IFERP
connecting engineers... developing research

ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)
Vol 5, Issue 4, April 2018

Bridging for Cross Protocol Talk in IoT Devices using Windows Communication Foundation

[1] Rustom Mamlook, [2] Omer F. Khan, [3] Thabit Sultan Mohammed
Electrical and Computer Engineering Department, Dhofar University

Abstract: In the Internet of Things (IoT), multiple communication protocols are used to connect the smart device. Wi-Fi, Xbee, ZigBee, Bluetooth, and LoRaWAN are some of the communication channels utilized for connectivity by devices using some IoT platform. In order to enable the development of smart services for IoT platforms, there are solutions by different vendors to connect between IoT devices. For example, multiple IoT platforms are available in the market namely IoTivity platform developed by Open Connectivity Foundation (OCF), AllJoyn platform from All Seen Alliance, Weave made by Google, and Home Kit by Apple. In view of such segmentation of IoT platforms, IoT Application's development has been made complex, where IoT device and accompanying application compatibility with available platforms requires support for multiple protocols. To simplify the complexity introduced by multiple platforms, M2M [4] International standard was already proposed as the bridge for integrating IoT protocols. In our paper we implement a proxy web service using Windows Communication Foundation (WCF) as a way to translate communication in one IoT protocol to another. In our implementation of middleware, we allowed the MQTT broker to accept messages which were passed to the Web Service from various devices over Hyper Text Protocol's POST or GET Commands. Bridging between WCF Web Service and MQTT broker was enabled with duplex communication. Hence devices supporting either HTTP protocol or MQTT protocol were able to communicate transparently.

Index Terms— IOT.

I. INTRODUCTION

The internet of Things (IoT) and Web of Things (WoT) are the latest conventions of ICT (Information and Communication Technology) to bring disconnected devices together in an ecosystem where connectivity is the essence and internet is the communication backbone. Such connectivity gives rise to smart services such as smart city, smart grid, agricultural systems, smart health care and smart production. In order to reduce time to market for development of IoT devices and applications, a number of integrated IoT platforms were designed on top of existing web based protocols to enable safe and reliable connections among the IoT devices. The main goal of developing the existing web technologies for IoT devices was to reduce the overhead and latency of the networks they utilized. Therefore, new lightweight protocols were developed at the time for operation of IoT applications. This development at one end provided the solutions but on the other end, provided complexity of dealing with vendor based solutions with different IoT standards emerging, making it harder for developers to keep up with application of IoT in real life. In our paper, we propose to utilize web services (as a bridge) for all the communication protocols required between IoT devices.

II. METHODOLOGY

Since WoT and IoT closely resemble where networked devices are using web and internet respectively, the following bridging Scheme is suitable to glue together different protocols at the repository level. CRUDN [1] stands for create, read, update, delete and notify operations on the underlining database.

[MQTT Repository] ← CRUDN ← [Stored Procedures] ← [API Service Calls] ← [HTTP(S) Request (POST, GET)]

[IoTivity Repository] ← CRUDN ← [Stored Procedures] ← [API Service Calls] ← [HTTP(S) Request (POST, GET)]

[AllJoyn Repository] ← CRUDN ← [Stored Procedures] ← [API Service Calls] ← [HTTP(S) Request (POST, GET)]

[OM2M Repository] ← CRUDN ← [Stored Procedures] ← [API Service Calls] ← [HTTP(S) Request (POST, GET)]

[OpenMTC Repository] ← CRUDN ← [Stored Procedures] ← [API Service Calls] ← [HTTP(S) Request (POST, GET)]

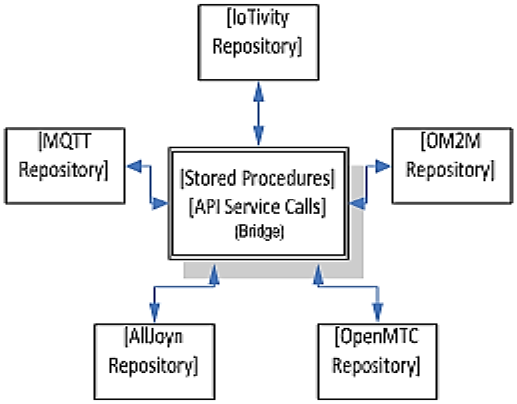
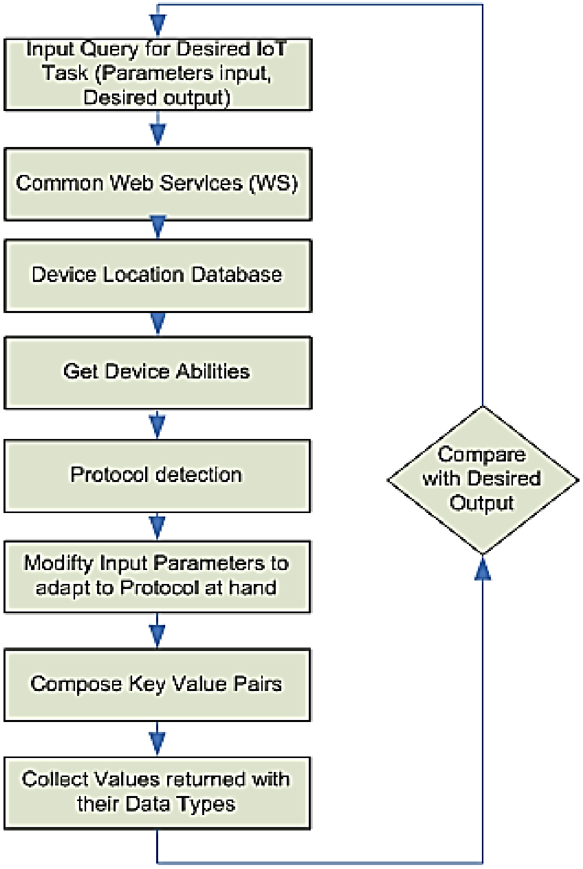
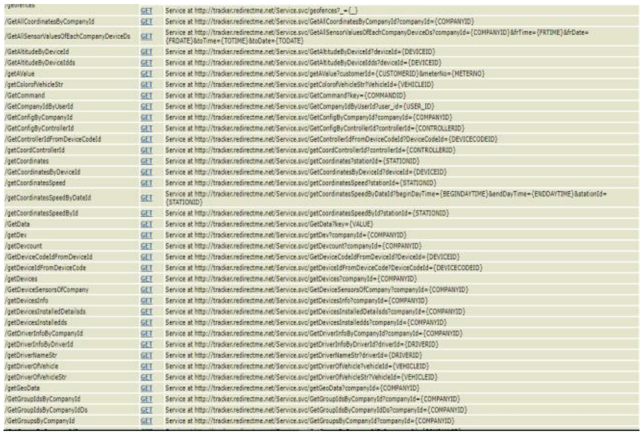


Figure 1: One Bridge-Multiple Protocols

Stored procedures provide transparent access to a common database. Stored procedures are invoked through web service calls using POST and GET commands. Each protocol has an element of HTTP requests that are made to the common database using stored procedure.


Input Query for Desired IoT Task (Parameters input, Desired output)
Common Web Services (WS)
Device Location Database
Get Device Abilities
Protocol detection
Modifty Input Parameters to adapt to Protocol at hand
Compose Key Value Pairs
Collect Values returned with their Data Types
Compare with Desired Output

However, a need exists for standardizing the APIs and device management interfaces in the light of existing communication frameworks. This will make the development of IoT applications focused on the presentation and processing the data being received by the devices. IoT application and device support many different protocols and data models in order to interconnect with each other on different platforms. In this paper, we propose to use the WCF standard as a middleware [2] for IoT protocol integration. We use MQTT [3] and WCF as the test cases for our protocol bridging design. Our contribution lies in demonstrating the possibilities of using only the WCF for IoT Protocols interworking using service operations by endpoints defined in Web Model. A WSDL document describes a web service. It specifies the location of the service, and the methods of the service, using major elements such as type, message, port type and binding. The request-response type allows the following operation, which shows that it is well suited for the IoT bridging Protocols. One-way The operation can receive a message but will not return a response Request-response The operation can receive a request and will return a response Solicit-response The operation can send a request and will wait for a response Notification The operation can send a message but will not wait for a response. We focus on utilizing Web API for data exchange and device management using browser based interface, across these platforms where standard specifications have already been defined in the Windows Communication Foundation. The format and presentation is standardized by using cascaded style sheets from a central repository to devices servicing them to clients. We explore and compare different device management possibilities in MQTT through WCF and propose such design for bridging.



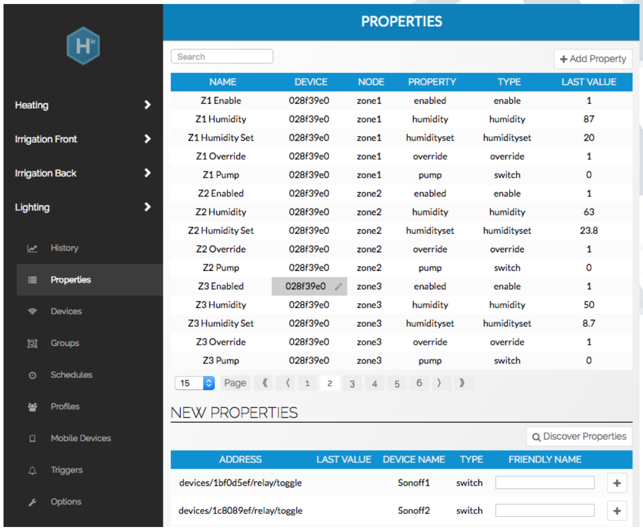
/geofences	GET	Service at http://tracker.redirectme.net/Service.svc/geofences?_={_}
/GetAllCoordinatesByCompanyId	GET	Service at http://tracker.redirectme.net/Service.svc/GetAllCoordinatesByCompanyId?companyId={COMPANYID}
/GetAllSensorValuesOfEachCompanyDeviceDs	GET	Service at http://tracker.redirectme.net/Service.svc/GetAllSensorValuesOfEachCompanyDeviceDs?companyId={COMPANYID}&frTime={FRTIME}&frDate={FRDATE}&toTime={TOTIME}&toDate={TODATE}
/GetAltitudeByDeviceId	GET	Service at http://tracker.redirectme.net/Service.svc/GetAltitudeByDeviceId?deviceId={DEVICEID}
/GetAltitudeByDeviceIdds	GET	Service at http://tracker.redirectme.net/Service.svc/GetAltitudeByDeviceIdds?deviceId={DEVICEID}
/getAValue	GET	Service at http://tracker.redirectme.net/Service.svc/getAValue?customerId={CUSTOMERID}&meterNo={METERNO}
/getColorofVehicleStr	GET	Service at http://tracker.redirectme.net/Service.svc/getColorofVehicleStr?VehicleId={VEHICLEID}
/GetCommand	GET	Service at http://tracker.redirectme.net/Service.svc/GetCommand?key={COMMANDID}
/GetCompanyIdByUserId	GET	Service at http://tracker.redirectme.net/Service.svc/GetCompanyIdByUserId?user_id={USER_ID}
/GetConfigByCompanyId	GET	Service at http://tracker.redirectme.net/Service.svc/GetConfigByCompanyId?companyId={COMPANYID}
/GetConfigByControllerId	GET	Service at http://tracker.redirectme.net/Service.svc/GetConfigByControllerId?controllerId={CONTROLLERID}
/GetControllerIdFromDeviceCodeId	GET	Service at http://tracker.redirectme.net/Service.svc/GetControllerIdFromDeviceCodeId?DeviceCodeId={DEVICECODEID}
/getCoordControllerId	GET	Service at http://tracker.redirectme.net/Service.svc/getCoordControllerId?controllerId={CONTROLLERID}
/getCoordinates	GET	Service at http://tracker.redirectme.net/Service.svc/getCoordinates?stationId={STATIONID}
/GetCoordinatesByDeviceId	GET	Service at http://tracker.redirectme.net/Service.svc/GetCoordinatesByDeviceId?deviceId={DEVICEID}
/getCoordinatesSpeed	GET	Service at http://tracker.redirectme.net/Service.svc/getCoordinatesSpeed?stationId={STATIONID}
/getCoordinatesSpeedByDateId	GET	Service at http://tracker.redirectme.net/Service.svc/getCoordinatesSpeedByDateId?beginDayTime={BEGINDAYTIME}&endDayTime={ENDDAYTIME}&stationId={STATIONID}
/getCoordinatesSpeedById	GET	Service at http://tracker.redirectme.net/Service.svc/getCoordinatesSpeedById?stationId={STATIONID}
/GetData	GET	Service at http://tracker.redirectme.net/Service.svc/GetData?key={VALUE}
/getDev	GET	Service at http://tracker.redirectme.net/Service.svc/getDev?companyId={COMPANYID}
/getDevcount	GET	Service at http://tracker.redirectme.net/Service.svc/getDevcount?companyId={COMPANYID}
/GetDeviceCodeIdFromDeviceId	GET	Service at http://tracker.redirectme.net/Service.svc/GetDeviceCodeIdFromDeviceId?DeviceId={DEVICEID}
/getDeviceIdFromDeviceCode	GET	Service at http://tracker.redirectme.net/Service.svc/getDeviceIdFromDeviceCode?DeviceCodeId={DEVICECODEID}
/getDevices	GET	Service at http://tracker.redirectme.net/Service.svc/getDevices?companyId={COMPANYID}
/GetDeviceSensorsOfCompany	GET	Service at http://tracker.redirectme.net/Service.svc/GetDeviceSensorsOfCompany?companyId={COMPANYID}
/getDevicesInfo	GET	Service at http://tracker.redirectme.net/Service.svc/getDevicesInfo?companyId={COMPANYID}
/getDevicesInstalledDetailsds	GET	Service at http://tracker.redirectme.net/Service.svc/getDevicesInstalledDetailsds?companyId={COMPANYID}
/getDevicesInstalledds	GET	Service at http://tracker.redirectme.net/Service.svc/getDevicesInstalledds?companyId={COMPANYID}
/GetDriverInfoByCompanyId	GET	Service at http://tracker.redirectme.net/Service.svc/GetDriverInfoByCompanyId?companyId={COMPANYID}
/getDriverInfoByDriverId	GET	Service at http://tracker.redirectme.net/Service.svc/getDriverInfoByDriverId?driverId={DRIVERID}
/getDriverNameStr	GET	Service at http://tracker.redirectme.net/Service.svc/getDriverNameStr?driverId={DRIVERID}
/getDriverOfVehicle	GET	Service at http://tracker.redirectme.net/Service.svc/getDriverOfVehicle?vehicleId={VEHICLEID}
/getDriverOfVehicleStr	GET	Service at http://tracker.redirectme.net/Service.svc/getDriverOfVehicleStr?VehicleId={VEHICLEID}
/getGeoData	GET	Service at http://tracker.redirectme.net/Service.svc/getGeoData?companyId={COMPANYID}
/GetGroupIdsByCompanyId	GET	Service at http://tracker.redirectme.net/Service.svc/GetGroupIdsByCompanyId?companyId={COMPANYID}
/GetGroupIdsByCompanyIdDs	GET	Service at http://tracker.redirectme.net/Service.svc/GetGroupIdsByCompanyIdDs?companyId={COMPANYID}
/GetGroupsByCompanyId	GET	Service at http://tracker.redirectme.net/Service.svc/GetGroupsByCompanyId?companyId={COMPANYID}

Figure 2: Web-Service Operations provide functions for an IoT Bridge

Because MQTT uses the publish and subscribe model with client identifier used for the registry to the MQTT broker, the discovery mechanisms to discover other devices is through the broker keeping a database of its register. A user interface is provided to let the users initiate the MQTT device discovery through the MQTT protocol.

The implementation of web service talking to the MQTT database allows performing operations directly to the devices registered at the broker.

Once an MQTT device has been discovered, its metadata will be shared with the clients connected using the web service. In this way a real-time polling of device mechanism is provided by implementing AJAX e.g. Signal-R). Asynchronous JavaScript + XML" is a set of Web development techniques using many Web technologies on the client side to create asynchronous Web applications. In our implementation, we use Arduino Mega as client's devices based on ESP8266 frameworks. We also implement an open source control panel (Homie-control – A high-level application for the IoT) [5] application to let the user control and read the states of the devices.



PROPERTIES

NAME	DEVICE	NODE	PROPERTY	TYPE	LAST VALUE
Z1 Enable	028f39e0	zone1	enabled	enable	1
Z1 Humidity	028f39e0	zone1	humidity	humidity	87
Z1 Humidity Set	028f39e0	zone1	humidityset	humidityset	20
Z1 Override	028f39e0	zone1	override	override	1
Z1 Pump	028f39e0	zone1	pump	switch	0
Z2 Enabled	028f39e0	zone2	enabled	enable	1
Z2 Humidity	028f39e0	zone2	humidity	humidity	63
Z2 Humidity Set	028f39e0	zone2	humidityset	humidityset	23.8
Z2 Override	028f39e0	zone2	override	override	1
Z2 Pump	028f39e0	zone2	pump	switch	0
Z3 Enabled	028f39e0	zone3	enabled	enable	1
Z3 Humidity	028f39e0	zone3	humidity	humidity	50
Z3 Humidity Set	028f39e0	zone3	humidityset	humidityset	8.7
Z3 Override	028f39e0	zone3	override	override	1
Z3 Pump	028f39e0	zone3	pump	switch	0

NEW PROPERTIES

ADDRESS	LAST VALUE	DEVICE NAME	TYPE	FRIENDLY NAME
devices/1bf0d5ef/relay/toggle		Sonoff1	switch	
devices/1c8089ef/relay/toggle		Sonoff2	switch	

Figure 3: Homie-IoT, implementation

III. CONCLUSION AND FUTURE WORK

In this paper, the WCF and WSDL standards are the bridging agents to make different IoT platforms integrate with each other. Different protocols such as IoTivity and AllJoyn, MQTT etc. can communicated with each other as a transparent. The Web Services Description Language (WSDL) is an XML-based syntax for presenting information. It is a definition language for structured description of the functions offered by a web service. The functions can be invoked by the devices anytime at any place without the need for constantly being connected to the IoT ecosystem. We implement and verify this integration architecture based on the existing Web Service interworking specifications. In particular, our paper also focuses on the running Device Management (DM) functions from different platform standards centrally available to a Web-based Graphical User Interface called a High-Level application for Homie IoT. Homie-control provides a web UI to manage Homie devices as well as a series of virtual python devices to allow extended functionality. In the future, we aspire to extend the integration of other IoT platforms such as Google Weave and Apple Home Kit. Furthermore, an integration of tiny encryption algorithms used to encrypt communication between different IoT data exchanges will be looked at in the future.

REFERENCES

[1] Jung Yeon Seo# , Dae Won Lee* , Hwa Min Lee#, Performance Comparison of CRUD Operations in IoT based Big Data Computing, in International Journal on Advanced Science Engineering Information Technology. Vol.7 (2017) No. 5 ISSN: 2088-5334

[2] Yi YANG∗, Zhiliang WANG, Quanbin LIU, Lu WANG, "Building a Pervasive SOA Based IOT Communication Middleware Using Runtime Compilation and Reflection", in the Journal of Computational Information Systems 8: 2 (2012) 643–654 available at http://www.jofcis.com/

[3] Urs Hunkeler & Hong Linh Truong, Andy Stanford-Clark, "MQTT-S – A Publish/Subscribe Protocol for Wireless Sensor Networks"

[4] Mohamed H. Elgazzar , Perspectives on M2M protocols A comparative study between different M2M protocols, 2015 IEEE Seventh International Conference on Intelligent Computing and Information Systems (ICICIS'15)

[5] A high-level-mobile-ready web application for the Homie-esp8266 IoT framework https:// github. com/stufisher /homie-control,