Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer e-ISSN: 2548-964X
Vol. 1, No. 12, Desember 2017, hlm. 1560-1567 http://j-ptiik.ub.ac.id

Pengembangan Iot Middleware Berbasis Event-Based dengan Protokol Komunikasi CoAP, MQTT dan Websocket

Husnul Anwari[1], Eko Sakti Pramukantoro[2], M. Hannats Hanafi[3]

Program Studi Teknik Informatika, Fakultas Ilmu Komputer, Universitas Brawijaya
Email: [1]husnulhamidiah@gmail.com, [2]eko.sakti@ub.ac.id, [3]hannats.hanafi@ub.ac.id

Abstrak

Kondisi Internet of Things (IoT) saat ini digambarkan seperti terjebak dalam silo. Artinya tiap komunikasi antar perangkat terbatas hanya pada satu domain, hal ini dikarenakan permasalahan pada *middleware* yang digunakan terbatas untuk protokol data sesuai perangkat tertentu atau dikenal dengan syntactical interoperability. Solusinya dari permasalahan tersebut adalah sebuah arsitektur *middleware* multi-protokol yang dapat digunakan oleh semua jenis perangkat. Penelitian ini dikembangkan sebuah midlleware untuk menjawab tantangan syntactical interoperability dengan tiga data protocol yaitu CoAP, MQTT dan Websocket. Dari hasil pengujian *middleware* yang dikembangkan memenuhi syarat interoperabilitas.

Kata kunci: *IoT, Middleware, Gateway, CoAP, MQTT, Websocket*

Abstract

The condition of the Internet of Things (IoT) is currently depicted as trapped in a silo. This means that each communication between devices is limited to one domain, this is due to problems at the middleware that is used is limited to protocol data according to specific devices, known as syntactical interoperability. The solution of the problem is a multi-protocol middleware architecture that can be used by all types of devices. This research developed a midlleware to answer the challenge of syntactical interoperability with three data protocol that is coap, mqtt and websocket. From the results of testing developed middleware eligible interoperability.

Keywords: *IoT, Middleware, Gateway, COAP, MQTT, Websockets*

1. PENDAHULUAN

Internet of Things merupakan perpaduan antara internet dan *ubiquitos computing*. IoT melibatkan interaksi antara beragam perangkat seperti sensor, agregator, aktuator dan aplikasi dalam berbagai macam domain. Pada dasarnya IoT terdiri dari dua komponen utama yakni internet dan *things.*

P. Desai (2015) menggambarkan permasalahan ini sebagai akibat model IoT saat ini dimana tiap domain terjebak dalam sebuah silo, artinya tiap domain memiliki perangkat sensor yang terhubung dan mengirimkan data ke *middleware* kemudian data tersebut dibaca oleh aplikasi melalui API tertentu. Hal ini menyebabkan terbatasnya interaksi dalam IoT yang seharusnya semua perangkat dapat berkomunikasi satu sama lain tanpa harus ada batasan di tiap domain. Selain itu apabila terdapat sensor jenis baru, ia harus terhubung dengan setiap *middleware* yang ada untuk bisa berkomunikasi antar domain. Dari segi aplikasi, developer juga harus mengimplementasikan setiap API dari tiap domain agar dapat membaca data dari perangkat sensor. Hal senada juga diungkapkan oleh Thomas Zachariah (2015) dalam penelitiannya yang menyebutkan bahwa *middleware* yang ada saat ini belum mampu mengatasi masalah interoperabilitas. Hal ini dibuktikan dengan *smartwatch* dan *smart light bulb* yang hanya dapat digunakan dengan *middleware* tertentu agar terhubung dengan internet.

Middleware lain yang ada saat ini juga belum sepenuhnya mendukung interoperabilitas. W. Zhiliang (2011) mengembangkan *middleware* berbasis SOA yang berfokus pada kinerja. *Middleware* ini dapat meng-*handle* hingga 60 *request* per detik, akan tetapi tidak ada keterangan bagaimana *middleware* ini mengatasi masalah interoperabilitas. D. Conzon (2012) mengembangkan VIRTUS *Middleware*, sebuah *middleware* berbasis XMPP yang berfokus pada

fitur keamanan. Penelitian lain yaitu QEST oleh M. Collina (2012), *middleware* ini berperan sebagai *bridge* untuk protokol MQTT dan REST (*Representational State Transfer*). J. Boman (2014) mengembangkan sebuah *middleware* berbasis Global Sensor Network (GSN), akan tetapi kekurangan dari GSN adalah sensor yang dapat digunakan hanyalah sensor dengan standar IEEE 1451. Aleksandar Antonić (2015) mengembangkan *middleware* berbasis *cloud* dan menggunakan pola *publish/subscribe* yang disebut CUPUS. Sama halnya dengan penelitian sebelumnya, *middleware* ini hanya dapat digunakan oleh sensor tertentu.

Untuk mengatasi masalah interoperabilitas dibutuhkan sebuah *middleware* yang mampu mendukung interoperabilitas. Mohamed (2012), Fersi (2015) dan M. A. Razzaque (2016) mengartikan interoperabilitas sebagai kemampuan *middleware* untuk dapat digunakan oleh beragam jenis perangkat/teknologi/ aplikasi sebagai perantara untuk saling berkomunikasi dan bertukar data.

Dari pembahasan sebelumnya maka diusulkan sebuah *middleware* dengan pendekatan *event-based* yang mampu mendukung interoperabilitas berbagai macam perangkat atau sensor. Dari hasil pengujian *middleware* yang diusulkan mampu mengatasi masalah *syntatic interoperability* dengan menyediakan *gateway* untuk berkomunikasi dengan perangkat sensor IoT menggunakan protokol MQTT dan CoAP, serta mampu berkomunikasi dengan aplikasi lain menggunakan protokol Websocket.

2. KAJIAN PUSTAKA

Penelitian pertama yakni "*Challenges in Middleware Solutions for the Internet of Things*" oleh Mohamed (2012). Penelitian ini bertujuan untuk merumuskan tantangan-tantangan yang harus dihadapi oleh *middleware* dalam IoT. Penelitian ini juga membahas beberapa *middleware* dari tiga domain yakni *semanctic web and web services*, *sensor network and RFID* dan *robotics*. Dalam penelitian ini disebutkan bahwa tantangan yang harus dihadapi oleh *middleware* adalah *interoperability*, *scalability*, *device abstraction*, *spontaneous interaction*, *unfixed infrastructure*, *multiplicity* serta *security and privacy*. Pada penelitian in disebutkan bahwa interoperabilitas menjadi tantangan paling utama dari *middleware* dikarenakan keberagaman perangkat yang akan berinteraksi dalam IoT, baik yang sudah ada maupun yang akan ditemukan di masa mendatang.

M. A. Razzaque (2016) juga melakukan penelitian yang sama dan lebih mendalam tentang kebutuhan yang harus ada ketika mengembangkan *middleware* untuk IoT. Peneliti membagi kebutuhan ini menjadi tiga bagian, pertama yakni kebutuhan fungsional meliputi *resource discovery*, *resource management*, *data management*, *event menagement* dan *code management*. Kebutuhan fungsional ini mengharuskan sebuah *middleware* untuk dapat digunakan oleh beragam jenis perangkat dengan mudah tanpa harus memperhatikan teknologi dan data model yang digunakan oleh perangkat tersebut. Kebutuhan berikutnya yakni kebutuhan non-fungsional seperti *scalability*, *real time*, *reliability*, *availability*, *security and privacy*, *ease-of-deployment*, dan *popularity*. Terakhir, yakni kebutuhan arsitektural meliputi *interoperability*, *service-based*, *adaptive*, *context-awareness* dan *autonomous behavior*. Penelitian ini mendukung penelitian sebelumnya yakni salah satu tantangan *middleware* yang harus diselesaikan adalah interoperabilitas.

Thomas Zachariah (2015) menyebutkan salah satu masalah IoT saat ini terletak pada bagian *gateway*. Peneliti menjelaskan bahwa kondisi yang ada saat ini adalah sebuah *middleware* (*gateway* dan aplikasi) hanya dapat digunakan oleh satu jenis perangkat tertentu. Sebagai contoh yakni *smartwatch* seperti Apple Watch, Fitbit, Moto 360 membutuhkan aplikasi tertentu sebagai *gateway* untuk terhubung dengan internet. Hal yang sama juga terjadi pada *smart light bulb* seperti *Philips Hue* yang membutuhkan perangkat dan aplikasi tertentu untuk beroperasi. Untuk mengatasi masalah ini, peneliti menawarkan sebuah konsep arsitektur *gateway* pada *smartphone* berbasis *Bluetooth Low Energy* (BLE) dan IPv6.

Hal senada juga diungkapkan oleh P. Desai (2015) dalam penelitiannya berjudul *Semantic Gateway as a Service architecture for IoT Interoperability* yang menyebutkan bahwa kondisi IoT saat ini terbatas dalam silo. Peneliti mengungkapkan bahwa interaksi pada IoT terbatas hanya pada satu domain dan tak ada komunikasi antar domain. Hal ini dikarenakan permasalahan pada *middleware* yang digunakan terbatas untuk perangkat tertentu. Dalam penelitian ini P. Desai (2015) mengusulkan sebuah arsitektur *middleware* multi-protokol

yang dapat digunakan oleh semua jenis perangkat. Arsitektur *middleware* ini mendukung komunikasi perangkat dengan protokol CoAP, MQTT dan XMPP.

Beberapa survei juga telah dilakukan untuk mengetahui apa saja jenis *middleware* yang ada saat ini dan bagaimana kinerja tiap *middleware* dalam mengatasi tantangan di atas. (Anne, et al., 2016) membagi *middleware* menjadi tiga jenis yakni *Service-based middleware*, *Cloud-based middleware*, dan *Actor-based middleware.*

telah melakukan penelitian terkait metode pengujian interoperabilitas sebuah sistem. Penelitian ini membahas semua metode yang sudah ada dan membandingkannya dilihat dari seberapa luas cakupan interoperabilitas yang diuji. Untuk mengevaluasi metode pengujian yang ada, pertama peneliti menyebutkan cakupan interoperabilitas berdasarkan hasil penelitian oleh S. Koussouris (2011). Cakupan interoperabilitas ini dibagi menjadi 4 level. Level pertama meliputi Data *interoperability*, *Process interoperability*, *Rules interoperability*, *Object interoperability*, *Software system interoperability* dan *Cultural interoperability*. Level kedua meliputi *Knowledge interoperability*, *Services interoperability*, *Social network interoperability*, *Electornic interoperability*. Level ketiga yakni *Cloud interoperability* sedangkan level terakhir yakni *Ecosystem interoperability*.

Dari penelitian yang dilakukan, peneliti mengambil kesimpulan bahwa semua metode pengujian yang ada saat ini sudah mencakup level 1 dan level 2 dari interoperabilitas tetapi tidak untuk level 3 dan 4.

3. PERANCANGAN SISTEM

Tujuan utama dari *middleware* yang dikembangkan adalah menyediakan *gateway* multi-protokol bagi sensor untuk mengirimkan data melalui protokol CoAP atau MQTT. Tujuan berikutnya yakni menyediakan *gateway* bagi aplikasi untuk membaca data yang dikirimkan sensor melalui protokol Websocket. *Middleware* ini berguna bagi developer untuk fokus pada aplikasi ataupun perangkat IoT yang dikembangkan tanpa harus melihat protokol yang digunakan.

3.1. Kebutuhan Fungsional

Kebutuhan fungsional adalah kebutuhan-kebutuhan yang harus dipenuhi dan proses-proses yang dapat dilakukan oleh sistem. Kebutuhan fungsional sistem dalam penelitian ini dijelaskan pada tabel berikut :

Tabel 1. Daftar Kebutuhan Fungsional

No	Kebutuhan Fungsional
1	Raspbery Pi dapat menyediakan layanan acess point bagi perangkat sensor
2	Perangkat sensor dapat terhubung dengan Access point dari Raspbery Pi sehingga pertukaran data dapat terjadi
3	Perangkat sensor dapat mengirimkan data suhu dan kelembapan ke *middleware* melalui protokol CoAP
4	Perangkat sensor dapat mengirimkan data suhu dan kelembapan ke *middleware* melalui protokol MQTT
5	*Middleware* dapat menerima data suhu dan kelembapan dari sensor melalui protokol CoAP, MQTT serta CoAP dan MQTT.
6	*Middleware* dapat mengirimkan data suhu dan kelembapan dari sensor dengan protokol CoAP, MQTT serta CoAP dan MQTT ke aplikasi web melalui protokol Websocket

3.2. Kebutuhan Non-Fungsional

Kebutuhan non-fungsional adalah kebutuhan yang menitikberatkan pada perilaku dan batasan fungsi pada sistem. Dalam penelitian ini kebutuhan non-fungsional terdiri dari kebutuhan perangkat keras (tabel 2) dan kebutuhan perangkat lunak (tabel 3).

Tabel 2. Daftar Kebutuhan Perangkat Keras

Perangkat	Keterangan
Raspberry Pi	Raspberry Pi digunakan sebagai perangkat untuk menjalankan *middleware* dan juga sebagai Access point.
USB Adapter TL-WN722N – TP-Link	Perangkat ini dibutuhkan sebagai pedukung Raspberry Pi agar dapat digunakan sebagai Access point.
ESP8266	Perangkat ini digunakan sebagai mikrokontroler bagi sensor suhu dan kelembapan yang dibuat. Mikrokontroler ini nantinya dirangkai dengan modul DHT11/DHT22 dan diprogram sehingga dapat mengirimkan data suhu dan kelembapan melalui protokol CoAP dan MQTT.
DHT11 dan DHT22	Modul sensor untuk membaca suhu dan kelembapan.
Server Cloud (Ubuntu instance)	Perangkat ini digunakan untuk menjalankan aplikasi web sebagai *subscribe*r yang membaca data dari *middleware* melalui protokol Websocket.
Lenovo G40-70 Core i3	Perangkat ini digunakan untuk mengakses aplikasi web untuk menampilkan data dari *middleware*.

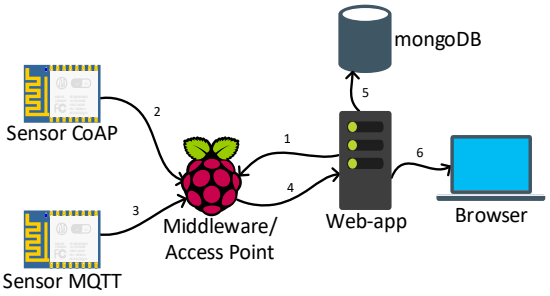
Tabel 3. Daftar Kebutuhan Perangkat Lunak

Perangkat	Keterangan

Raspibian Jessie	Operating system untuk Raspberry Pi
Redis	Media penyimpanan data dalam memori yang juga bertindak sebagai broker
Node.js	Framework dengan berbasis event-driven dan asynchronus untuk mengembangkan *middleware*.
NodeMCU Firmware	Firmware berbasis LUA untuk mikrokontroler ESP8266
ESPlorer	IDE untuk memprogram ESP8266 menggunakan LUA
MongoDB	Basis data No-SQL untuk menyimpan data dari sensor

3.3. Perancangan Alur Komunikasi

Pada penelitian ini dibangun sebuah sistem sebagai lingkungan pengujian dari *middleware* yang dikembangkan. Sistem ini terdiri dari dua perangkat sensor yang akan mengirimkan data ke *middleware* melalui protokol CoAP dan MQTT, *middleware* multi-protokol sebagai broker serta aplikasi web yang akan menyimpan dan menampilkan data tersebut secara real-time. Perancangan alur sistem dapat dilihat pada gambar di bawah ini :



Gambar 1. Rancangan sistem

Alur sistem yang ada pada gambar di atas dijelaskan sebagai berikut :

1. Aplikasi *subscribe* topik home/kitchen dan home/garage ke *middleware*
2. Sensor CoAP mengirimkan data suhu dan kelembapan dengan topik home/kitchen setiap 30 detik sekali.
3. Sensor MQTT mengirimkan data suhu dan kelembapan dengan topik home/garage setiap 30 detik sekali.
4. Aplikasi menerima data suhu dan kelembapan untuk topik home/kitchen dan home/garage yang dikirimkan oleh sensor
5. Aplikasi menyimpan data tersebut pada database MongoDB.
6. Aplikasi diakses dengan browser.

Dalam sistem ini terdapat dua jenis sensor yang digunakan yakni sensor berbasis CoAP dan sensor berbasis MQTT. Sensor CoAP akan mengirimkan data suhu dan kelembapan ke *middleware* dengan topik home/kitchen dengan mengirimkan POST request ke midleware. Sedangkan sensor MQTT akan mengirimkan data suhu dan kelembapan dengan topik home/garage dengan melakukan *publish* ke *middleware*. Sebelum sensor dapat mengirimkan data, masing-masing sensor harus terhubung dengan Access point dari Raspberry Pi terlebih dahulu. Pengiriman data dari perangkat sensor pada penelitian ini dilakukan tiap 30 detik untuk mendapatkan kuantitas data dari sensor guna tahap pengujian.

Di sisi lain, setiap kali sensor mengirimkan data, *middleware* akan mengirimkan data tersebut ke aplikasi web secara real-time menggunakan protokol Websocket. Sebelum aplikasi web dapat menerima data, aplikasi web harus terhubung terlebih dahulu dengan *middleware* melalui jaringan dan *subscribe* pada topik yang dikirimkan oleh sensor. Setelah data diterima oleh aplikasi web, data tersebut selanjutnya disimpan dalam basis data MongoDB.

4. IMPLEMENTASI

Bagian ini menjelaskan bagaimana *middleware* dikembangkan dari dasar. Dimulai dari instalasi Redis, instalasi framewrok Node.js, hingga implementasi tiap komponen *middleware* (sensor *gateway*, service unit, dan application *gateway*).

4.1. Instalasi Redis

Ada beberapa opsi untuk menginstall Redis, akan tetapi pada penelitian kali ini penulis menginstall Redis pada mesin yang sama dengan *middleware*. Selain mengurangi jumlah perangkat yang dibutuhkan, transaksi dengan Redis akan lebih cepat karena diakses melalui localhost.

4.2. Instalasi Framework Node.js

Sesuai dengan spesifikasi kebutuhan *middleware* pada bab sebelumnya, bahwa *middleware* ini dikembangkan menggunakan *framework* Node.js. Oleh karena itu hal mendasar yang harus dilakukan sebelum melangkah lebih jauh yaitu menginstall Node.js. Sama seperti Redis, untuk mengisntall Node.js diperlukan Raspberry Pi dengan OS berbasis ARM yang didukung oleh Node.js. Oleh karena Redis dan Node.js berada dalam satu mesin, maka OS yang digunakan kali ini juga sama

yakni Raspibian Jessie. Dalam penelitian kali ini Node.js yang digunakan adalah versi 6.9.2.

4.3. Implementasi *Middleware*

Implementasi *middleware* dibagi menjadi beberapa bagian. Pertama yakni inplementasi service unit. Service unit mempunyai 3 fungsi utama yakni menyediakan *data management*, *service delivery* dan *interface definition*. Masing-masing fungsi inilah yang akan dijelaskan lebih lanjut. Fungsi pertama yakni *data management*, untuk hal ini Service layer mempunyai sebuah abstraksi tipe data yang akan disimpan dengan beberapa *accessors* dan *mutator method*. *Service delivery* dan *interface definition* di implementasikan dengan membuat beberapa fungsi seperti Save, Find, FindOrCreate, *Publish*, *Subscribe*, dan Uns*ubscribe* yang nantinya berperan sebagai API bagi *gateway* untuk berinteraksi dengan Redis. *Pseudo-code* di bawah ini menjelaskan fungsi Save. Setelah nilai key dan value berhasil disimpan di Redis, selanjutnya key dan value tersebut di *publish*.

Berikutnya implementasi CoAP dan MQTT *gateway*. CoAP *gateway* menyediakan *interface* bagi sensor untuk mengirimkan data ke *middleware*. Sensor mengirimkan data dengan melakukan POST *request* ke *middleware*. Oleh karena itu dalam CoAP *gateway* diimplementasikan sebuah fungsi untuk meng-*handle request* tersebut.

Ketika *request* diterima, CoAP *gateway* melakukan validasi URL yang digunakan, apabila tidak valid, proses dihentikan dan kode 4.05 (*no permission*) dikirimkan ke sensor. Sebaliknya, apabila valid proses berikutnya yakni memanggil fungsi findOrCreate untuk menyimpan data yang dikirimkan ke Redis. Terakhir yakni mengirim kode 2.01 (*created*) sebagai tanda bahwa data sudah berhasil tersimpan.

Berbeda dengan CoAP *gateway*, MQTT *gateway* menggunakan paradigma *event-driven*, artinya sebuah fungsi akan dijalankan ketika ada *event* tertentu. Misalkan ketika sensor mengirimkan data, *event publish* terjadi dan fungsi pada *event* tersebut akan dijalankan. Event yang ada pada MQTT *gateway* yakni *connect*, *publish*, *disconnect*, *error* dan *close*.

Implementasi terakhir yakni Application *gateway*. *Appication gateway* juga menggunakan paradigma *event-driven*, sama seperti MQTT *gateway*. *Event* yang ada pada *application gateway* yakni connection, *subscribe* dan disconnect. *Event* connection terjadi ketika terdapat aplikasi yang terhubung dengan *middleware* melalui Websocket. *Event* berikutnya yakni *subscribe*, *event* ini menjadi *interface* bagi aplikasi untuk membaca data dari sensor. Ketika event ini terjadi pertama *application gateway* melakukan *subscribe* ke Redis untuk topik yang diminta oleh aplikasi dan kedua mencari nilai dari topik tersebut dengan fungsi find. Jika ditemukan, data tersebut akan dikirimkan ke aplikasi. Hal ini sangat berguna karena setiap kali aplikasi melakukan *subscribe* ia akan menerima data saat itu juga meskipun sensor belum mengirimkan data untuk topik yang diminta. Apabila data untuk topik tersebut tidak ada, maka *application gateway* akan menunggu hingga sensor mengirimkan data kemudian meneruskannya ke aplikasi dengan emit. *Event* selanjutnya yakni disconnect yang terjadi ketika koneksi antara aplikasi dan *middleware* terputus. Fungsi pada *event* ini akan melakukan uns*ubscribe* topik dari aplikasi yang terputus.

4.4. Implementasi Sensor

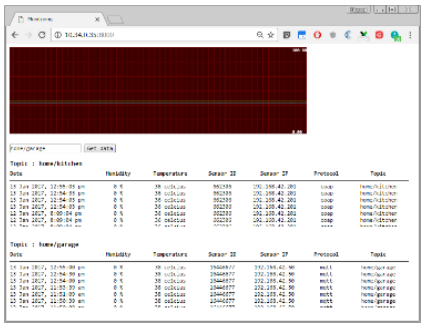
Sebelum melakukan pengujian *middleware*, ada beberapa hal yang perlu dipersiapkan, salah satunya adalah perangkat sensor. Pada penelitian ini dibutuhkan dua sensor yang mengimplementasikan protokol CoAP dan MQTT. Sesuai dengan analisis kebutuhan, sensor ini akan dibangun menggunakan mikrokontroler NodeMCU (ESP8266) dan modul DHT11 (sensor suhu dan kelembapan). Hasil dari rangkaian sensor dapat dilihat pada gambar ini :

Gambar 2. Rangkaian sensor suhu dan kelembapan

4.5. Implementasi Aplikasi Web

Aplikasi web dikembangkan menggunakan framework Node.js pada server dan HTML untuk web yang ditampilkan. Implementasi aplikasi in dimulai dengan membuat kode program untuk server dimana harus dapat terhubung dan membaca data dari *middleware*

melalui protokol Websocket sesuai dengan bagian perancangan. Selanjutnya membuat web yang ditampilkan pada browser menggunakan HTML5, CSS dan Javascript. Hasil akhir Implementasi dari aplikasi web ini dapat dilihat pada gambar di bawah ini :



Gambar 3. Aplikasi web

5. PENGUJIAN DAN ANALISIS

Setelah proses implementasi selesai, langkah berikutnya yakni melakukan pengujian pada *middleware*. Pengujian ini dibutuhkan untuk mengetahui apakah kinerja dari *middleware* sudah sesuai dengan kebutuhan. Pengujian ini dikatakan berhasil apabila tidak ada hasil pengujian yang menunjukkan kesalahan. Penjelasan dan kesesuaian hasil pengujian dengan kebutuhan sistem dijelaskan di bawah ini :

1. Pengiriman pesan dari protokol CoAP (NON)

   Pengujian ini dilakukan dengan client A melakukan *subscribe* untuk topik foobar melalui protokol Websocket. Client B *publish* ke topik foobar dengan data {foo: bar} melalui protokol CoAP (Non-Confirmable). Selanjutnya client A harus menerima data yang sama yakni {foo: bar} melalui protokol Websocket.

```
Scenario: [MD_001] Receiving message from CoAP (NON)
✔ Given client "A" subscribe to "foobar" via WEBSOCKET
✔ When client "B" publishes "{'foo':'bar'}" to "foobar" via COAP with QoS 0
✔ Then client "A" should have received "{'foo':'bar'}" from "foobar" via WEBSOCKET
```

Gambar 4. Hasil pengujian 1

2. Pengiriman pesan dari protokol CoAP (CON)

   Pengujian ini dilakukan dengan client A melakukan *subscribe* untuk topik foobar melalui protokol Websocket. Client B *publish* ke topik foobar dengan data {foo: bar} melalui protokol CoAP (Confirmable). Selanjutnya client A harus menerima data yang sama yakni {foo: bar} melalui protokol Websocket.

```
Scenario: [MD_002]Receiving message from CoAP (CON)
✔ Given client "A" subscribe to "foobar" via WEBSOCKET
✔ When client "B" publishes "{'foo':'bar'}" to "foobar" via COAP with QoS 1
✔ Then client "A" should have received "{'foo':'bar'}" from "foobar" via WEBSOCKET
```

Gambar 5. Hasil pengujian 2

3. Pengiriman pesan dari protokol CoAP (CON) dengan dua *subscribe*r

   Pengujian ini dilakukan dengan client A dan client C melakukan *subscribe* untuk topik foobar melalui protokol Websocket. Client B *publish* ke topik foobar dengan data {foo: bar} melalui protokol CoAP (Confirmable). Selanjutnya client A dan client C harus menerima data yang sama yakni {foo: bar} melalui protokol Websocket.

```
Scenario: [MD_003] Receiving message from CoAP (CON) packet with two applications
✔ Given client "A" subscribe to "foobar" via WEBSOCKET
✔ And client "C" subscribe to "foobar" via WEBSOCKET
✔ When client "B" publishes "{'foo':'bar'}" to "foobar" via COAP with QoS 1
✔ Then client "A" should have received "{'foo':'bar'}" from "foobar" via WEBSOCKET
✔ And client "C" should have received "{'foo':'bar'}" from "foobar" via WEBSOCKET
```

Gambar 6. Hasil pengujian 3

4. Pengiriman pesan dari protokol MQTT (QoS 0)

   Pengujian ini dilakukan dengan client A melakukan *subscribe* untuk topik foobar melalui protokol Websocket. Client B *publish* ke topik foobar dengan data {foo: bar} melalui protokol MQTT (QoS 0). Selanjutnya client A harus menerima data yang sama yakni {foo: bar} melalui protokol Websocket.

```
Scenario: [MD_004] Receiving message from MQTT (QoS 0)
✔ Given client "A" subscribe to "foobar" via WEBSOCKET
✔ When client "B" publishes "{'foo':'bar'}" to "foobar" via MQTT with QoS 0
✔ Then client "A" should have received "{'foo':'bar'}" from "foobar" via WEBSOCKET
```

Gambar 7. Hasil pengujian 4

5. Pengiriman pesan dari protokol MQTT (QoS 1)

   Pengujian ini dilakukan dengan client A melakukan *subscribe* untuk topik foobar melalui protokol Websocket. Client B *publish* ke topik foobar dengan data {foo: bar} melalui protokol MQTT (QoS 1). Selanjutnya client A harus menerima data yang sama yakni {foo: bar} melalui protokol Websocket.

```
Scenario: [MD_005] Receiving message from MQTT (QoS 1)
✔ Given client "A" subscribe to "foobar" via WEBSOCKET
✔ When client "B" publishes "{'foo':'bar'}" to "foobar" via MQTT with QoS 1
✔ Then client "A" should have received "{'foo':'bar'}" from "foobar" via WEBSOCKET
```

Gambar 8. Hasil pengujian 5

6. Pengiriman pesan dari protokol MQTT (QoS 2)

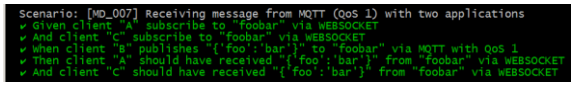
   Pengujian ini dilakukan dengan client A melakukan *subscribe* untuk topik foobar melalui protokol Websocket. Client B *publish* ke topik foobar dengan data {foo: bar} melalui protokol MQTT (QoS 0). Selanjutnya client A harus menerima data yang sama yakni {foo: bar} melalui protokol Websocket.

```
Scenario: [MD_006] Receiving message from MQTT (QoS 2)
✔ Given client "A" subscribe to "foobar" via WEBSOCKET
✔ When client "B" publishes "{'foo':'bar'}" to "foobar" via MQTT with QoS 2
✔ Then client "A" should have received "{'foo':'bar'}" from "foobar" via WEBSOCKET
```

Gambar 9. Hasil pengujian 6

7. Pengiriman pesan dari protokol MQTT (QoS 1) dengan dua *subscribe*r

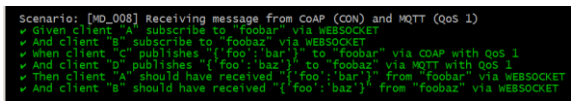
Pengujian ini dilakukan dengan client A dan client C melakukan *subscribe* untuk topik foobar melalui protokol Websocket. Client B *publish* ke topik foobar dengan data {foo: bar} melalui protokol MQTT (QoS 1). Selanjutnya client A dan client C harus menerima data yang sama yakni {foo: bar} melalui protokol Websocket.



Gambar 10. Hasil pengujian 7

8. Pengiriman pesan dari protokol CoAP (CON) dan MQTT (QoS 1)

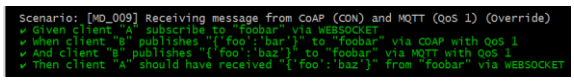
Pengujian ini dilakukan dengan client A melakukan *subscribe* untuk topik foobar dan client C melakukan *subscribe* untuk topik foobaz melalui protokol Websocket. Client C *publish* ke topik foobar melalui protokol CoAP (CON) dan Client D *publish* ke topik foobaz melalui protokol MQTT (QoS 1) dengan data {foo: bar}. Selanjutnya client A harus menerima data yang sama yakni {foo: bar} dari topik foobar dan client B harus menerima data yang sama yakni {foo: baz} dari topik foobaz melalui protokol Websocket.



Gambar 11. Hasil pengujian 8

9. Pengiriman pesan dari protokol CoAP (CON) dan MQTT (QoS 1). Pengujian override data pada Redis.

Pengujian ini dilakukan dengan client A melakukan *subscribe* untuk topik foobar melalui protokol Websocket. Client B *publish* ke topik foobar dengan data {foo: bar} melalui protokol CoAP (CON). Selanjutnya Client B *publish* ke topik foobar dengan data {foo: baz} melalui protokol MQTT (QoS 1). Client A harus menerima data terakhir yang dikirimkan untuk topik foobar yakni {foo: baz} melalui protokol Websocket.



Gambar 12. Hasil pengujian 9

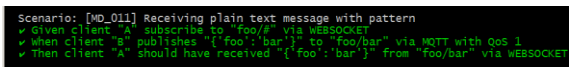
10. *Subscribe* topik menggunakan pattern *

Pengujian ini dilakukan dengan client A melakukan *subscribe* untuk topik foo/* melalui protokol Websocket. Client B *publish* ke topik foo/bar dengan data {foo: bar} melalui protokol MQTT (QoS 1). Selanjutnya client A harus menerima data yang sama yakni {foo: bar} melalui protokol Websocket.

Gambar 13. Hasil pengujian 10

11. *Subscribe* topik menggunakan pattern #

Pengujian ini dilakukan dengan client A melakukan *subscribe* untuk topik foo/# melalui protokol Websocket. Client B *publish* ke topik foo/bar dengan data {foo: bar} melalui protokol MQTT (QoS 1). Selanjutnya client A harus menerima data yang sama yakni {foo: bar} melalui protokol Websocket.



Gambar 14. Hasil pengujian 11

6. KESIMPULAN

Berdasarkan hasil perancangan, implementasi dan pengujian yang telah dilakukan didapatkan kesimpulan sebagai berikut; Implementasi IoT *middleware* dengan pendekatan *event-driven* dapat dilakukan dengan menerapkan arsitektur ends-to-*middleware* dan pola *publish/subscribe*. Dalam hal ini *middleware* bertindak sebagai *gateway* multi-protokol, sensor bertindak sebagai *publisher* yang mengirimkan data dan di sisi lain, sebuah aplikasi bertindak sebagai subscriber untuk membaca data yang dikirimkan sensor. Komunikasi atara protokol CoAP, MQTT dan Websocket pada *middleware* dapat dicapai dengan membuat *gateway* untuk masing-masing protokol, kemudian di hubungkan dengan sebuah broker.

7. DAFTAR PUSTAKA

Ala Al-Fuqaha, M. A. (2015). Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications. *IEEE Communication Surveys & Tutorials*, Vol. 17, No. 4.

Aleksandar Antonić, M. M. (2015, December). A Mobile Crowd Sensing Ecosystem Enabled by CUPUS: Cloud-based Publish/Subscribe Middleware for the Internet of Things. *Future Generation*

Computer Systems.

Anne H. H. Ngu, M. G. (2016, March). IoT Middleware: A Survey on Issues and Enabling Technologies. *IEEE Internet of Things Journal*, Vol. X, No. X.

Champaneria, H. B. (2016). Internet of things: Survey and case studies. 2015 *International Conference on Electrical, Electronics, Signals, Communication and Optimization (EESCO)*, 1-6. doi:10.1109/EESCO.2015.7253713

D. Blaauw, P. D.-V. (2012). *The Terraswarm Research Center (TSRC): White Paper*.

D. Conzon, T. B. (2012). The VIRTUS Middleware: An XMPP Based Architecture for Secure IoT Communications. *2012 21st International Conference on Computer Communications and Networks (ICCCN)*, 1-6.

Dinesh Thangavel, X. M.-X.-Y. (2014). *Performance Evaluation of MQTT and CoAP Performance Evaluation of MQTT and CoAP*. Singapore: National University of Singapore.

Fersi, G. (2015). *Middleware* for Internet of Things: A Study. *2015 International Conference on Distributed Computing in Sensor Systems*.

HiveMQ. (2015). *MQTT Essential Part 2 : Publish & Subscribe*. Dipetik February 28, 2016, dari HiveMQ: http://www.hivemq.com/blog/mqtt-essential-part2-publish-subscribe

HiveMQ. (2015). *MQTT Essential Part 4: MQTT Publish, Subscribe & Unsubscribe*. Dipetik February 28, 2016, dari HiveMQ: http://www.hivemq.com/blog/mqtt-essential-part-4-mqtt-publish-subscribe-unsubscribe

IERC, E. R. (2015). *IoT Semantic Interoperability: Research Challenges, Best Practices, Recommendations and Next Steps*.

IETF, I. E. (2016). *The Constrained Application Protocol (CoAP) RFC7252*. Internet Engineering Task Force.

J. Boman, J. T. (2014). Flexible IoT middleware for integration of things and applications. *10th IEEE International Conference on Collaborative Computing: Networking, Applications and Worksharing*, 481-488.

M. A. Razzaque, M. M.-J. (2016, February). Middleware for Internet of Things: A Survey. *EEE Internet of Things Journal*, vol. 3, no. 1, 70-95. doi:10.1109/JIOT.2015.2498900

M. Collina, G. E.-C. (2012). Introducing the QEST broker: Scaling the IoT by bridging MQTT and REST. *2012 IEEE 23rd International Symposium on Personal, Indoor and Mobile Radio Communications - (PIMRC)*, 36-41. doi:10.1109/PIMRC.2012.6362813

Mohamed, M. A. (2012, May). Challenges in Middleware Solutions for the Internet of Things. *2012 International Conference on Collaboration Technologies and Systems (CTS)*, 21-26. doi:10.1109/CTS.2012.6261022

Oasis. (2014). *MQTT Version 3.1.1*.

P. Desai, A. S. (2015). Semantic Gateway as a Service Architecture for IoT Interoperability. *2015 IEEE International Conference on Mobile Services*, 313-319. doi:10.1109/MobServ.2015.51

Reza Rezaei, T. K. (2014, January). *Interoperability evaluation models: A systematic review. Computers in Industry*, 65(1), 1-23. doi:10.1016/j.compind.2013.09.001

Thomas Zachariah, N. K. (2015, February). The Internet of Things Has a Gateway Problem. *In Proceedings of the 16th International Workshop on Mobile Computing Systems and Applications (HotMobile '15)*, 27-32. doi:10.1145/2699343.2699344

W. Zhiliang, Y. Y. (2011). A SOA Based IOT Communication Middleware. *2011 International Conference on Mechatronic Science, Electric Engineering and Computer (MEC)*, 2555-2558. doi:10.1109/MEC.2011.6026014

Y. Wen, Z. L. (2011). A Middleware Architecture for Sensor Networks Applied to Industry Solutions of Internet of Things. *2011 Second International Conference on Digital Manufacturing & Automation*, 50-54. doi:10.1109/ICDMA.2011.21