**IoTivity Architecture, resource model and abstraction**

**IoTivity (OIC) Overview:** The IoTivity Open-Source Framework (IoTivity) is an open-source project that aims to provide a standardized and interoperable framework for the IoT. IoTivity is designed to connect and manage IoT devices and services, allowing them to work together seamlessly.

* Standardization and Interoperability: IoTivity is designed to bridge the gap between different IoT devices and platforms, ensuring that they can communicate and work together, regardless of their manufacturer or underlying technology. This standardization is essential for achieving true interoperability in the IoT ecosystem.
* Open Source: IoTivity is an open-source project under the Linux Foundation. This open-source approach fosters collaboration, transparency, and widespread adoption. The source code is freely available, allowing developers to modify and contribute to the project.
* Cross-Platform Compatibility: IoTivity is platform-agnostic, making it compatible with various operating systems and hardware. It provides software development kits (SDKs) for different platforms, including Linux, Android, and Windows, to facilitate cross-platform development.
* Protocols and Standards: IoTivity adheres to established IoT protocols and standards. It relies on CoAP (Constrained Application Protocol) for device communication and the OASIS OSLC (Open Services for Lifecycle Collaboration) standard for device discovery and management.
* Security: Security is a top priority in IoTivity. It incorporates secure communication protocols, including DTLS (Datagram Transport Layer Security), to protect data in transit. Device authentication and authorization mechanisms are also a key part of its security framework.

**Design Principles of IoTivity:** IoTivity is built on several key design principles to ensure its effectiveness and usability in the IoT landscape:

* Open Source: Being open source, IoTivity encourages community-driven development and collaboration.
* Resource-Oriented Model: IoTivity follows a resource-oriented approach. Everything in the IoTivity framework, including devices and services, is represented as resources. Each resource is identified by a unique URI, and standard methods (POST (create), GET (read), PUT (update), DELETE) can be used to interact with these resources.
* Interoperability: IoTivity ensures that devices from different manufacturers can work together seamlessly by adhering to common standards.
* Layered Architecture: IoTivity follows a layered architectural approach. It consists of several layers, including the Application Layer, Connectivity Abstraction Layer, and the Device Layer. Each layer has well-defined functions and APIs, allowing for easy extensibility and customization.
* Device Discovery and Management: IoTivity provides comprehensive device discovery and management capabilities. Devices can be discovered on the network, and their capabilities are exposed through resource descriptions. This enables easy integration and control of IoT devices.
* Plug and Play: IoTivity is designed for plug-and-play interoperability. Devices can be seamlessly integrated into the network, and their functionalities can be discovered automatically, reducing the complexity of setting up IoT systems.
* Cross-Platform Support: IoTivity offers SDKs (Software Development Kits) and APIs (Application Programming Interfaces) for various platforms, making it easier for developers to create applications that run on different operating systems and hardware.
* Scalability: IoTivity is scalable from small-scale IoT deployments to large-scale, enterprise-level systems. It is suitable for both consumer and industrial IoT applications.
* Security and Privacy: Security and privacy are paramount in IoTivity's design. It provides mechanisms for secure communication, authentication, and authorization, ensuring the confidentiality and integrity of data exchanged between devices.
* Community-Driven Development: As an open-source project, IoTivity thrives on community contributions. This collaborative approach ensures continuous improvement, bug fixes, and the development of new features.
* Sustainability: IoTivity aims to be sustainable over the long term by encouraging industry-wide adoption and adherence to standards.

**IoTivity stack architecture:** The IoTivity stack architecture is designed to be modular and extensible. This makes it easy to add new features and functionality to the stack, and to port the stack to new devices and platforms. IoTivity's stack architecture is divided into several layers, each responsible for specific functionalities. The key layers include:

* **Porting layer:** The porting layer provides a set of abstract interfaces that the IoTivity stack uses to interact with the underlying operating system and hardware platform. This includes interfaces for networking, security, and memory management. The porting layer also provides APIs for the IoTivity stack to access device hardware, such as sensors and actuators. The porting layer is implemented using a platform-specific adapter for each operating system and hardware platform that the IoTivity stack supports. This allows the IoTivity stack to be portable to a wide range of devices, from resource-constrained devices such as microcontrollers to more powerful devices such as servers.
* **Core layer:** The core layer provides the core functionality of the IoTivity stack, such as device discovery, resource management, and data exchange. The core layer also implements the OIC Core Specification.

It includes the following key components:

* + Resource Directory (RD): This component is responsible for keeping track of all available resources (IoT devices, services, and data) in the network. It facilitates resource discovery.
  + Resource Container (RC): It stores and manages resources, making them accessible to applications and other devices in the network.
  + CoAP (Constrained Application Protocol) Engine: CoAP is a lightweight and efficient protocol designed for IoT. The CoAP engine handles communication between devices, allowing them to exchange data, commands, and status information.

The core layer provides the following functionalities:

* + Device discovery: The core layer provides APIs for applications to discover IoTivity devices on the local network.
  + Resource management: The core layer provides APIs for applications to manage resources on IoTivity devices. This includes APIs for creating, reading, updating, and deleting resources.
  + Data exchange: The core layer provides APIs for applications to exchange data with IoTivity devices. This data can be in the form of sensor data, actuator commands, or any other type of data.

The core layer is implemented using a set of protocols that are designed to be lightweight and efficient. This makes the core layer suitable for use on a wide range of devices, including resource-constrained devices.

* **Service layer:** The service layer builds on top of the core layer to provide additional functionality, such as device management, security, and data storage. The service layer also provides APIs that can be used by applications to interact with IoTivity devices. The service layer provides the following functionality:
  + Device management: The service layer provides APIs for applications to manage IoTivity devices. This includes APIs for provisioning devices, configuring devices, and updating firmware.
  + Security: The service layer provides APIs for applications to secure IoTivity devices. This includes APIs for authentication, authorization, and encryption.
  + Data storage: The service layer provides APIs for applications to store data from IoTivity devices. This data can be stored on the local device, on a cloud server, or on a hybrid storage system.

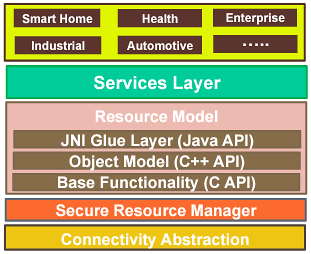
The service layer is implemented using a set of protocols and services that are designed to be secure and scalable. This makes the service layer suitable for use in enterprise-grade IoT applications.

* **Application layer:** The application layer contains the applications that interact with IoTivity devices. Applications can use the APIs provided by the service layer to discover devices, access resources, and exchange data.

Applications can be developed for a variety of platforms, including mobile devices, desktop computers, and servers. Applications can also be developed for specific use cases, such as smart home automation, industrial IoT, or smart city applications.

The IoTivity stack architecture provides a flexible and extensible platform for developing and deploying IoT applications. The modular architecture makes it easy to add new features and functionality to the stack, and to port the stack to new devices and platforms. The IoTivity stack is also designed to be secure and scalable, making it suitable for use in a wide range of IoT applications.

* **Connectivity Abstraction Layer:** Below the Application Layer is the Connectivity Abstraction Layer. It is responsible for abstracting the underlying network technologies and communication protocols. This layer ensures that applications can communicate over various network technologies seamlessly.
* **Device Layer:** Below the Core Layer is the Device Layer. This layer represents the actual IoT devices. It interacts with the Core Layer to expose device capabilities as resources, making them available for control and monitoring.
* **Physical Layer:** The Physical Layer comprises the actual hardware components of IoT devices, such as sensors, actuators, communication interfaces, and other physical components.

****

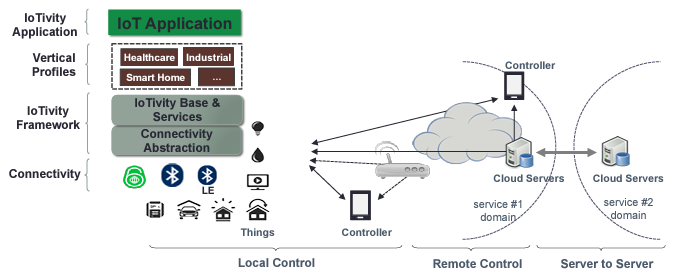
**IoTivity Components and Modules:** IoTivity includes various components and modules that support its architecture:

* Resource: Resources represent IoT devices, services, and data. They are the primary entities in IoTivity. Each resource is identified by a unique URI and can be manipulated using standard methods like GET, PUT, POST, and DELETE.
* Resource Server: The Resource Server manages and exposes resources to the network. It handles incoming requests and ensures that resources are accessible for reading or modification.
* Resource Client: The Resource Client is responsible for sending requests to Resource Servers to interact with resources. It can be used by applications to access and control IoT devices and services.
* Resource Directory (RD): The RD maintains a directory of all available resources in the network. It helps devices discover and locate resources efficiently.
* Resource Container (RC): The RC is responsible for managing and storing resources. It ensures that resources are accessible for applications and other devices.
* Security: Security is a critical component in IoTivity. It provides mechanisms for secure communication, authentication, and authorization to protect data in transit and at rest.

**Why is IoTivity useful?**

* Cross-platform support
* Uniform and easy-to-use APIs
* Based on open standards
* Support for multiple connectivity types
* Extensible to support proprietary protocols

**Role in IoT ecosystem**

****

**Resource Model**

The resource model in IoTivity is central to its architecture. The IoTivity resource model is a data model that defines how IoT devices and applications expose their capabilities. The resource model is based on the concept of resources, which are logical entities that represent the capabilities of a device or application.

Resources are identified by unique URIs and can have a variety of properties, such as name, type, and value. Resources can also be organized into hierarchies.

It represents the key entities within the IoT ecosystem, including devices, services, and data, as resources. Each resource is identified by a unique Uniform Resource Identifier (URI), and it is accessible using standard methods such as GET, PUT, POST, and DELETE.

The resource model simplifies the way IoT devices and data are presented, making them universally accessible and manageable. The IoTivity resource model is designed to be flexible and extensible. This allows developers to define resources that are specific to their needs. However, the IoTivity Core Specification defines a set of common resources that are supported by all IoTivity devices.

Key characteristics of the resource model in IoTivity include:

* **Resource Identification:** Every resource has a unique URI that provides a standard way to locate and access it within the IoT ecosystem. For example, a resource representing a smart light bulb might have a URI like /devices/lighting/smartbulb.
* **Resource Representation:** Resources can represent both physical devices and services, as well as abstract data. They provide a standard representation for their state, capabilities, and available interactions. This representation is typically in the form of a data structure that can be read and modified.
* **Resource State:** Resources can have different states or attributes that describe their properties. For instance, a smart thermostat resource might have attributes for the current temperature setting, mode (heating or cooling), and operational status.
* **Resource Actions:** Resources can expose actions that can be performed on them. These actions are defined using standard methods, such as GET for reading, PUT for updating, POST for executing a specific function, and DELETE for removing.
* **Resource Relationships:** Resources can have relationships with other resources. For example, a smart home resource representing a room may have relationships with resources representing devices in that room, such as lights, thermostats, and locks.

Some examples of common IoTivity resources include:

* Light resource: This resource represents a light bulb and can be used to turn the light on or off, adjust the brightness, and change the color.
* Temperature sensor resource: This resource represents a temperature sensor and can be used to read the current temperature.
* Door lock resource: This resource represents a door lock and can be used to lock or unlock the door.

**Abstraction in IoTivity**

The IoTivity stack provides a layer of abstraction between applications and IoT devices. Abstraction in IoTivity refers to the process of simplifying the complexities of IoT devices, services, and data to make them more accessible and manageable. This simplification involves abstracting the underlying hardware and communication protocols, allowing developers and applications to interact with IoT entities in a uniform way.

The abstraction layer is implemented by the core layer of the IoTivity stack. The core layer provides a set of APIs that applications can use to interact with IoTivity devices. These APIs are independent of the specific device hardware and communication protocols.

Key aspects of abstraction in IoTivity include:

* **Communication Abstraction:** IoTivity abstracts the underlying communication protocols and technologies. This means that developers do not need to be concerned with the specifics of how devices communicate over different networks. The framework takes care of the communication details.
* **Network Abstraction:** IoTivity provides network abstraction, allowing IoT devices to work seamlessly over various network types, such as Wi-Fi, Bluetooth, Ethernet, and cellular networks. This abstraction ensures cross-network compatibility.
* **Device Abstraction:** Abstraction at the device level allows IoT devices to be represented as resources, making it easy for applications to discover and interact with them. Device capabilities are exposed through resources, simplifying control and monitoring.
* **Resource Abstraction:** IoTivity abstracts the representation of resources. Applications can access and manipulate resources using standard methods, without needing to understand the complexities of the devices or services they represent.
* **Interoperability:** The abstraction layer ensures that resources from different devices and manufacturers can be controlled and monitored in a uniform way, promoting interoperability in the IoT ecosystem.

The abstraction layer provided by the IoTivity stack has several benefits:

* It makes it easier for developers to develop applications for IoT devices. Developers do not need to know the details of the device hardware or communication protocols.
* It allows applications to be ported to different IoT devices without having to be rewritten.
* It improves the security of IoT applications by hiding the underlying details of the device hardware and communication protocols from attackers.

**Example**

The following example shows how an application can use the IoTivity stack to interact with a light bulb:

// Get a handle to the light bulb resource.

OicResource \*lightResource = GetResource("/light");

// Turn the light on.

SetProperty(lightResource, "power", "on");

// Wait for the light to turn on.

WaitForPropertyChange(lightResource, "power");

// Turn the light off.

SetProperty(lightResource, "power", "off");

// Wait for the light to turn off.

WaitForPropertyChange(lightResource, "power");

The application does not need to know the details of the device hardware or communication protocols to interact with the light bulb. The IoTivity stack handles all the underlying details.