WoT.City: Decentralized Internet of Things Software Framework for a Peer-to-Peer and Interoperable IoT Device

Jollen Chen

January 5, 2017

WoT.City Open Source Project,
`jollen@wotcity.com`

Abstract. In recent years, the development of Internet of Things (IoT) applications has become increasingly complex. Some studies have attempted to address this problem. The common characteristic of these studies is the ability to stream data to the cloud over the web. Moreover, most software architectures presented in these studies do not provide IoT interoperability; however, the OpenIoT platform does provide interoperable IoT applications[1]. Notwithstanding several studies, most methods do not provide peer-to-peer networking. This paper proposes a software architecture that provides peer-to-peer IoT networking and interoperable IoT application framework. This software architecture also provides a flow-based programming environment for writing IoT applications.

Keywords: Internet of Things, Interoperability, Peer to Peer, Web of Things, Flow-Based, Decentralized

1. Introduction

The motivation of this paper is to propose a new design software architecture for the development of interoperable Internet of Things (IoT) applications. The software architecture adopts W3C's Web of Things (WoT) ontology. In addition, the software architecture also provides peer-to-peer networking capabilities for a decentralized IoT model.

This paper presents an overview of the WoT.City software framework. The software framework is currently available as an open-source project, and it consists of three sub-projects. Accessible at (a) wotcity.io (https://github.com/wotcity), (b) devify.io (https://github.com/DevifyPlatform), and (c) flowchain.io (https://github.com/flowchain). Given JavaScript's current ubiquitous nature, it is natural to use a JavaScript platform for the IoT. Thus, WoT.City offers a 100% JavaScript platform for the IoT.

2. Architectural Description

W3C's WoT ontology provides a standards-based model to represent a physical device, or the so-called physical object, as an application server [2]. An IoT application server comprises a virtual representation of a physical object, and the application runs as an application server on an IoT device. An IoT application might be installed and run on an application processor-based high-performance device or a microcontroller device. Typically, the Node.js JavaScript runtime is used for high-performance devices, a JavaScript engines, such as the Mongoose full-stack IoT platform [3] and JerryScript [4] are used for microcontroller-based resource-constrained devices.

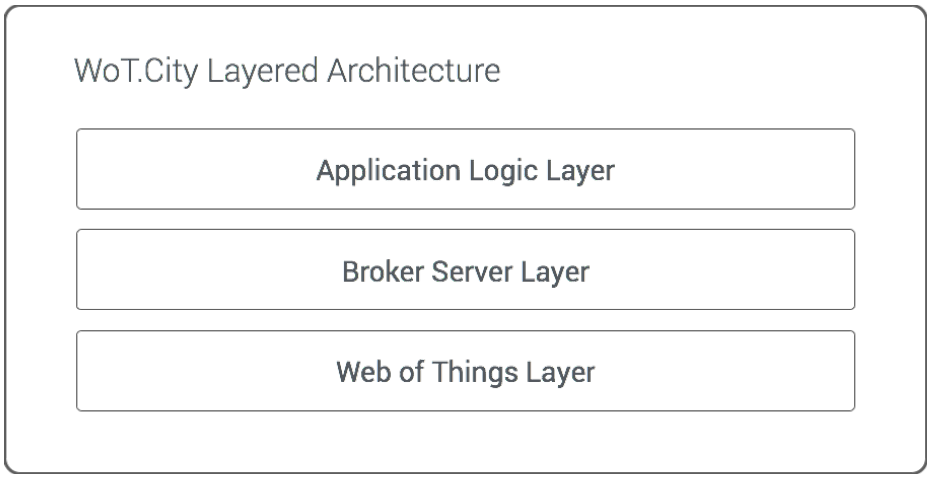

Fig. 1. Architectural design of WoT.City framework

As shown in Fig. 1, the WoT.City full-stack software framework is a three-layer design. The first layer is responsible for dealing with HTTP, CoAP, and the WebSocket communication protocol. The first layer also describes the IoT device properties. This layer uses the WoT model. Accordingly, a physical device is represented as a virtual thing with (a) protocols binding, (b) URL routing, (c) event emitting and (d) request handler functionalities.

The broker server layer implements peer-to-peer communication, REST-style RPC operations and a distributed hash table (DHT). The code size of the broker server layer is extremely light weight; thus, it can run on laptops, mobile devices, and even resource-constrained devices. It aims to help the WoT layer create IoT application servers. To begin developing customized IoT application servers, several project boilerplates can be downloaded through the WoT.City open source project.

An IoT application typically uses the flow-based programming (FBP) paradigm. The FBP paradigm defines applications as networks of black box processes that exchange data across predefined connections by message passing[5]. Thus, the application layer offers an FBP environment.

In addition, FBP is naturally component-oriented, and with WoT.City's light weight FBP engine, applications can be assembled with predefined components that can connect as data processing networks.

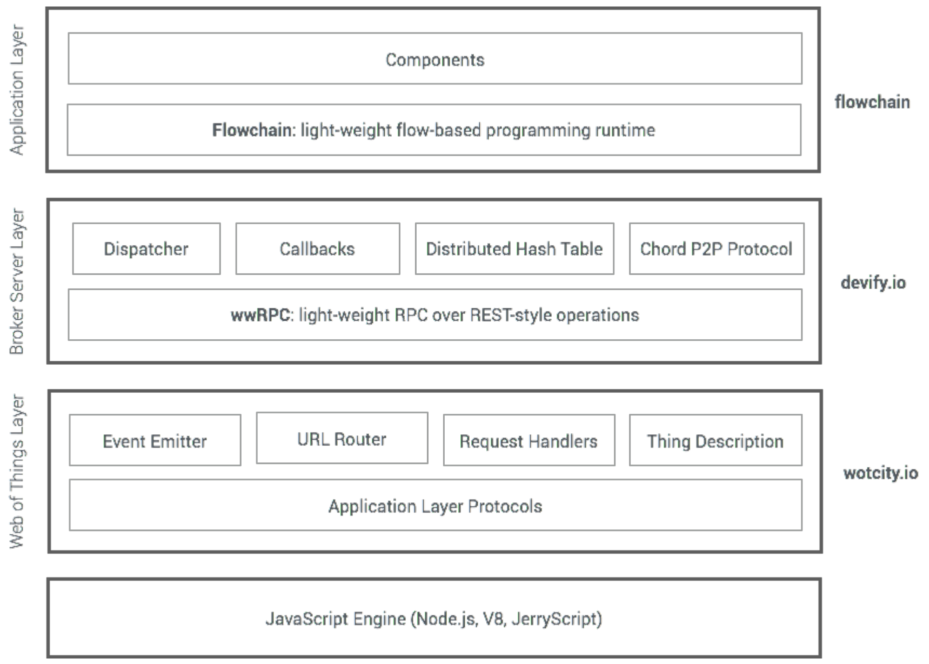

Fig. 2. WoT.City framework components

Figure 2 shows the software components of the WoT.City framework. The WoT is a use case for open markets of applications and services based on the roles of web technologies. Moreover, it manages a physical device as a "Virtual Thing with Thing Properties." To implement this model, the "Thing Description" component of the WoT.City framework describes "Thing Properties" in the JSON data format. Consequently, the WoT represents the Virtual Thing in URI convention. As shown in Listing 1, the URL Router component of the WoT.City framework defines URIs to represent a Virtual Thing.

Listing 1. Virtual Thing in URI convention

```
[coap|ws]://[hostname]/object/[name]/send
[coap|ws]://[hostname]/object/[name]/viewer
```

A Virtual Thing is referred to as a “node” in this paper. In the WoT.City framework, CoAP and WebSocket are the primary protocol bindings for a node. CoAP is an application layer protocol intended for applications on constrained devices. The WebSocket protocol provides a standardized way to facilitate real-time data transfer.

As shown in Listing 1, the URIs can represent two types of nodes, i.e., sender and viewer nodes. A sender node transfers time-series data to another node over WebSocket or CoAP. A viewer node receives time-series data from other nodes. In addition, the Request Handlers component accepts incoming requests, receives data and triggers necessary events.

The WoT layer makes distribution possible by providing a service contract without exposing server side implementation details [6]. In addition, the broker server layer encapsulates the peer-to-peer and RPC technical details on the device server side. Typically, the broker architectural pattern can be utilized to structure distributed software systems with decoupled components that interact by remote service invocations [7]. Thus, the broker server layer implements the broker architectural pattern to hide such technical details.

3. Peer-to-Peer Network

MQTT, a frequently referenced IoT technology, is a publish-subscribe-based lightweight messaging protocol for use on top of TCP/IP. In MQTT networks, connected nodes are managed by MQTT brokers. In addition, MQTT brokers export their nodes for external visibility. Note that the WoT.City framework’s broker server does not export its nodes.

As shown in Fig. 3, in WoT.City networks, nodes connected to the same broker are “grouped” together with their broker as a virtual node, i.e., a virtual node comprises a broker and its corresponding nodes. In this manner, the broker is responsible for managing its connected nodes. Moreover, these brokers are organized as a peer-to-peer networ using the DHT.

Unlike MQTT, the WoT.City broker does not export its nodes; it hides nodes such that all nodes are internal and not visible externally.

As shown in Fig. 2, the DHT and Chord P2P Protocol are key components of the peer-to-peer networks. In addition, web-to-web RPC (wwRPC) exists is another key component of the WoT.City framework. The wwRPC component offers REST-style RPC operations and collaborates with the DHT. As a result, to enable such capabilities, the device must have a WoT.City application server installed on it. Writing a WoT.City broker server is simple, as shown in Listing 2. The broker server starts and subsequently joins a peer-to-peer network.

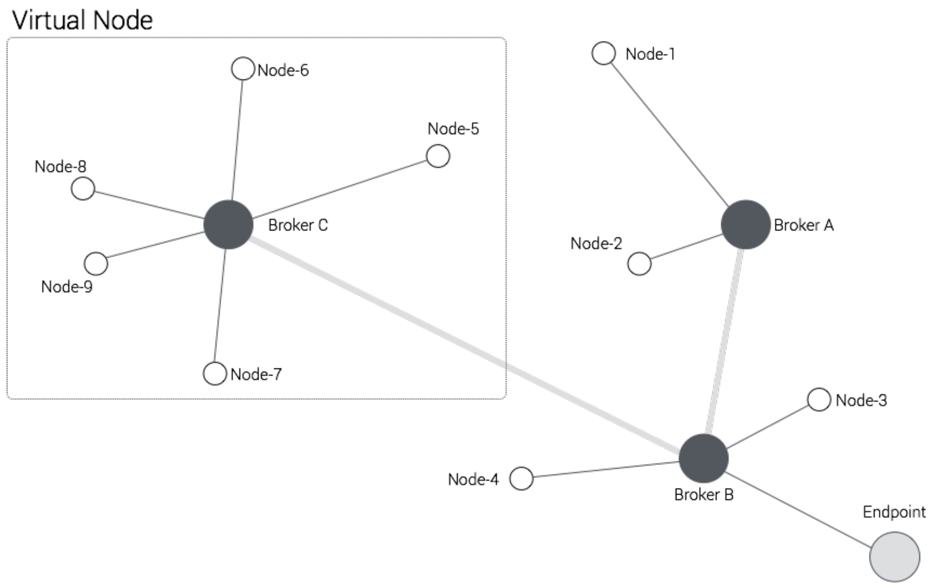

Fig. 3. WoT.City framework network topology

Listing 2. Sample broker program code

```
// Require Broker class in Devify Platform
var DevifyBroker = require('devify.io').Broker;

// To instantiate a broker server instance
var broker = new DevifyBroker({
    host: '192.168.0.1', port: 8000,
    join: {address: '192.168.0.100', port: 8000}
});

// To start the broker server
broker.start();

// The virtual node is up and listening
```

A broker server also offers periodic health and failure checks for nodes. As mentioned previously, the broker hides all implementation details. Moreover, as shown in Fig. 4, the broker server can forward request to an “endpoint.” Ordinarily, the endpoint is a cloud-based REST API platform, such as Dropbox and Twilio or another broker server.

As shown in Fig. 2, the Chord P2P Protocol component implements the Chord peer-to-peer protocol. Chord is a protocol and algorithm for a peer-to-peer DHT.

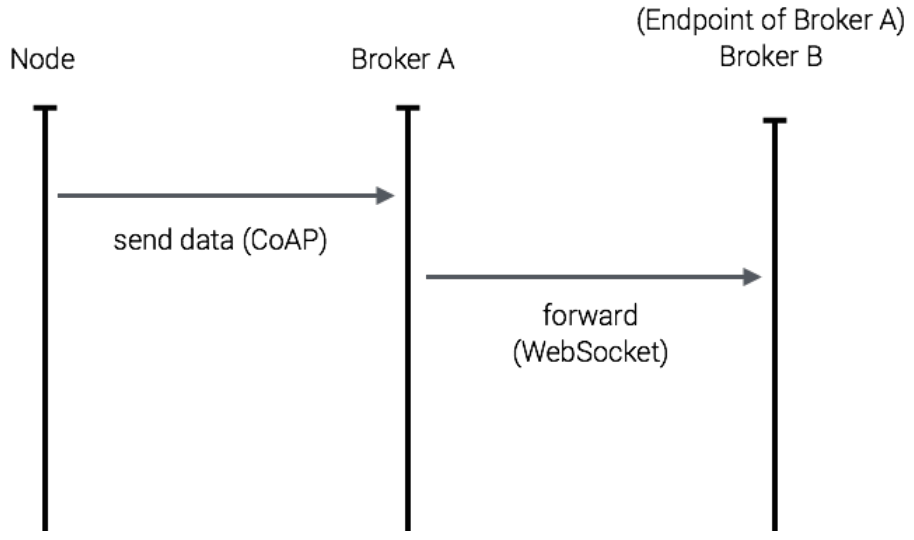

Fig. 4. Example of request forwarding

A DHT stores key-value pairs by assigning keys to different nodes [8]. Chord messages are sent to peer nodes via wwRPC. The dispatcher component receives and dispatches these RPC messages. Another significant design of wwRPC is that the dispatcher component uses an event-driven concurrency model to handle RPC messages. Currently, many developers avoid the multi-thread model and employ an event-driven approach to concurrency management [9]. Consequently, due to the scalability limits of threads, wwRPC uses an event-driven model.

4. IoT Applications

WoT.City presents a software framework that simplifies the creation of IoT applications by reusing existing web technologies and applying the FBP programming paradigm. For example, setting up a sensor node to gather information and communicate with other nodes requires only a few lines of code. Moreover, to build a complete peer-to-peer and decentralized IoT network also requires only a few lines of code.

The FBP paradigm defines applications as networks and exchanges data across predefined connections[10]. WoT.City utilizes the FBP paradigm for IoT application development. Thus, the WoT.City software framework can be used to write flow-based IoT applications.

Developers can write IoT application code using the FBP paradigm and JavaScript. With the FBP paradigm, an IoT application is described by components and their corresponding connections. As shown in Fig. 5, an

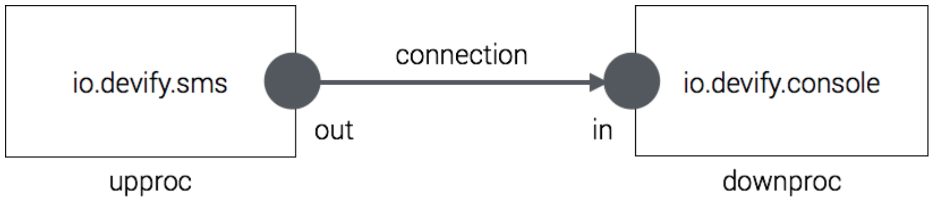

Fig. 5. Flowchain: FBP runtime engine

application is described as a “graph” in JSON format. Listing 3 shows this implementation.

Listing 3. Flowchain application sample (the “graph”)

```
{
    "type": "coapBroker",
    "connections": [
        {
            "upproc": "io.devify.sms,
            "upport": "out",
            "downproc": "io.devify.console,
            "downport": "in"
        }
    ]
}
```

Flowchain internal is an FBP-like system in 100% JavaScript with a unidirectional data flow design. Consequently, the Flowchain has a single output port and a single input port. The unidirectional data flow reduces the complexity of device interoperability. The “connection” is created from the “outPort” of one component to the “inPort” of another component. The Flowchain runtime engine is responsible for executing the “graph” and processing the data flow. Moreover, the FBP components are highly decoupled; thus, developers can build and publish the reusable components.

5. Conclusions

The mission of WoT.City is to establish an open-source project for decentralized IoT software framework. Furthermore, WoT.City convergences emerging IoT trends: (a) a full-stack JavaScript software framework, (b) device interoperability via REST-style RPC operations, (c) a peer-to-peer network for a decentralized IoT model, and (d) an FBP model for IoT applications. All work is available at GitHub as open-source projects (Fig. 6).

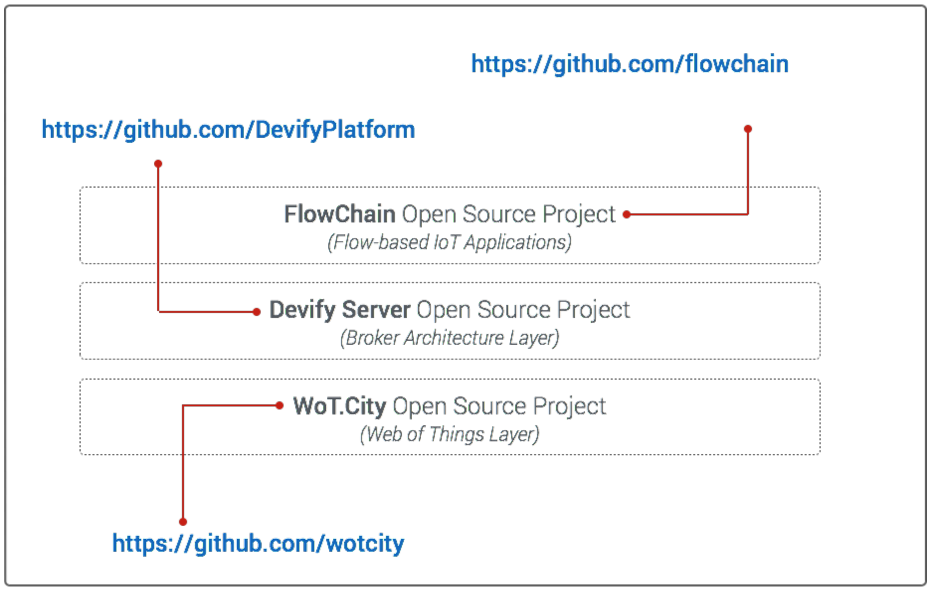
https://github.com/flowchain
https://github.com/DevifyPlatform
FlowChain Open Source Project
(Flow-based IoT Applications)
Devify Server Open Source Project
(Broker Architecture Layer)
WoT.City Open Source Project
(Web of Things Layer)
https://github.com/wotcity

Fig. 6. WoT.City open source project

Subsequently, IoT devices in a decentralized IoT network may require a new model to exchange data. The data exchange model must be more secure and ensure data privacy. Moreover, WoT.City has begun to build a blockchain-based decentralized IoT platform.

Future work. Given the blockchain's private ledger nature, it is natural that WoT.City will use blockchain technology to provide secured and trusted data exchange. In other words, WoT.City will use blockchain technology to maintain trusted records of all data exchanged between devices rather than exchanging data through a centralized IoT platform.

6. References

1. Soldatos, John (et al.): OpenIoT: Open Source Internet-of-Things in the Cloud. In: Pro-ceedings of International Workshop on Interoperability and Open-Source Solutions for the Internet of Things, pp. 13-25. Springer (2015).
2. Dave Raggett. An introduction to the Web of Things Framework. https://www.w3.org/2015/05/wot-framework.pdf
3. Mongoose full-stack IoT platform, https://mongoose-iot.com
4. Ultra-lightweight JavaScript engine for the Internet of Things, https://github.com/Samsung/jerryscript
5. Flow-based programming, https://en.wikipedia.org/wiki/Flow-based_programming

6. Web of Things Interest Group Charter, https://www.w3.org/2014/12/wot-ig-charter.html
7. Buschmann, Frank, et al. Pattern-Oriented Software Architecture. John Wiley & Sons Ltd, 1996.
8. Stoica, Morris (et al.): Chord: A scalable peer-to-peer lookup service for internet applica-tions. ACM SIGCOMM Computer Communication Review. 31 (4): 149.
9. Welsh, M., Culler (et al.): SEDA: An Architecture for Well-Conditioned, Scalable Inter-net Services. In Proc. 18th Symposium on Operating Systems Principles (SOSP-18), Banff, Canada, 2001.
10. J. Paul Morrison: Data Stream Linkage Mechanism. IBM Systems Journal Vol. 17, No. 4, 1978