  - Things as software objects with **properties**, **actions** and **events**
  - Applications scripts **decoupled** from underlying protocols which can be selected according to need
  - Based upon **rich metadata**
- Server creates a software object based upon the thing's description
  - What properties, actions and events does it have?

Web Servers at Many Scales

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



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

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

Servers are free to choose which scripting languages they support
Precompile service behaviour for constrained devices

The Web as a system of agents

- The Web of Things is essentially a system of agents as defined by Carl Hewitt in 1973
- A set of things hosted on different servers
- Application scripts define the agent behaviour
- These servers exchange messages over a variety of protocols
- Messages take a variable amount of time to transfer that depends on the protocol, network, and devices
- Application scripts only have access to the state of the instances of the things or proxies for things on that server
  - Thing proxies cache the state of things on behalf of the underlying protocols
- Even if a server asks another server for the state for a thing, by the time it gets a result, it may be out of date
- Even time is uncertain due to the limitations of synchronising clocks across servers
  - Moreover, many IoT devices lack real-time clocks

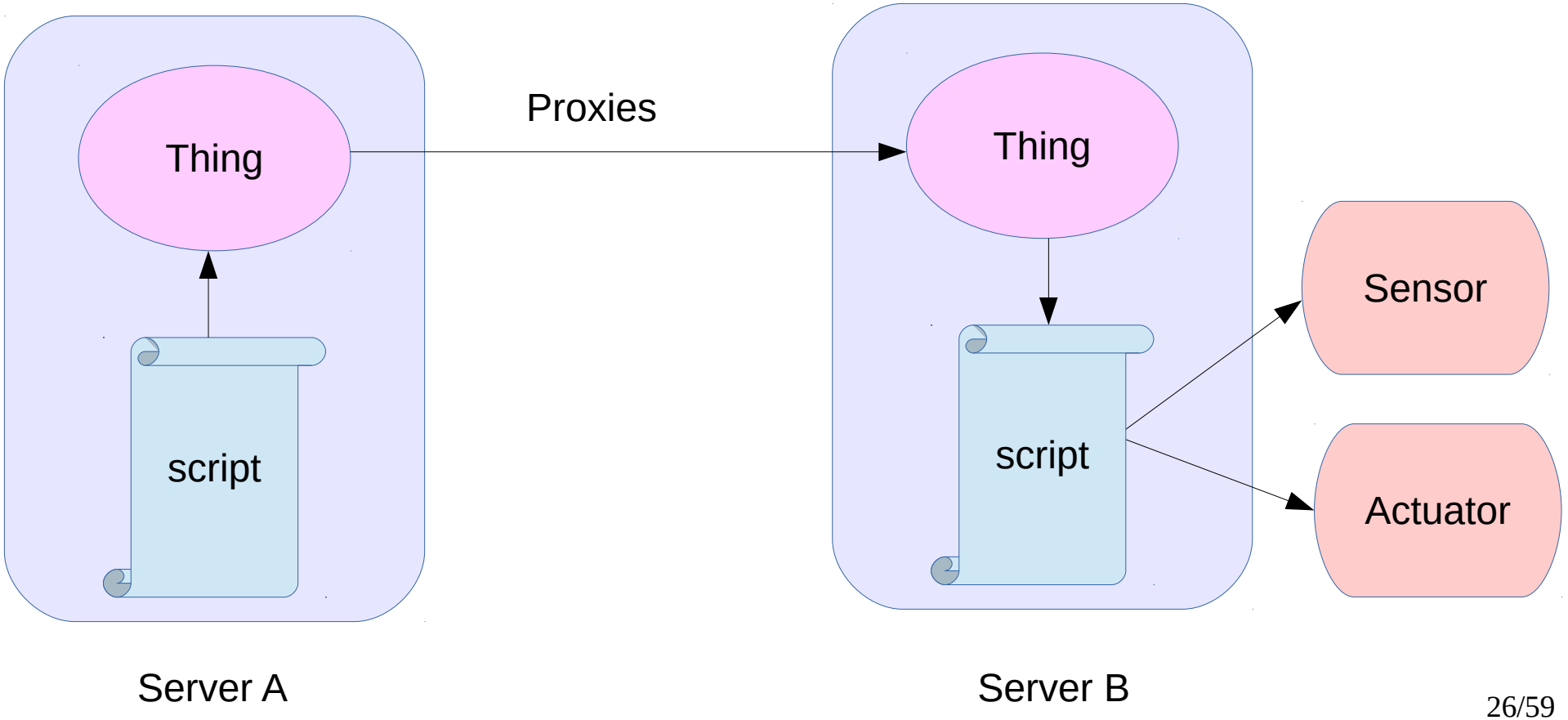
Many Protocols

- Internet Protocols
  - HTTP*
  - Web Sockets
  - CoAP*
  - MQTT
  - XMPP
  - AMQP
- IoT protocols
  - CoAP over 6LoPAN
    - IPv6 over 802.15.4
    - IPv4 & IPv6 over WiFi
  - MQTT-SN
  - Bluetooth Smart (BLE)
  - ZigBee
  - KNX
  - EchoNet
  - ETSI LTN, Weightless, LoRaWAN, SIGFOX UNB, ...
  - *and a great many more*

* Commonly used with REST

Distributed Web of Things

- Virtual representations of physical or abstract entities for use by application scripts
  - Each thing has a URI for a description that can be used to create a proxy for that thing, allowing scripts to interact with a local proxy on behalf of a remote entity
  - Scripting things in terms of their metadata, properties, actions and events
  - Web page scripts in browser can create proxies for things on servers



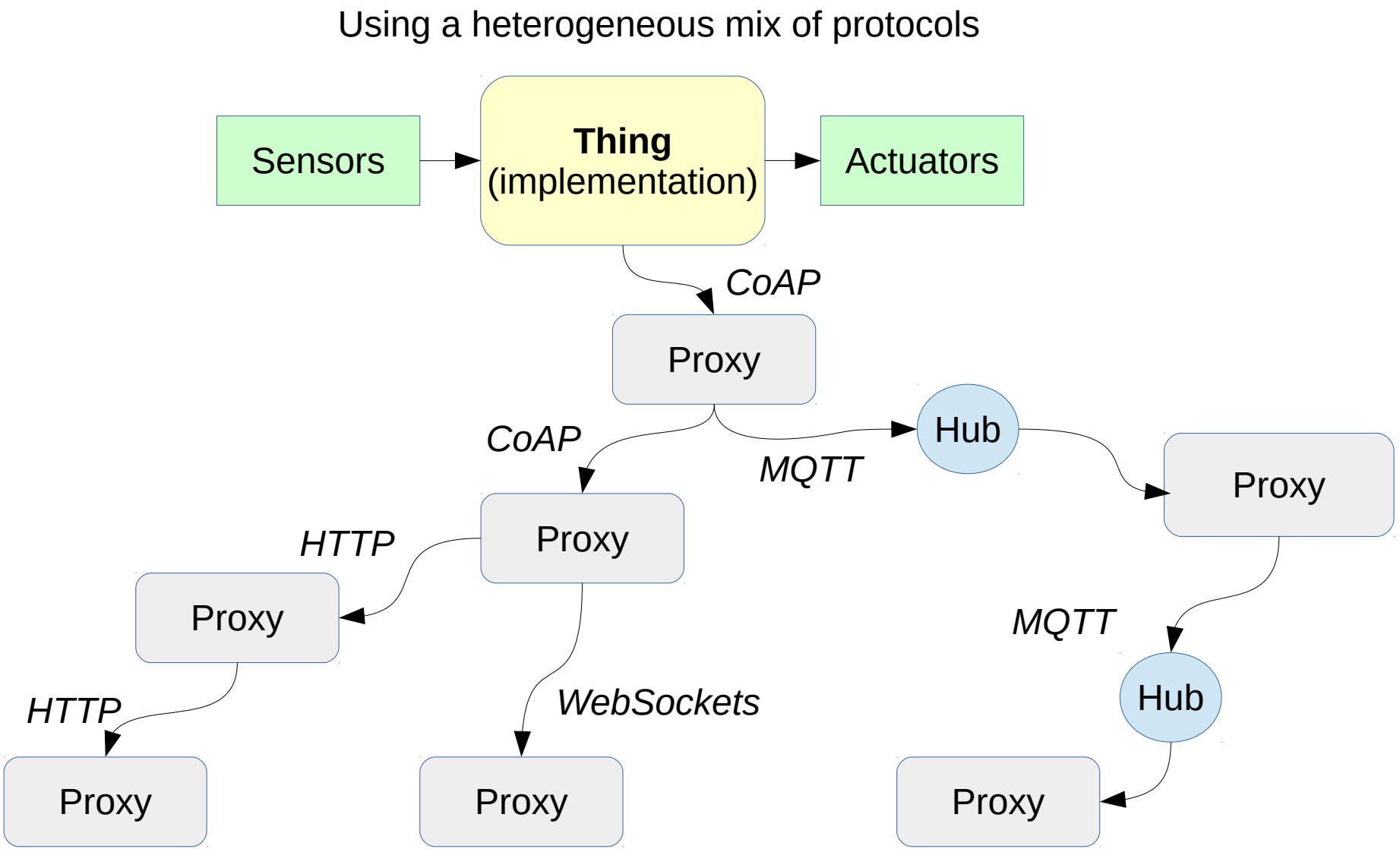
Web of Things Topologies

- The Web of Things lends itself to different topologies
  - **Peer to Peer**
    - Devices talk directly to one another
    - Each device can host a mix of things and proxies for things on other devices
  - **Peer to Peer via Cloud**
    - Using a message routing network
    - Using WebRTC data channel
  - **Star** – Hub as controller for cluster of devices
    - The hub has proxies for the things on each of the devices
  - **Device to Cloud**
    - Device registers things on Cloud-based server
  - **Star to Cloud**
    - Hub acts as gateway between devices and the cloud

Server Metadata

- When server A configures a proxy for a thing on server B it needs to discover which protocols that server B supports
  - Protocols
  - Data formats
  - Ports
  - Security
- Server A can set up object properties via setters and getters that hook into the transfer layer
  - Mapping abstract messages to concrete protocols

Metadata as basis for decoupling services from protocols



Some Requirements

- Create a thing from its metadata and an implementation
- Destroy a thing and all of its proxies
- Register a proxy for a thing
- Unregister a proxy for a thing
- Access to metadata for things and servers
- Notify events and updates to properties and metadata
- Invoke actions and asynchronously return their results
- These can be considered as abstract messages and mapped to communication patterns over specific protocols
- Allow for cyclic dependencies between things

Layers

Application	Scripts that define thing behaviour in terms of their properties, actions and events, using APIs for control of sensor and actuator hardware
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, ...
Transport	REST based protocols, e.g. HTTP, CoAP Pub-Sub protocols, e.g. MQTT, XMPP Others, potentially including non IP transports
Network	Underlying communication technology with support for exchange of simple messages (packets)

Dependencies across Things

- One thing may depend upon another
  - Agent example which depends on door and light
- The dependent 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 hold messages for things until they have started
  - Avoids the need for messages that signal when things are ready
- I've got this working on my NodeJS server

Communication Patterns

- The properties for a given thing can be updated by the application script on the server hosting the thing, and by applications scripts on servers hosting proxies for that thing
- The proxies form a tree rooted in a thing
- Updates can be pushed from the thing to its proxies
- Updates can be pushed from a proxy to the thing, and from there to the other proxies
- Push can be related to pub-sub and message routing
- Another approach is to pull updates via polling

Simple Message Exchange

- Reliable in-sequence delivery of messages
- Asynchronous messaging across a communications channel
- Extensible message format, e.g. JSON or XML
- Efficient encodings for messages
- Can be layered on top of transport protocols
  - Web Sockets over TCP
  - Reliable messaging layer over UDP
  - Non-IP based communications technologies

Protocol Bindings

- REST family of protocols, e.g. HTTP & CoAP
  - See Ari's document on recommended practices
  - Pull protocol using stability metadata
  - Cache management across GET on complete state (all properties) vs GET on individual properties
- Pub-Sub protocols, e.g. MQTT & XMPP
- Non-IP based technologies
  - e.g. ZigBee, Bluetooth, KNX, EnOcean and communication technologies with very small messages

REST Bindings

- REST = representational state transfer
  - GET, PUT, POST, PATCH, DELETE methods
    - GET – get all of a thing's properties (cacheable)
    - PUT – put all of a thing's properties
    - POST – for actions and events
    - PATCH – update a subset of properties
    - DELETE – delete a proxy?
  - Hierarchical path for addressing resources
  - GET can also be used to get subset of properties
    - But need to handle caching with care to avoid awkward interaction with cache entries for GET on all properties
    - The proxy for a thing as the cache of its properties
- How valuable is REST in this context?

API Patterns

- Different languages and conventions
  - Static vs dynamically typed languages
    - Dynamic languages allow objects to be given properties at run-time
    - Static languages force thing properties to be passed as method arguments
  - Call backs, e.g. for handling events, or results from actions
  - JSON & Objects as arguments
  - Promises as a popular pattern for JavaScript
- Locally generated events to allow applications scripts to listen for changes
  - Changes to properties
  - Changes to metadata
  - Lifetime events, e.g. when a thing is destroyed
  - Distinct from the events declared in the thing's model

Example

- 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
  - With a simple mapping to RDF

Thing Descriptions

Server uses URI for a thing to download its description and create a local proxy object for use by scripts

- Door

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

- Light switch

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

Vocabulary context defines bindings of core vocabulary to URIs for RDF nodes.
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.

CoAP*

- UDP analog of HTTP for constrained devices
  - RFC7252 from the IETF CoRE Working Group
  - HTTP & TCP are too memory hungry!
- Designed for RESTful services
  - Roy Fielding's representational state transfer
    - PUT & GET transfer complete state
- GET, PUT, POST, DELETE and Observe
  - Support for breaking up and reassembling resources that don't fit into a single short packet
  - No support for HTTP's PATCH method
  - Clean HTTP-CoAP mapping for gateways
- Pub-Sub mechanism
  - Interested parties register with GET and observe header
  - Notifications are sent asynchronously with Observe header
    - See draft-ietf-core-observe
- Resource discovery
  - Unicast and multicast queries (RFC7390)
  - Link format (RFC6690) analogous to HTTP Link header
    - With well defined mapping to RDF
  - GET /.well-known/core returns list of resources

*"**CoAP is aimed at tiny resource constrained devices, e.g. IoT system on a chip, where TCP and HTTP are not a good fit**"*

Matthias Kovatsch, ETH Zurich

Security is based on DTLS

- Elliptic Curve Cryptography
- Pre-shared secrets, certs or raw public keys
- IETF currently working on authentication and authorisation (ACE), and
- DTLS profiles (DICE)

IETF Class devices 1 and above
- 10 KB RAM and 100 KB Flash

In use by
- OMA Lightweight M2M
- IPSO Alliance
- ETSI M2M & OneM2M

* *Hands on with CoAP*

MQTT*

pub-sub for the masses

- Pub-sub messaging protocol over TCP/IP
  - Topic based message routing via brokers & gateways
  - MQTT-SN runs over UDP for smaller devices
- Designed for constrained devices
  - Connect, publish, (un)subscribe, disconnect
  - Message body treated as byte array
  - Smallest possible packet size is 2 bytes
- Features
  - 3 quality of service levels
    - 0: at most once delivery
    - 1: at least once delivery
    - 2: exactly once delivery
  - Retained messages (last known good value)
  - Topic wildcards
  - Last will & testament for broker to publish if client goes offline
  - Persistent sessions
  - Heartbeats

OASIS MQTT v3.1.1

- 1st byte contains
- Message type (4 bits)
- DUP flag (1 bit)
- QoS level (2 bits)
- Retain flag (1 bit)
- 2nd byte contains length in bytes
- Top bit set implies length continues in next byte (max of 4 bytes for length)
- Followed by *length* bytes as sequence of length prefixed fields
- Variable header, e.g. client Id, topic name, and packet Id
- Message payload

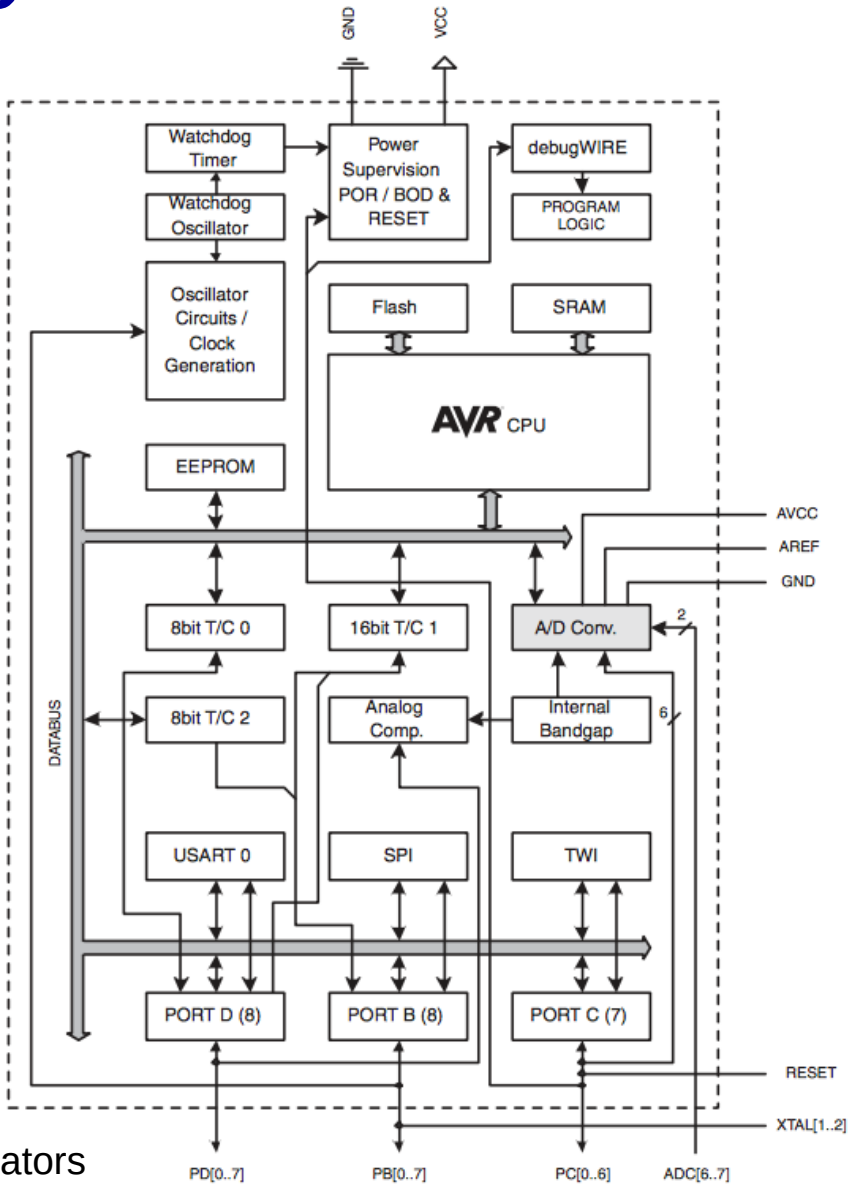
MQTT-SN over UDP

- Multicast socket based discovery of message gateway

* Originally named “message queuing telemetry transport”

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

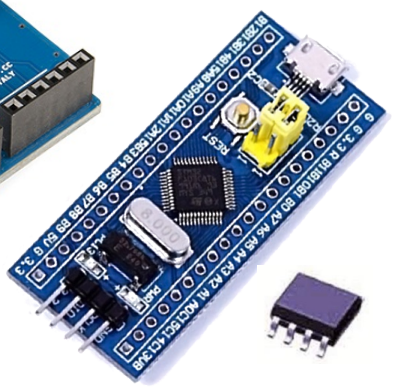


Building Momentum through the Maker Community

- 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



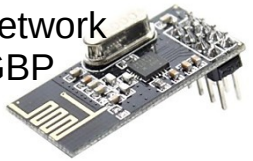
ESP8266 96 KB
RAM, 512KB Flash
32 bit MCU + WiFi
1.5 GBP

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

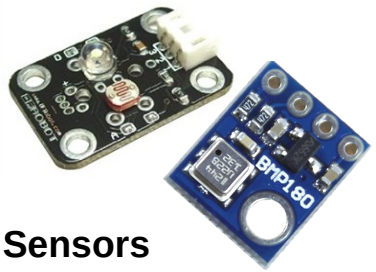
NodeJS based Web of Things server with many libraries available for IoT (run on Raspberry Pi as Home Hub)

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

ATECC108A ECC Crypto
ATAES103A AES Crypto



Sensors

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

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

Authentication

- W3C is seeking to move the Web away from user name and password
  - Increasing emphasis on public key cryptography
  - Learning lessons from experience with PKI
- New Web Authentication WG planned with support from the FIDO Alliance and other groups
    - Multi-factor authentication as appropriate to context
    - Focus on assuring that this is the same device+user as when the user account with the website was originally set up
    - Does ***not*** address binding of Web Identity to Real-World Identity
- W3C hardware based Web Security WG
  - Leveraging secure elements of various kinds including SIMs
    - Secure tamper-proof storage and computation
      - Provisioning opportunities and management of updates

Security & Resilience

- Safety and compliance
  - Ensuring that cyber-physical systems are safe and comply with all applicable legislation and guidelines
- Secure by design
  - Strong authentication
  - Encrypted communications
  - Tamper proof storage of keys and credentials
  - Best practices for provisioning and security updates
- Resilience is about coping with faults, demand spikes and cyber-attacks
  - How to predictably and gracefully adapt to threats
  - System level response
  - Need for monitoring to detect problems
    - Trip-wires
    - Abnormal behaviours
  - Defence in depth with security zones

Provisioning and Updates

- The challenge involved in setting up devices and services
  - Establishing the security context
    - Creating a identity for the device, generating and registering its public key(s)
    - Setting up the trust relationships, e.g. for software updates
    - Threat models for provisioning
- Secure software updates
  - Essential for strong security
    - To fix security bugs, and to update security settings in line with best practices
    - To address evolving software interfaces exposed by services
  - Pushed or Pulled
  - Authenticating updates
    - Using previously provisioned public keys for trusted sources
  - Streaming updates (too large to fit in RAM)
  - Recovering from errors
    - Temporary network problems resulting in incomplete updates of Flash memory
    - Boot loaders need to test for successful updates and enter a safe update mode if previous update failed to complete
    - Implications for the security context

Credentials

- Attestations by trusted 3rd party about the attributes of an identity
  - Needed to tie web identity to real-world identity
  - Applicable to people, IoT devices, services, ...
- Increasingly important for an online world
- Ephemeral vs Long Lived credentials
  - Reduced risks through short lived credentials issued against a session ID
    - Potential role for secure elements
- W3C is collecting use cases and requirements with a view to a Credentials Working Group

Privacy and Contracts

- The IoT makes attention to security and privacy particularly important given the amount of personal or confidential information that can be collected by sensors
- Privacy laws vary considerably across jurisdictions
- Contract law by contrast is much more uniform
- Terms & conditions as basis for binding agreement between service providers and service consumers
  - Including the liability taken on by the service provider
  - Enables dynamic markets of services at “Internet speed”
- Used in conjunction with access control
  - Dependency on identity management and authentication
- Precedent of Creative Commons 3 level agreements
  - Icons
  - Human readable
  - Legal details for lawyers

Simplifying Discovery

- Many different ways to discover things
  - mDNS, UPnP and other local area network techniques
  - Bluetooth (beacons), ZigBee
  - NFC, barcodes, IR and audio chirps
  - By following dependencies in Thing descriptions
  - Devices can register themselves in hubs or cloud
  - Social relationships between people and things
    - Personal and organisational zones
    - Spatial (geographic) zones, temporal zones
      - Events and processes as abstract entities
- Simplify discovery via agent API
  - Context based discovery queries
    - Aided by semantic descriptions
  - Agents can collaborate but should respect privacy

Thingsonomies

- The purpose of a “thing” can be defined ***formally*** in respect to an *ontology*
- The purpose can be defined ***informally*** using *free text*, e.g. as one or more tags chosen by the maintainer
- Co-occurrence of tags across many “things” performs an informal expression of semantics
  - In same way as folksonomies for images or blog posts
- Statistical treatment of natural language and cognitive models make this increasingly attractive, e.g.
  - Apple Siri
  - Google Now
  - IBM Watson

Network Efficiency

It is all in the metadata!

- Smart meters vs Security Cameras 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
- 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
- The need to collect representative use cases

Varying kinds of data

- Different kinds of sensors and actuators have very different kinds of data requirements
  - This needs to be reflected in the metadata
- Simple sensors where you just need the latest value, e.g. a temperature and humidity sensor
- Sensor streams where you need a log of readings over time
  - Ability to query data for a specific time or time window
  - Composite data values for each reading
  - Interpolation between readings for smoothly varying properties
    - Programming path of robot hand via smooth control of its joints
- Real-time sensor streams
  - ECG as example of healthcare sensor stream
  - Remotely controllable Security Camera
    - Higher bandwidth and need for low latency
    - Role of events to draw attention to a given sensor

W3C Web of Things Activity

Web of Things at W3C

W3C Web of Things Interest Group: http://www.w3.org/WoT/IG/

- W3C Workshop in Berlin in June 2014
- Launch of Web of Things Interest Group in 2015
  - Task forces
    - Thing descriptions
    - APIs and protocols
    - Discovery and provisioning
    - Security, privacy and resilience
    - Communications
  - Emphasis of implementation experience
    - Demos and plug-fests
  - Face to face meetings
    - Past: Munich, Sunnyvale, Sapporo
    - Joint meetings with IRTF Thing to Thing Research Group
    - Future: Jan '16 Nice, France, April '16 Cambridge MA, July '16 Asia, September '16 Lisbon, Portugal

Plans for Launching Web of Things Working Group

- The Interest Group is working on
  - Use cases, requirements, technology landscape and plans for launching working groups
  - W3C Interest Groups prepare the ground for standards but don't develop standards
  - W3C Working Groups are chartered to develop standards (W3C Recommendations)
- We're collecting ideas including
  - Metadata vocabularies for thing descriptions and servers
  - Serialisations of metadata, e.g. as JSON-LD
  - APIs and bindings to specific protocols
- We expect to launch the WoT WG in 2016

How can we work together?