Design Patterns for an Internet Of Things

A Design Pattern Framework for IoT Architecture

Michael J Koster
May 26th, 2014

Design Patterns are reusable solutions to common problems

Design Patterns provide well known ways to solve design problems commonly encountered in a particular discipline or problem domain. As an example, three different design patterns to handle traffic flow at a road intersection are stop signs, traffic lights, and roundabout. Each have advantages and disadvantages specific to particular traffic patterns and other contextual factors.

IoT presents design problems in many areas and at many levels in the system. There are many diverse use cases, with different resource constraints, and with many standards, products, and technologies available. How do we determine which are appropriate, what the specific advantages and disadvantages are without a context from which to evaluate their use?

Design Patterns are building blocks of architecture

Design patterns provide a way to build an end to end solution in well specified ways and to provide an understanding of the use of different components of the system in a system context. A reference architecture can be constructed from a set of design patterns, and from this the behavior of the system may be modeled and understood.

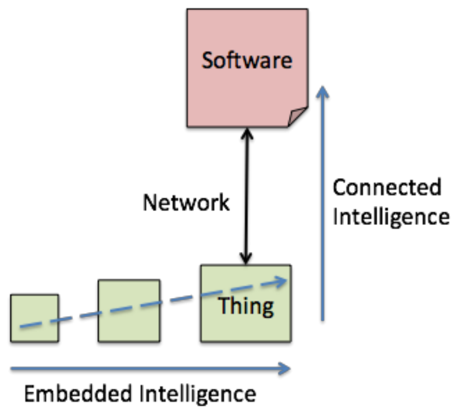
The articles published here up to this point have focused on a set of design patterns around data models and information models for the Internet of Things, and most recently began investigating an open stack approach, where use-case appropriate end to end solutions can be built from various components and design patterns.

Next I would like to look at a set of system design patterns useful in the construction of IoT architecture solutions, with a focus on common patterns for interoperability. I have divided the design patterns somewhat arbitrarily into areas of information models, interaction models, application programming models, infrastructure models, and use case patterns.

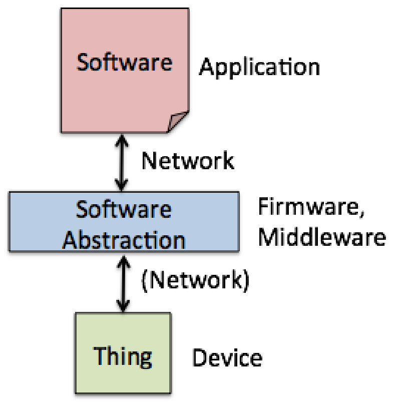
The design pattern examples were drawn from www and internet, best practices and standards, those which appear to be successful and well known, and those which satisfy some need or solve some problem. These are only examples and not meant to be exhaustive. However, the intention is to enable covering most common IoT use cases using a set of design patterns which can use well known standards and practices.

Each of these areas is further decomposed into a spectrum of higher level patterns built on lower level patterns. I present only a few examples of common design patterns; this is not meant to be an exhaustive treatment of the subject.

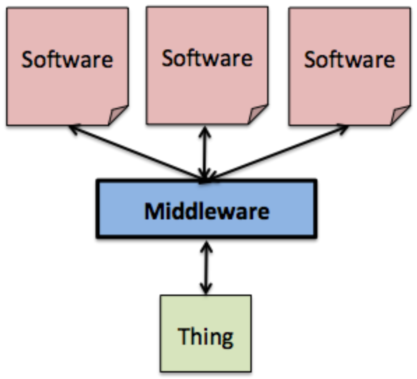
Design patterns for connected things represents the fundamental proposition of an Internet of Things; that is connecting things using networks and software. Here are some examples of fundamental design patterns for IoT:



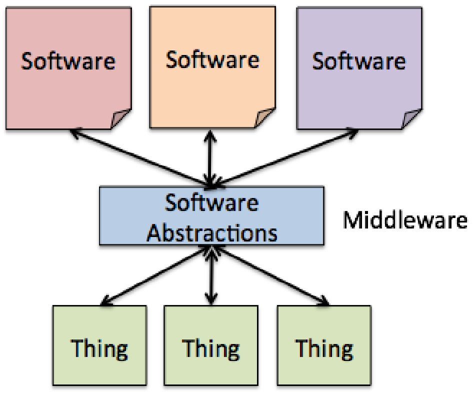
- *Software connected to thing via a network*: the fundamental proposition of the IoT is that connected software is of higher value than embedded software. Connected software is less resource constrained and can integrate more diverse data sources.



- *Virtualization*: Software connecting to an abstract representation of the thing. Virtualization makes it easier to create reusable software and devices.

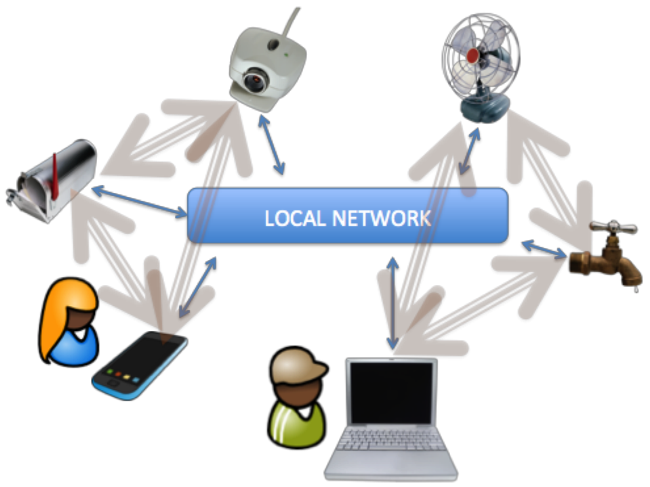


- *Virtualization through middleware*: allows many (web) applications to interact with things. Middleware can cache the state of the thing and minimize network traffic and power drain on constrained devices, and can also serve as a persistent end point for things that aren't reachable over the network due to power cycling, firewalls, etc.

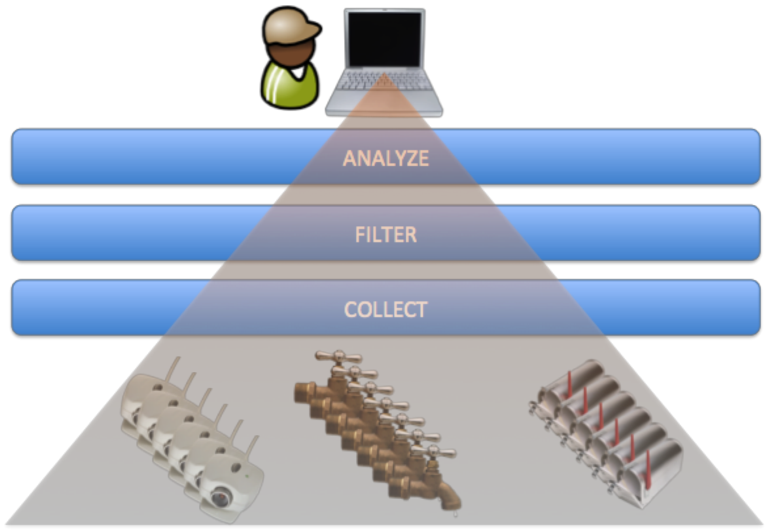


- *Middleware Platform*: Allows many-to-many interactions between applications and things. Enables realization of connected environments and network effects. Can provide standard interfaces for things and application software.
- *Thing-thing interaction*: things that contain some application software interact directly with other things on local networks

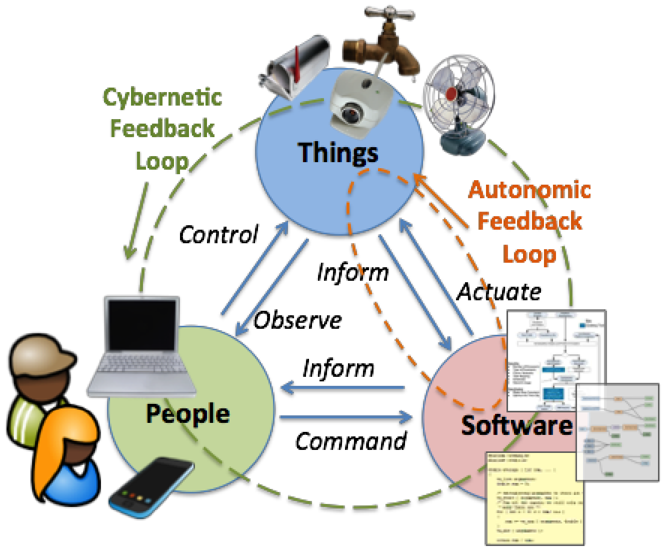
Design patterns for IoT use cases describe the system level use case mapped onto high level design patterns like gateways and web services:



- *Devices talk to other devices peer-to-peer*: local network connectivity enables proximal ad-hoc networking, service federation and chaining, media stream continuity.
- *Personal tracking device uses smartphone as gateway*: common pattern for bluetooth and WiFi connectivity.
- *Smarthome local application controller and gateway*: application gateway pattern.



- *Monitor a large number of devices over a large area*: collect, filter, analyze pattern used in Smart City, surveillance systems, large scale resource monitoring.



- *People interacting with autonomic feedback loops*: general purpose use case involving people and automation in defined roles. Autonomic control relieves people of the job of monitoring, but lets them be involved in high level control and exception handling.

There are many more design patterns at all levels of design. The following list contains some more common patterns based on modern web patterns and practices that are relevant to IoT architecture.

Design patterns for information models consist of lower layers of data models and representation, upon which are built higher level encapsulation and function. Some examples of information model design patterns:

- *Structured data*: XML documents, JSON objects
- *Web Objects*: multiple resources at a URI endpoint, object encapsulation
- *VIrtual Objects, Smart Objects*: a set of resources that represent a physical thing or other data source
- *Composite Objects*: Virtual Objects composed of resources from other objects
- *Hypermedia, HATEOAS*: Applications use pointers to URI endpoints that externalize application state
- *Semantic Hyperlinks*: Hyperlinks with embedded semantic tags, e.g. relation=value
- *Information model*: Collection of semantic hyperlinks describing a resource
- *Context model*: Information model layer describing context of a set of resources
- *Binding model*: Information model describing dynamic binding of actions to resources
- *Resource Directory, Catalog*: A collection of model instances describing sets of resources
- *Resource constructor*: Information model that informs the construction of resource instance
- *Access control model*: Information model that specifies access control policies and constraints for a set of resources

Design Patterns for Interaction describe how different parts of a system interact and communicate with each other, including communication protocols. Some examples:

- *REST*: REpresentational State Transfer, design pattern allowing for externalization of application state in reusable, shareable resources
- *Asynchronous Events*: State updates propagate through the system as they occur
- *Resource Binding*: Associating a resource with an action, bridges REST to Asynchronous Events
- *Observer Pattern*: A binding of resource updates to a protocol action or handler
- *Publish/Subscribe*: A communication pattern where a client registers interest in a topic by subscribing, updates to a topic are published to all subscribers
- *Broker*: A central service to connect publishers with subscribers
- *Proxy*: A machine that provides an interface on behalf of another interface
- *Protocol Bridge*: A bidirectional translator between two protocols
- *Resource Discovery*: A process where resources are found by specifying attributes
- *Resource Registration*: An endpoint informs a resource directory of it's resources
- *Sleeping/Non-reachable Endpoint*: An endpoint is not reachable and must participate in protocol by initiating all interactions with reachable or always-on endpoints

Design patterns for Application Programming describe ways that software and interfaces are created, managed, deployed, and used in IoT applications.

- *REST Objects*: Mapping of REST API resources onto program objects in the application language, using libraries
- *Event handler, onEvent*: application code that responds to asynchronous events
- *Event driven flow*: a set of application handlers that operate in an event driven graph containing series cascade and parallel constructs
- *State Machine*: a logic construct where a next state depends on a set of inputs and the current state, evaluated by a set of logic rules associated with each state
- *State Externalization*: the ability to create stateless application software by mapping application state onto external resources
- *Rule oriented programming*: using a set of rules or rule language to program state machine logic
- *Abstraction of applications*: stateless application software uses application templates for reusability
- *Application templates*: abstract application components with well defined interfaces
- *Modular applications*: applications consisting of one or more reusable components
- *Applications run anywhere, location independent applications*: Application components can run anywhere, in devices, on local network servers, in gateways, in edge servers, in cloud, on user devices.
- *Discovery and Linking*: Integrates resources into applications by resolving resource links, sets attributes in application objects
- *Object Constructor*: Creates application software objects from metadata models

Design patterns for infrastructure describe how different network and device technology is used to solve problems with the physical infrastructure of IoT. How do low power devices connect to wireless sensor networks and ultimately connect to services and applications:

- *6LowPAN edge router*: moves packets from 6LowPAN network to IPV6, does header compression
- *WSN access point*: mediated access from WSN network to IP backbone using a joining protocol e.g. WiFi
- *Mesh routing*: routing network protocol messages through other endpoint nodes in a network
- *Application gateway*: device with both network connectivity and the ability to run application components locally
- *Behind-NAT connectivity*: using reachable service or broker, applications and devices connect to each other from behind NAT firewalls
- *M2M WAN*: Wireless Service Providers specialize in M2M and provide wireless WAN networks, e.g. SigFox

Design patterns for IoT security describe design patterns for IoT security problems.

- *Access control using data models*: semantic hyperlinks control access to resources based on the embedded metadata
- *Social to physical graph relationship*: well defined concepts of ownership and access delegation between people, entities, and things
- *PGP and asymmetric public-key cryptography on devices*: ways of creating SSL sessions and signing data between devices and applications
- *DTLS over UDP*: security for resource constrained devices
- *End-to-end encryption*: transmitting and storing encrypted data independent of channel encryption
- *Device Management*: using device identity, registration, and secure key exchange

There is no one "reference" architecture for an Internet of Things. *Design Patterns* are a way to construct architecture solutions for specific use cases and use case classes.

The work to find system architecture solutions to Internet of Things problems has led to an obvious conclusion that there is no single architecture appropriate for most IoT use cases. The full spectrum of IoT presents a broad range of diverse use cases and resource constraints, and thus motivates a range of architecture solutions. Still, we would like to ground the discussion in a reference set of technical concepts, to help promote a unified understanding and break down the *silos of thought* around IoT architecture. We would also like to find opportunities for standardization and commonality.

Different architecture solutions are appropriate for different use case classes, and architecture is expected to be reusable within a particular class of use cases.Therefore it makes more sense to talk about IoT architecture as a set of *Design Patterns*, working together to achieve an end-to-end solution for some problem.