UNIT III

IOT ARCHITECTURE - IoT Open source architecture (OIC)- OIC Architecture & Design principles- IoT Devices and deployment models- IoTivity : An Open source IoT stack - Overview- IoTivity stack architecture- Resource model and Abstraction.

IOT OPEN SOURCE ARCHITECTURE (OIC)

There is a need for organisations to provide a validated, modular, flexible IoT architecturethat is built to be open, interoperable and cost effective. The architecture should deliver end-to-end open source IoT that addresses enterprise level IoT needs.The characteristics of open source IoT architecture are:
Loosely coupled, modular and secure
Platform independent
Scalable, flexible and can be deployed anywhere
Based on open standards
Streaming analytics and machine learning
Open and interoperable on the hybrid cloud
Application agility and integration

OIC ARCHITECTURE AND DESIGN PRINCIPLESIOT DEVICES AND DEPLOYEMENT MODELS

Different types of deployement models are:-

1)**Public Cloud**
The public cloud makes it possible for anybody to access systems and services. The publiccloud may be less secure as it is open to everyone. The public cloud is one in which cloudinfrastructure services are provided over the internet to the general people or major industrygroups.

2)**Private Cloud**
The private cloud deployment model is the exact opposite of the public cloud deploymentmodel. It’s a one-on-one environment for a single user (customer). There is no need to shareyour hardware with anyone else. The distinction between private and public clouds is in howyou handle all of the hardware.

3)**Hybrid Cloud**
By bridging the public and private worlds with a layer of proprietary software, hybrid cloudcomputing gives the best of both worlds. With a hybrid solution, you may host the app in asafe environment while taking advantage of the public cloud’s cost savings

4)**Community Cloud**

It allows systems and services to be accessible by a group of organizations. It is a distributedsystem that is created by integrating the services of different clouds to address the specificneeds of a community, industry, or business.

IOTIVITY Overview-

IoTivity is an open source software framework enabling seamless device-to-device connectivity toaddress the emerging needs of the Internet of Things. IoTivity is an open source softwareframework implementing OCF.

Architecture

IoTivity is an open-source, reference implementation of the OCF Secure IP Device Framework.

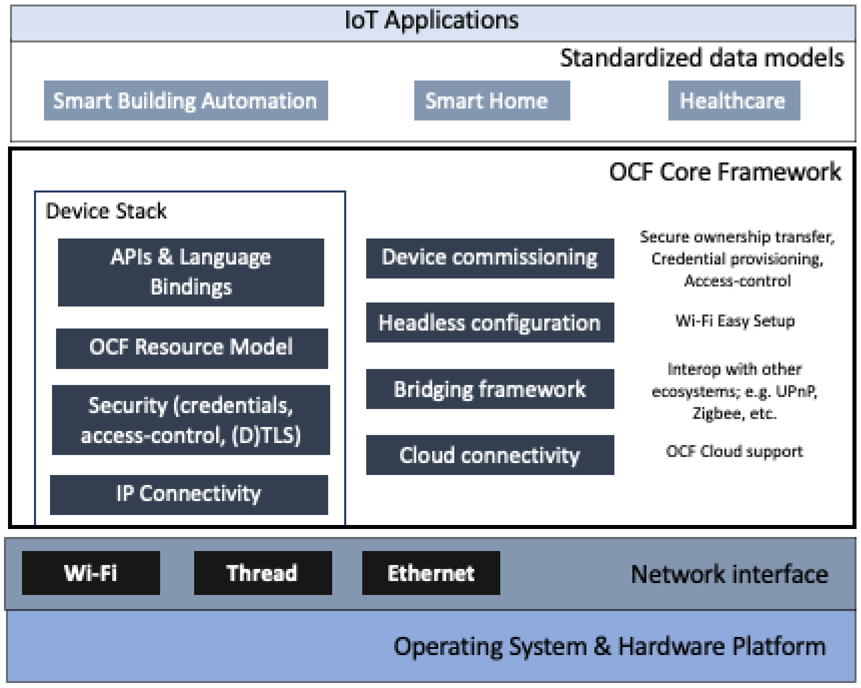
The OCF Secure IP Device Framework provides a versatile communications layer with best-in-class security for Device-to-Device (D2D) and Device-to-Cloud (D2C) connectivity over IP. IoT interoperability is achieved through the use of consensus-derived, industry standard data models spanning an array of usage verticals. The OCF Secure IP Device Framework may be harnessed alongside other IoT technologies in a synergistic fashion to lend a comprehensive and robust IoT solution.

Please review the following resources for more details:

- OCF Security
- Device Commissioning
- Cloud Connectivity
- Bridging
- Headless Configuration

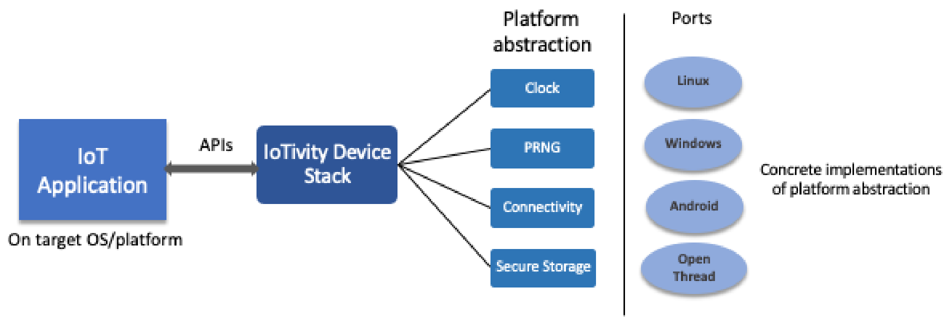
The IoTivity project offers device vendors and application developers royalty-free access to OCF technologies under the Apache 2.0 license.

IoTivity stack FeaturesPermalink



IoTivity device stack and modules

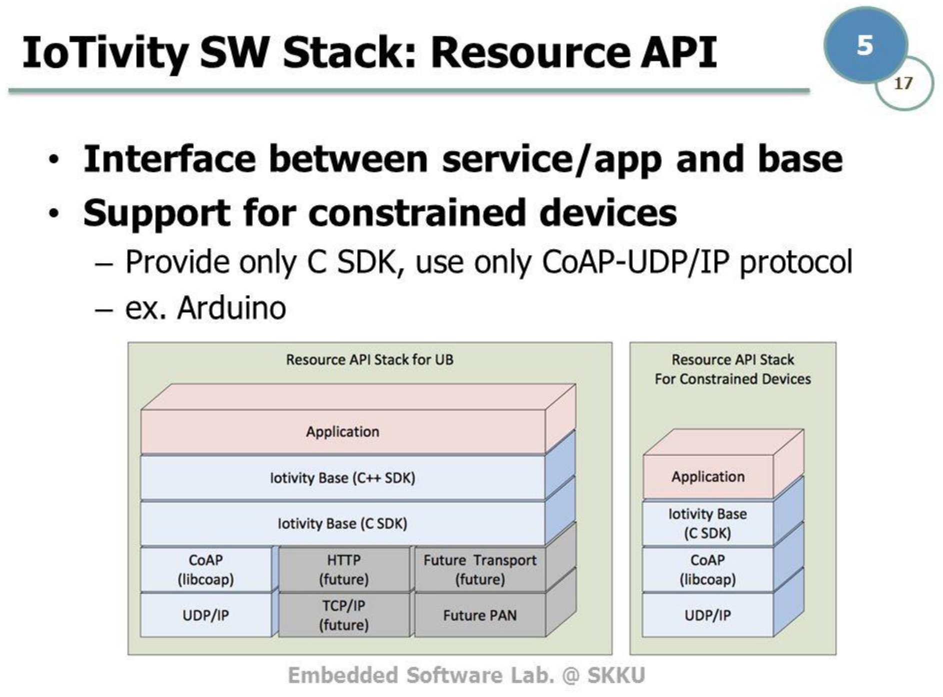
- OS agnostic: The IoTivity device stack and modules work cross-platform (pure C code) and execute in an event-driven style. The stack interacts with lower level OS/hardware platform-specific functionality through a set of abstract interfaces. This decoupling of the common OCF standards related functionality from platform adaptation code promotes ease of long-term maintenance and evolution of the stack through successive releases of the OCF specifications.



IoTivity was designed for rapid portability to any deployment target

- Porting layer: The platform abstraction is a set of generically defined interfaces which elicit a specific contract from implementations. The stack utilizes these interfaces to interact with the underlying OS/platform. The simplicity and boundedness of these interface definitions allow them to be rapidly implemented on any chosen OS/target. Such an implementation constitutes a “port”.

- Optional support for static memory: On minimal environments lacking heap allocation functions, the stack may be configured to statically allocate all internal structures by setting a number of build-time parameters, which by consequence constrain the allowable workload for an application.
- C and Java APIs: The API structure and naming closely aligns with OCF specification constructs, aiding ease of understanding.


IoTivity SW Stack: Resource API

- **Interface between service/app and base**
- **Support for constrained devices**
  - Provide only C SDK, use only CoAP-UDP/IP protocol
  - ex. Arduino

5
17
Resource API Stack for UB
Application
Iotivity Base (C++ SDK)
Iotivity Base (C SDK)
CoAP (libcoap)
HTTP (future)
Future Transport (future)
UDP/IP
TCP/IP (future)
Future PAN
Resource API Stack For Constrained Devices
Application
Iotivity Base (C SDK)
CoAP (libcoap)
UDP/IP
Embedded Software Lab. @ SKKU

Puppet - Resource Abstraction Layer

In Puppet, Resource Abstraction Layer (RAL) can be considered as the core conceptualized model on which the whole infrastructure and Puppet setup works. In RAL, each alphabet has its own significant meaning which is defined as follows.

Resource [R]
A resource can be considered as all the resources which are used to model any configuration in Puppet. They are basically in-built resources which are by default present in Puppet. They can be considered as a set of resources belonging to a pre-defined resource type. They are similar to

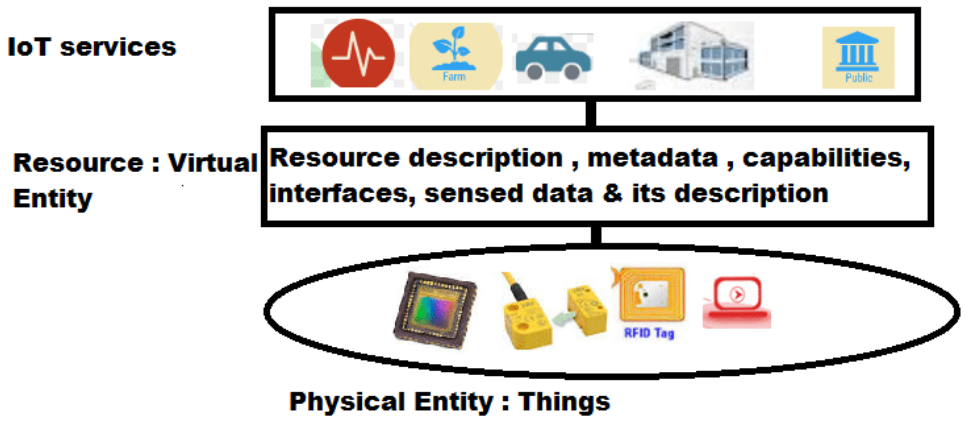
OOP concept in any other programming language wherein the object is an instance of class. In Puppet, its resource is an instance of a resource type.

Abstraction [A]
Abstraction can be considered as a key feature where the resources are defined independently from the target OS. In other words, while writing any manifest file the user need not worry about the target machine or the OS, which is present on that particular machine. In abstraction, resources give enough information about what needs to exist on the Puppet agent.

Puppet will take care of all the functionalities or magic happening behind the scene. Regardless of the resources and OS, Puppet will take care of implementing the configuration on the target machine, wherein the user need not worry how Puppet does behind the scenes.

In abstraction, Puppet separates out the resources from its implementation. This platformspecific configuration exists from providers. We can use multiple subcommands along with its providers.

Layer [L]
It is possible that one defines an entire machine setup and configuration in terms of collection of resources, and it can be viewed and managed via Puppet's CLI interface.


IoT services
Farm
Public
Resource : Virtual Entity
Resource description , metadata , capabilities, interfaces, sensed data & its description
RFID Tag
Physical Entity : Things