Vol.2018-GN-103 No.23
Vol.2018-CDS-21 No.23
Vol.2018-DCC-18 No.23
2018/1/27

ECHONET Lite WebAPI と Protocol Bridge

藤田裕之[†1]　杉村博[†1]　村上隆史[†1]　一色正男[†1]

概要：ECHONET Lite 機器を Web Application から容易に扱えるようにするために HTTP/REST ベースの ECHONET Lite WebAPI（EL-WebAPI）を定義した．Machine readable format の機器仕様 Device Description も定義し ECHONET Lite の主要機器に関して作成した．EL-WebAPI を実証するために EL-WebAPI と ECHONET Lite を双方向に変換する Protocol Bridge を開発した．Protocol Bridge は Node.js/Node-RED を利用して実装した．

キーワード：エコーネットライト，ECHONET Lite，WebAPI，REST

ECHONET Lite WebAPI and Protocol Bridge

HIROYUKI FUJITA[†1]　HIROSHI SUGIMURA[†1]
TAKASHI MURAKAMI[†1]　MASAO ISSHIKI[†1]

Abstract: We have defined ECHONET Lite WebAPI(EL-WebAPI) that is based on HTT/REST framework in order for Web Application to control ECHONET Lite devices easily. We also defined Device Description that describes device specifications in a machine readable format. We have developed the Protocol Bridge between ECHONET Lite and EL-WebAPI to evaluate EL-WebAPI utilizing Node.JS and Node-RED.

Keywords: ECHONET Lite, WebAPI, REST

1. はじめに

IoT に関連するソフトウエア開発のフレームワークは様々な団体や企業が提案しカオス状態となっている．例えば Open Connectivity Foundation[1]は Qualcomm が開発した Alljoyn[2]や Intel が開発した IoTivity[3]を提案している．Open Mobile Alliance[4]は docomo が開発した GotAPI[5]を提案している．Google は Android の framework として Android Things[6]を，Apple は iOS の framework として HomeKit[7]を提案している．

これらのフレームワークは HTML5 などの Web 標準の技術を利用することで，スマートフォンや PC 用のアプリケーションを容易に開発することができる．

一方国内の HEMS の標準プロトコルである ECHONET Lite は TCP/UDP 上に規定された通信プロトコルである．組み込み機器での実装を前提に機器オブジェクト（EOJ），サービス（ESV），プロパティ（EPC），データ（EDT）など全ての項目をバイナリーデータで扱う．ECHONET Lite のコントローラを組み込みシステムとして開発するには今の仕様でも構わないが，最近はやりのマルチプラットフォーム（iPhone, Android, Windows, Mac など）対応のアプリケーションやサービスの開発は HTML5 を利用した Web Application が主流であり，現在の ECHONET Lite プロトコルとは相性が悪い．

†1 神奈川工科大学
Kanagawa Institute of Technology

そこで ECHONET Lite のプロトコルを専用の Protocol Bridge 経由で Web API に変換することで ECHONET Lite 機器を制御する Web Application を容易に開発できるようにすることを考えた（図 1 参照）．さらに，ECHONET Lite の機器寄りの実装を抽象化することで ECHONET Lite の仕様を理解していなくても使える API を目指す．

この論文では，ECHONET Lite WebAPI をどのように設計したかを説明するとともに，Protocol Bridge の実装についても説明する．

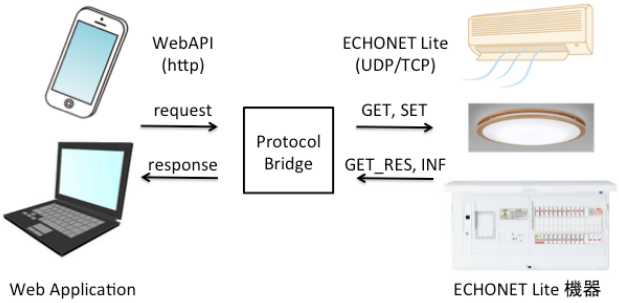

図 1　ECHONET Lite WebAPI と Protocol Bridge

2. これまでの Web API の試みと課題

2.1 試み

これまで WebAPI で ECHONET Lite を扱う試みがいくつか存在している．

経済産業省のスマートハウス・ビル標準・事業促進検討会から HEMS 情報基盤-HEMS データ利活用事業者間 API 標準仕様書[8]が提案されている．これは大規模 HEMS 実証

Vol.2018-GN-103 No.23
Vol.2018-CDS-21 No.23
Vol.2018-DCC-18 No.23
2018/1/27

事業において ECHONET Lite のコマンドのログを WebAPI で取得できるようにするものである．ECHONET Lite のコマンドログはバイナリーデータのまま利用されている．

大和田茂（Sony CSL）は ECHONET Lite 機器も含めネットワーク上の家電をコントロールするプラットフォーム Kadecot[9][10]を開発した．機器名は文字列で指定するがプロパティやデータはバイナリーデータのまま扱っているので ECHONET Lite の仕様を理解していることが前提となる．

大和ハウス工業株式会社は Sony CSL と共同で住宅 API[11]を提案し，実行環境として PicoGW[12]を開発している．ECHONET Lite のプロパティやデータも意味のある文字列に変換して扱っているので住宅 API は ECHONET Lite 対応の実用的 Web API の一つと言える．しかし基本的に ECHONET Lite のプロパティやデータを一対一に WebAPI にマッピングしたものであるため，ECHONET Lite のデバイス寄りの実装がそのまま引き継がれている．なお PicoGW で使用している文字列データは，筆者が昨年公開した”ECHONET Lite 機器オブジェクト詳細仕様の JSON データ”[13]を利用している．

2.2 課題

以下に，ECHONET Lite プロトコルを WebAPI 化する場合の検討課題を列挙する

1. データ表現

ECHONET Lite のデータは組み込み機器の実装を前提としたバイナリーデータであり，プロパティ毎にバイト単位でデータの解釈を定義している．デバイスの実装が色濃く反映されたプロパティのデータは WebAPI として扱いにくい．

A) オブジェクト・サービス・プロパティ・ステータスコードなどが全てバイナリーデータで定義されている．

例１：家庭用エアコンのオブジェクトコードは 0x0130，動作状態プロパティのプロパティコードは 0x80，ON/OFF に対応するステータスコードは 0x30/0x31 である．

B) データの型が乏しい．型は整数と RAW データである．

例２：小数値はアプリケーション側で換算する

例３：bitmap (1bit や数 bit)で状態を表現する場合がある

例４：２種類の値しかとらないデータを Boolean 型としてあつかっていない．

C) 同等の状態に対してプロパティ毎に異なる値をアサインする場合がある．例：家庭用エアコンの動作状態の ON/OFF は 0x30/0x31 であるが加湿モード設定の ON/OFF は 0x41/0x42 である．

D) 数値データの特定の値を数値ではなく状態として利用する場合がある．

例５：家庭用エアコンの風量設定は 0x31 から 0x38 までは８段階のレベルを示し，0x41 は自動設定を示す

例６：家庭用エアコンの消費電流計測値は 0x0000 から 0xFFFD までは数値として扱い，0xFFFE は「アンダーフロー」0xFFFF は「オーバーフロー」を表す

2. データ構造

１つのプロパティで複数要素のデータを扱う場合の統一したデータ構造の定義が存在しない

3. データの換算

ECHONET Lite でプロパティ値を取得したのちに，実際の値にするための換算が必要な場合がある．

例７：温度センサの温度計測値プロパティでは取得した値を 0.1 倍すると実際の値となる．

例８：低圧スマート電力量メータの積算電力量計測値プロパティは取得した値に係数プロパティの値と積算電力量単位プロパティの値を乗算するすると実際の値となる．

4. アトミックオペレーション

SET と GET を組み合わせて操作することを前提としたプロパティがある．

例９：低圧スマート電力量メータの積算電力量計測値履歴１プロパティの値を GET する前に積算履歴収集日１プロパティを SET する必要がある．

5. コントローラとデバイスの Interaction

ECHONET Lite におけるコントローラとデバイスの Interaction はプロパティデータの取得（GET），プロパティデータの設定（SET），状態の通知（INF）である．Web Application では「プロパティデータの取得」と「設定」は標準技術で実装ができるが「状態の通知」の扱いに関しては検討が必要である．また「設定」のみ対応するプロパティの扱いも検討が必要である．

6. Machine readable な format のデバイスの仕様書の欠如

ECHONET Lite 規格の仕様は, 約 300 ページの ECHONET Lite 規格書[14]と約 500 ページの ECHONET 機器オブジェクト詳細規定[15]に記載されている．プログラムを書く際には必要なデータの値を上記２つの PDF フォーマットの規格書から探してプログラムに手入力することになる．

3. WebAPI の設計方針

WebAPI を幅広く利用してもらうために WebAPI を設計する上で以下のような方針を定めた．

A) できるだけ標準的な技術を利用する

B) その上で必要な拡張を行う

C) 使いやすい（ECHONET Lite の仕様を理解していなくても使える）API を目指す

WebAPI を実現する手法としては現時点で一番ポピュラーで，さまざまなネットワーク環境で利用しやすい REST を採用し，データフォーマットは JSON とする．

この前提で Interaction とデバイスのモデル化を検討していたところ W3C[16]は Web Of Things(WoT)[17]として IoT に関する標準規格を策定中であることがわかった．汎用性を重視するため現時点ではまだ詳細が決まっていないが，REST を前提としたドラフト[18]の Interaction の考え方とデ

Vol.2018-GN-103 No.23
Vol.2018-CDS-21 No.23
Vol.2018-DCC-18 No.23
2018/1/27

バイスモデルの記述の仕方が課題解決の参考になったので以下に簡単に説明する．

Interaction として property の read/write 以外に action と event を用いる．action は property の write では表現しにくい動作（例：リセット，トグル，アラームの解除）に対応する．event はデバイスの状態変化に対応する．

デバイスモデルは property, action, event の詳細を JSON encoding の machine readable format として記述する．

これらを参考にして ECHONET Lite WebAPI を策定していくことにする．

4. WebAPI

4.1 Interaction

機器制御の標準的な操作は property の read/write であり，WebAPI では property の GET/PUT で実現する．ECHONET Lite では property の GET/SET に対応する．

リセットや状態のトグル，アラームの解除など property の write では表現が難しくかつ property の read が意味を持たない機能に関しては WebAPI では action として扱う．アクセスメソッドは POST である．ECHONET Lite では SET のみの property などに対応する．

機器からの通知（ECHONET Lite では INF に対応する）を WebAPI でリアルタイムに扱うには long polling, websocket, MQTT などの framework が考えられるが，今回は汎用性を重視しシンプルな REST を前提として考え WebAPI でのリアルタイム通知は実装しない．その代わりに INF 情報を Protocol Bridge がロギングしておき, WebAPI では event という Interaction でログを取得する．

以上の WebAPI と ECHONET Lite における Interaction の対応を表 1 に示す．

表 1　WebAPI と ECHONET Lite の Interaction の対応

WebAPI	ECHONET Lite	Comment
GET/PUT property	GET/SET	標準的な操作
POST action	SET only	Property の write で表現が難しい動作
GET event	INF	機器からの通知

4.2 URL

上記を考慮して REST のリソース（URL）を以下のように定義する．”GET /” 以外の URL は W3C の REST ドラフトに準拠している．

`GET /`

デバイスリストの取得．デバイスリストはデバイスを特定するための deviceId のリストである．deviceId は ProtocolBridge が各デバイスにアサインするユニークな値である．

`GET /<deviceId>`

指定したデバイスの Device Description の取得.

`GET /<deviceId>/properties/<propertyName>`

指定した property の値の取得.

`PUT /<deviceId>/properties/<propertyName>`

指定した property の値の設定.

`POST /<deviceId>/actions/<actionName>`

指定した action の実行.

`GET /<deviceId>/events/<eventName>`

指定した event の log の取得.

4.3 データ型とデータ構造

データ表現に起因する課題は，下記の方針で対応する．

A) オブジェクト，プロパティ，状態は名前を定義する
B) データの型を充実させる
C) 複数要素のデータを表現する統一した構造を定義する

オブジェクト，プロパティ，状態の名前は基本的に ECHONET Lite の英語の description を利用し lower camel case として定義した．

W3C の WoT ドラフトではデータ型（data type）として boolean, integer, number, string, array, object を定義している．

ECHONET Lite EL-WebAPI では上記の data type を採用し以下の拡張を行った．

Data type “boolean”と”key”には，取りうる値とその説明を記述する”value”という member を定義した．

```
"value": {
   <value>: {
     "ja":<description in Japanese>},
     "en":<description in English>}
   }
   ...
}
```

Data type “number”と”integer”にはデータの単位，最小値，最大値を記述する”unit”, ”minimum” , ”maximum”という member を定義した．

```
"unit":<unit>,
“minimum”:<minimum value>,
“maximum”:< maximum value>,
```

よく使われる特定の型として “level”, “date”, “percentage”, “raw”を定義した．

“level” は強弱のレベルを 1（最弱）から 10（最強）の整数値で表す．

“date”は Date&Time を表すデータ型．ISO8601 準拠. "yyyy-MM-ddThh:mm:ss"の format の string である．

“percentage”は割合を百分率の整数値で表す．単位は%.

“raw”はバイナリーデータをそのまま利用する場合．1byte データの配列で記述する．

複数要素のデータを表現するには”array”と”object”を利用する．

“array”は同じ data type のデータの配列である。

“object”は複数の要素のデータから構成されるデータを JSON の object 型式で表現する．各項目は boolean から array の data type を利用して記述する．

表 2 に EL-WebAPI の data type の一覧を示す．

Vol.2018-GN-103 No.23
Vol.2018-CDS-21 No.23
Vol.2018-DCC-18 No.23
2018/1/27

表 2 EL-WebAPI の data type

Data Type	Desription	Member	Data format
boolean	true または false の値	value	boolean
key	状態を表す keyword	value	string
number	固定小数点数値	unit minimum maximum	number
integer	整数値	unit minimum maximum	number
level	強弱のレベルを表す		number
date	Date&Time を表す		string
percentage	割合を百分率で表す		number
raw	バイナリデータを表す		number
array	配列		array
object	複数の要素のデータから構成されるデータを表現する		object

以下に Device Description 内の data の記述例と REST の response 例を示す.

Example of Device Description (Boolean, key)

```
data:{
  "type": "boolean",
  "value": {
    "true": {"ja":"異常あり", "en":"Fault"},
    "false": {"ja":"異常無し", "en":"No Fault"}
  }
}

data:{
  "type":"key",
  "value":{
    "normal":{"ja":"通常灯",
              "en":"Normal Lighting"},
    "night":{"ja":"常夜灯",
             "en":"Night Lighting"},
    "color":{"ja":"カラー灯",
             "en":"Color Lighting"}
  }
}
```

Example of REST response in JSON format (Boolean, key)

```
{"on":true}, {"on":false}
{"operatingMode":"color"}
```

Example of device description (number, integer)

```
data:{
  "type": "number",
  "unit": "kWh"
}

data:{
  "type": "integer",
  "unit": "℃",
  "minimum":0,
  "maximum":50
}
```

Example of REST response in JSON format (number, integer)

```
{"integralEnergy":15.5},
{"temperature":25},
{"temperature":-100}
```

Example of REST response in JSON format (level)

```
{"airFlow":4}
```

Example of REST response in JSON format (date)

```
{"date":"2017-10-10T13:50:40"}
```

Example of REST response in JSON format (percentage)

```
{"humidity":25}
```

Example of REST response in JSON format (raw)

```
{"raw":[23, 12, 0,...]}
```

Example of REST response in JSON format (array)

```
{"energy":[20, 34, 59, 109]}
```

Example of device description (object)

```
data:{
  "type": "object",
  "field":[
      {
          "name": "r",
          "description":{ "ja":"赤", "en":"Red" },
          "data": {
                   "type": "integer",
                   "minimum":0,
                   "maximum":255
          }
      },
      {
          "name": "g",
          "description":{ "ja":"緑", "en":"Green" },
          "data": {
                   "type": "integer",
                   "minimum":0,
                   "maximum":255
          }
      },
      {
          "name": "b",
          "description":{ "ja":"青", "en":"Blue" },
          "data": {
                   "type": "integer",
                   "minimum":0,
                   "maximum":255
          }
      }
  ]
}
```

Example of JSON data

Vol.2018-GN-103 No.23
Vol.2018-CDS-21 No.23
Vol.2018-DCC-18 No.23
2018/1/27

```
{
      "rgb": {
          "r": 20 },
          "g": 255},
          "b": 0  }
      }
}
```

4.4 Device Description

Device Description は ECHONET Lite WebAPI におけるデバイスの仕様の記述である. W3C の REST ドラフトに準拠し，4.3 節で定義した data type を利用する．Device Description の Format を以下に示す.

```
{
  "type":<device type>,
  "description":<description>,
  "properties":[
    {
      “name”:<property name>,
      “description”:<description>,
      “writable”:<flag to write>,
      “observable”:<flag to observe>,
      “query”:<data type object>
      “data”:<data type object>
    }
  ],
  "actions":[{“name”:<action name>}],
  "events":[{“name”:<event name>}]
}
```

表 3 に Device Description の各メンバーの説明を示す.

表 3　Device Description の member

Member	Description
type	device type 名
description	ECHONET Lite で定義された機器の名称
properties	property objects の集合
actions	action objects の集合
events	event objects の集合
property.name	property 名
property.description	ECHONET Lite で定義された property の名称
property.writable	property が write できる場合は true，できない場合は false
property.observable	property の状態変化を event で取得できる場合は true，できない場合は false
property.query	query の data type．GET で query が必要な場合のみ記述
property.data	property の data type
action.name	action 名
event.name	event 名

以下に例として一般照明の Device Description の抜粋を示す.

```
{
"type":"generalLighting",
"description":{"ja":"一般照明",
               "en":"General Lighting"},
"properties":[
  {
    "name":"on",
    "description":{"ja":"ON/OFF 状態",
                   "en":"ON/OFF Status" },
    "writable":true,
    "observable":true,
    "data":{
      "type":"boolean",
      "value":{
        "true":{"ja":"ON", "en":"ON"},
        "false":{"ja":"OFF", "en":"OFF"}
      }
    }
  },
  {
    "name":"brightness",
    "description":{"ja":"輝度",
                   "en":"Brightness" },
    "writable":true,
    "observable":false,
    "data":{
      "type":"percentage"
    }
  },
  {
    "name":"operatingMode",
    "description":{"ja":"動作モード",
                   "en":"Operating Mode"},
    "writable":true,
    "observable":false,
    "data":{
      "type":"key",
      "value":{
        "normal":{"ja":"通常灯",
                  "en":"Normal Lighting"},
        "night":{"ja":"常夜灯",
                 "en":"Night Lighting"},
        "color":{"ja":"カラー灯",
                 "en":"Color Lighting"}
      }
    }
  },
  {
    "name":"rgb",
    "description":{"ja":"RGB 設定",
                   "en":"RGB Value" },
    "writable":true,
    "observable":false,
    "data":{
      "type":"object",
      "field":[
        {
          "name":"r",
          "description":{ "ja":"赤", "en":"Red" },
          "data":{
```

```
            "type":"integer",
            "minimum":0,
            "maximum":255
          }
        },
        {
          "name":"g",
          // 以下省略
          }
        },
        {
          "name":"b",
          // 以下省略
          }
        }
      ]
    }
  }
 ],
 "actions":[],
 "events":[{"name":"on"}]
}
```

4.5 エラー処理

ERROR 処理に関しては W3C のドラフトに記述がないので以下のように定義した.

ERROR が発生した場合は Status Code を "400" とする.

ERROR 時の RESPONSE は以下のように ErrorType と ErrorMessage を返す.

```
// RESPONSE
400 Error
{
 "type":<ErrorType>,
 "message":<ErrorMessage>
}
```

ErrorType は Error の種類を示す keyword である．表 4 に ErrorType の詳細を示す。

表 4　ErrorType

ErrorType	Description	Example
rangeError	設定する値が仕様の範囲外の場合	number, integer: 値が min と max の間にない場合 key: key が存在しない場合 level: 値が 1...10 でない場合
referenceError	設定する deviceName や propertyName が存在しない場合	
typeError	設定する値の型が仕様に反する場合	
timeoutError	機器から一定時間内に返答がない場合	
deviceError	機器から受信したデータが error に対応する値の場合 機器から SNA を受信した場合	0xFFFE が Overflow を意味する場合 GET_SNA を受信した場合 SET_SNA を受信した場合

ErrorMessage は ERROR の詳細を記述する任意の String data.

以下にレスポンスの例を示す.

```
{"type":"rangeError", "message":"out of range"}
{"type":"rangeError", "message":"no key matches"}
{"type":"referenceError", "message":"deviceName
is wrong"}
{"type":'referenceError", "message":"propertyName
is wrong"}
{"type":"typeError", "message":"data should be
boolean"}
{"type":"timeoutError", "message":"timeout !"}
{"type":"deviceError", "message":"undefined"}
{"type":"deviceError", "message":"overflow"}
{"type":"deviceError", "message":"GET_SNA"}
```

エラー処理の実装としては，Protocol Bridge がエラーを検出する場合とデバイスがエラーを検出する場合がある.

Protocol Bridge は WebAPI の request 内容を Device Description の記述をもとにエラー判断を行う．ErrorType は” rangeError”, “referenceError”, “typeError”, “timeoutError” に対応する.

ECHONET Lite でのデバイスのエラー通知は，サービスコードの SNA(Service Not Available)を使う場合とエラーに対応するプロパティデータの値を利用する場合がある．後者の例としては 2.2 節例 6 があり，Protocol Bridge は WebAPI で以下のレスポンスを返す.

```
{"type":"deviceError", "message":"underflow"}
{"type":"deviceError", "message":"overflow"}
```

5. Protocol Bridge

5.1 構成

Protocol Bridge の構成を図 2 に示す. WebAPI 側の Device の仕様を記述した Device Description と ECHONET Lite 側の Object の仕様を記述した EL Object Description を利用して双方向の Protocol 変換を行っている．EL Object Description は ECHONET 機器オブジェクト詳細規定のデータ化[19]の成果を利用したものである.

Vol.2018-GN-103 No.23
Vol.2018-CDS-21 No.23
Vol.2018-DCC-18 No.23
2018/1/27

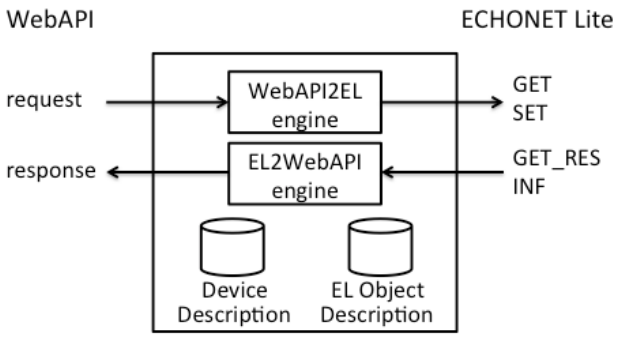

図２ Protocol Bridge の構成

Protocol Bridge の動作を簡単に説明する．

Web アプリから EL-WebAPI を利用した REST の request を受信すると，WebAPI2EL engine は Device Description を利用して Parse を行い，EL Object Description を利用して ECHONET Lite packet を生成し機器へ送信する．

機器から ECHONET Lite のコマンドを受信すると，EL2WebAPI engine は EL Object Description を利用して Parse を行い，Device Description を利用して EL-WebAPI の response を生成し Web アプリへ送信する．

多くの場合 EL-WebAPI と ECHONET Lite を単純に変換すればよいが「データの換算」と「アトミックオペレーション」が関係する場合は例外である．

データの換算に関する情報は，EL Object Description の property に” magnification”と” coefficient”という member で記述されている．ECHONET Lite の GET のレスポンスデータをパースする際に上記 member が存在する場合は値を換算したのちに EL-WebAPI のレスポンスに変換する．magnification は ECHONET Lite の GET で取得したプロパティの値を実際の値に変換する際の倍数を示す数値データである．coefficient は ECHONET Lite の GET で取得したプロパティの値を実際の値に変換する際に他のプロパティの値を係数として乗算する場合のプロパティ番号を示す．

アトミックオペレーションが必要なプロパティへの GET request には query が必要である．Protocol Bridge が ECHONET Lite で SET する値を query で指定する。

アトミックオペレーションに関する情報は EL Object Description の property に”atomic”という member に SET すべきプロパティ番号が記述されている．Protocol Bridge は WebAPI の GET リクエストをパースして EL Object Description に atomic という member が存在する場合は，query の値を atomic member のプロパティ番号に SET したのちにプロパティを GET する．

5.2 実装

EL-WebAPI を検証するために Protocol Bridge を Node-RED を利用して実装した．Node-RED は Node.JS を利用した Visual Programing 環境である．機能の単位となる node を JavaScript で記述し，node を接続して Flow を作成することでプログラムを記述する．Node-Red には UDP の送受信や JSON データのパースなどの node があらかじめ用意されているのでロジックの開発に集中できる．図３に Protocol Bridge の Flow を示す．

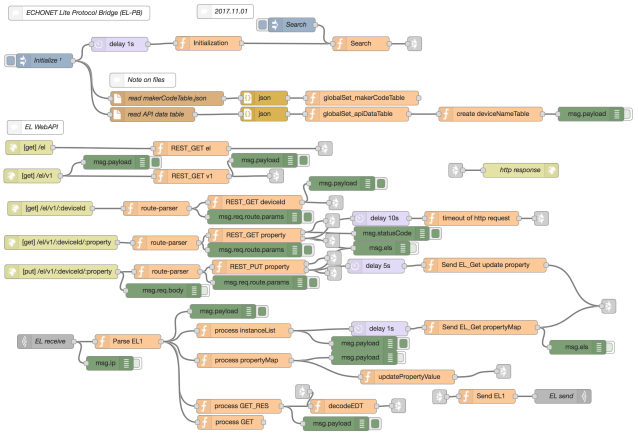

図３ Protocol Bridge の Flow

5.3 動作検証

一般照明を対象機器とし EL-WebAPI と Protocol Bridge の動作検証を行った． REST の request は POSTMAN[20] を利用した．一般照明の初期状態は消灯状態である．表５に動作検証の項目と結果を示す．結果の欄は動作仕様通りの結果が得られた場合は〇、得られなかった場合は×とする。

表５ 動作検証結果

Request	確認項目	結果
GET /	generalLighting_01 を含む Device List 取得	〇
GET /generalLighting_01	Device Description 取得	〇
GET /generalLighting_01/properties/on	Response が {“on”:false}	〇
PUT /generalLighting_01/properties/on {"on":true}	照明が点灯	〇
GET /generalLighting_01/events/on	“on”のログ取得	〇

6. 結論

ECHONET Lite の仕様を理解していなくても Web Application から容易に ECHONET Lite 機器の制御ができることを目指して EL-WebAPI を定義した．EL-WebAPI と ECHONET Lite のプロトコル変換を行う Protocol Bridge を開発し，実際に Web Application から ECHONET Lite 機器の制御を容易に行えることを確認した．今回提案した EL-WebAPI と Protocol Bridge が活用されて ECHONET Lite 機器を扱う Web アプリケーションやサービスが開発されることを期待している．

参考文献

[1] https://openconnectivity.org

Vol.2018-GN-103 No.23
Vol.2018-CDS-21 No.23
Vol.2018-DCC-18 No.23
2018/1/27

[2]https://openconnectivity.org/developer/reference-implementation/alljoyn
[3] https://www.iotivity.org
[4] http://openmobilealliance.org
[5] https://device-webapi.org/gotapi.html
[6] https://developer.android.com/things/index.html
[7] https://www.apple.com/jp/shop/accessories/all-accessories/homekit
[8]http://www.meti.go.jp/committee/kenkyukai/shoujo/smart_house/pdf/009_s10_00.pdf
[9] https://www.sonycsl.co.jp/tokyo/347/
[10] https://github.com/SonyCSL/Kadecot-JS
[11] http://www.daiwahouse.co.jp/lab/HousingAPI/
[12] https://github.com/KAIT-HEMS/PicoGW
[13] https://github.com/KAIT-HEMS/ECHONET-APPENDIX
[14] https://echonet.jp/spec_v112_lite
[15] https://echonet.jp/spec_object_rj
[16] https://www.w3.org
[17] https://www.w3.org/WoT/
[18] Web Thing API http://iot.mozilla.org/wot/#
[19] 藤田裕之「ECHONET 機器オブジェクト詳細規定のデータ化」情報処理学会第 18 回 CDS 研究会
[20] https://www.getpostman.com