A Platform for Controlling Remote Applications (Pending Title) (revised November 2016)

Jason D. Walker, Dr. George Markowsky, *Official Advisor, Professor of Computer Science, and* Zachary Hutchinson, *Unofficial Advisor, Ph. D Student University of Maine Orono*

Abstract—

Index Terms—

1 Introduction

The Internet of Things is a big buzzword right now. Currently there are a few major key players in the arena, however, there is no agreed upon standard for IoT. It is estimated that there will be 20.8 billion things connected by 2020. [1] This is an incredible number of connected devices and poses multiple questions on how all these things will be implemented let alone the security aspect of it all.

1.1 Problem

Products like the Nest and the Philips Hue lightbulb each have their own GUI and dedicated server to run the devices. The main problem is that while the IoT field is booming so isn't the required number of mobile and web applications required to run them. The field of IoT would benefit from having a dynamic and unified platform with an easy to use graphical user interface (GUI). A unified platform would create a common standard for embedding remote capabilities into any given application. While there is no standard framework currently being used there are common tools that are being used to build similar platforms.

This platform would be used by hobbyists or people who like to tinker with smart devices. Because so many smart devices have their own dedicated server to run them, building a simple unified platform that can be emebedded into virtually any application would be beneficial for the future of innovation in IoT. Allowing hobbyists to innovate and not have to worry about building a complete server setup would allow them to focus more on innovating new smart devices and less on the server logistics.

It's important to note that while this platform is being developed mainly for use with IoT devices and their applications, the platform is application independent and can be used to manipulate the state of any application. An in browser music player being controlled remotely is an example of being application independent. We can build a web application for a music player. We can embed some code that can call various functions of the application. This is no different from creating a Node.js application on a Raspberry Pi or other various IoT setups and embedding the same code to control the various functions of the IoT device application.

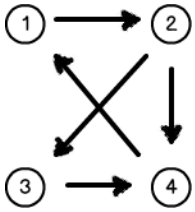
1.2 Purpose of Study

This project focuses on the implementation of the platform and not on the security aspect. The security aspect of IoT is a legitimate concern and should not be taken lightly. My focus however, is on the usage of embedding "remote control" logic within various applications. Some of the questions I will answer are (to be refined):

- Is there a need for a simplified and unified GUI for IoT devices and remote controlled applications? How many apps is to many?
- Can complex deterministic finite state automata machines be described and manipulated using a graph theory and Dijkstra's algorithm?
- How can a user control such complex machines from afar? What is the proposed and community accepted (thus far) standard for message passing?
- Can these deterministic FSA machines be dependent on other machines?

1.3 Approach

Directed cyclical graphs can be used to describe transitions of state in an application. These applications can be as simple as an application that manipulates the pins on a Raspberry Pi for light bulbs with two states. Directed cyclical graphs can also be used to describe more complex applications and the various states and their transitions. The question is whether we can use these graphs in an implementation to control the states (and transitions) of these machines. For example, a DFSA graph can be described using the an adjacency matrix depicted in *Figure 1*:



{
1: {2:1},
2: {3:1, 4:1},
3: {4:1},
4: {1:1},
}

Figure. 1 – Cyclic Directed Graph (Left), Corresponding Adjacency Matrix (Right)

In *Figure 1* the adjacency matrix has a series of states (1-4) and its corresponding edges, each edge has a weight of 1. If we take this matrix in JSON format and we read this in, we can reconstruct a graph depicting a directed cyclic graph. We can use the directed cyclic graph as a state machine. Through a variety of methods including; utilizing a RESTful interface, a MQTT message broker, and Dijkstra's algorithm. We can manipulate the directed graphs into transitioning between states. We can then build a Queue that would utilize the MQTT message broker to communicate with the application of the required function needs to be executed. In *Figure 3* you can see that the basic structure of the architecture.

the application. The server will build a queue of actions need to be taken. The MQTT message broker will send the command consisting of a function to execute to the application to handle one by one until the queue is empty. This will happen asynchronously as to not prevent the server from executing other incoming requests.

In the most basic sense the system will take in directed cyclic graph depicted by a JSON formatted document and a desired state. The system will then output a series of actions or transitions needed to be in that state. This system will be integrated with a RESTful service and MQTT message broker that will allow quick communication between the client and the server. An integral part of the project will revolve around the directed cyclic graph transitions and their dependencies and how we can use those to create complex applications. Think of this as grouping devices or applications together to control them (i.e. turning three lights on in a room).

2 Scope

2.1 Project Definition

This project will mainly focus on just the implementation of the system architecture described previously. Meaning anything that corresponds to getting applications to

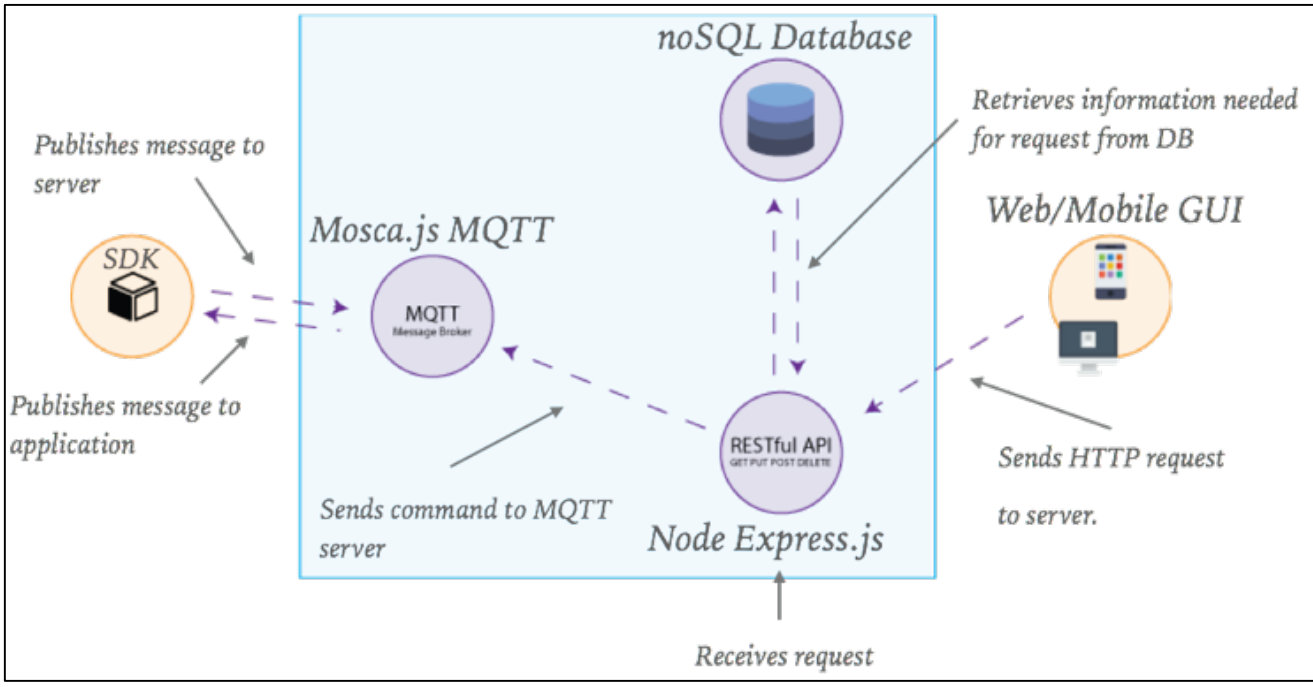


Figure 3 – Basic Architecture Network Flow Diagram

The diagram above describes the basic architecture as I currently have it planned. The GUI can be a web application or a mobile app. This is what triggers the desired state. When you tap a button on the GUI that is meant to trigger an action within the application it sends the request to the servers. The servers are integrated and connected to the database. The database holds information about each application connected to the network and the current state of

transition to different states using the GUI. Ideally in this project a user will hit an action on the GUI that will trigger an action to get the device to transition to different state such as turning a light on or pressing next in a music application. The action will trigger the directed cyclic graph to figure out the path needed to be taken and the functions it needs to execute to get the device/s into their proper state/s.

2.2 Components

The framework contains a few pieces of critical components. Some of the main components are:

- Web Application and/or Smart Phone Application
- RESTful Interface
- MQTT Message Broker
- Directed cyclic graph processing logic
- Mongo Database
- Device Library

The web application or smart phone application will consist of an interface that lists all their devices and their current states. They will be able to tap/click on a button that will trigger a state change for the device. For example, they want to turn their lights on. The button will send a request to the server indicating the device id and the desired state. The server will receive that request and send it to the graph processor. The processing feature will then utilize Dijkstra's shortest path algorithm to traverse the corresponding graph that is stored in the Mongo database. The processing system will have a queue of functions needed to be executed in the application/s to get the device into the desired state. Once we have a queue of functions, this will then be sent to the MQTT message broker. The MQTT message broker works by "subscribing" each device to a "topic". Using this idea, the MQTT message broker can "publish" each function needed to be executed to it's proper application "topic". The MQTT server will continue until the queue of function names is empty. At this point the device/s should be in the requested state and should then be sent back to update the GUI and database with the new state. The device library will really be a utility library to work with the server.

2.3 Dependency Handling

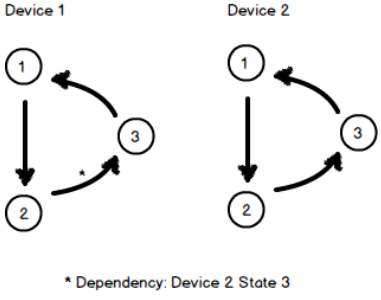


Figure 2

In Figure 2 represents two directed cyclic graphs. The current state of Device (application) 1 is state 1. The current state of Device (application) 2 is state 1 also. Let the transition between each state be fn12, fn23 and fn31. The transition on application 1 has a dependency that application 2 needs to be in state 3. When we run our dependency algorithm to find the needed actions to take we will end up with: [D1-fn12, D2-fn12, D2-fn23, D1-23]. This queue represents the function needed to be executed by the applications to be in the desired state.

2.4 Libraries & Methods

The server side application will be written in Node.js. Some of the important node dependencies include; a implementation of Dijkstra's algorithm, Mosca MQTT Node library, Express.js HTTP engine, and Mongoose for use with MongoDB. The directed cyclic graph processing feature class will be the major focus of this project. The feature will be mainly a series of available functions. Some of the feature that are to be included in this class are:

- Fetching of the graph matrix from the DB
- Computing the path Queue
- Computing the dependencies in the Queue
- Validating circular dependencies
- Validating DFSA graphs
- Returning the Queue for execution

3 Related Work

Particle.io

Particle.io has a very robust system that is incredibly impressive. Particle has a series of development platforms and products. One of the more impressive products they have is a cellular IoT SIM for cellular communication. They have various SDKs and IDE used to build the devices. [2]

Node.js

Node is a JavaScript runtime that was built on Chrome's V8 JavaScript engine. Node has a package ecosystem like many other languages. NPM is the largest ecosystem of open source libraries in the world. [4] Node is relevant to this project because it is what the server side application is being built on. The reasoning is that Node function extremely well as and HTTP request engine. It also functions very well as a chat engine. Due to the nature of the project and the similarities to HTTP requests and MQTT message brokers, developing with Node in this environment will work fine.

MQTT Protocol

MQTT is short for MQ Telemetry Transport. MQTT has a publish subscribe architecture. In the MQTT protocol the messages are delivered asynchronously ("pushed"). MQTT is an ideal protocol for constrained networks with low bandwidth, high latency, and data limit. MQTT is also useful for fragile connections. This is important because smart devices aren't meant to be just smart home devices. They can be field sensors in various parts of the world. MQTT has Quality of Service (QoS) levels which determine how each MQTT message get's delivered. [5] It has been accepted recently as a fairly standard tool to use in IoT platforms.

Amazon IoT

Similar to Particle, Amazon recently entered the field of IoT by purchasing 2lemetry and their platform that was create. They now offer an IoT platform as one of their web services.

ACKNOWLEDGMENT

REFERENCES

[1] By 2020, cross-industry devices will dominate the number of connected things used in the enterprise, "Gartner Says 6.4 Billion Connected," *"Things" Will Be in Use in 2016, Up 30 Percent From 2015*, 10-Nov-2015. [Online]. Available: http://www.gartner.com/newsroom/id/3165317. [Accessed: 01-Nov-2016].

[2] "Welcome New Members to the Forum! Resources, Projects, Tutorial Videos, and more," *Particle*. [Online]. Available: https://community.particle.io/. [Accessed: 01-Nov-2016].

[3] "AWS IoT - Amazon Web Services," *Amazon Web Services, Inc.* [Online]. Available: https://aws.amazon.com/iot/. [Accessed: 01-Nov-2016].

[4] N. Foundation, "Node.js", *Nodejs.org*, 2016. [Online]. Available: https://nodejs.org/en/. [Accessed: 03- Nov- 2016].

[5] R. Gupta, "5 Things to Know About MQTT – The Protocol for Internet of Things", *Ibm.com*, 2016. [Online]. Available: https://www.ibm.com/developerworks/community/blogs/5things/entry/5_things_to_know_about_mqtt_the_protocol_for_internet_of_things?lang=en. [Accessed: 03- Nov- 2016].

First A. Author

Second B. Author Jr.

Third C. Author