Vol.2015-SLDM-170 No.28
Vol.2015-EMB-36 No.28
2015/3/7

軽量言語を用いた組込みシステム開発環境の研究

金森一真[†1]　　早川栄一[†2]

組込みコンピュータの高性能化とネットワークに接続可能なインタフェースが搭載されたことにより、センサ機器を利用した Web サービスの開発が盛んになっている。従来の組込みシステム開発では、ターゲットデバイスの性質に応じて複数の言語や開発環境を用いることから、開発効率が低いという問題がある。本研究は、Web 開発に適した軽量言語を組込み開発に適用し、ネットワーク接続された組込み機器群のシステム開発を容易にする開発環境を提供する。上記の問題に対して、本環境では Web との親和性の高い JavaScript を記述言語として採用し、サーバ上で動作する開発環境およびデバイス制御ライブラリ群を用意することで、単一の言語環境で複数の機器の制御を可能とした。制御ライブラリには、開発の抽象度に応じた API の提供、サーバ上でセンサ機器の制御を行うための JSON を中心としたデータ通信機構を提供した。

Research of Embedded System Development Environment With Lightweight Language

KAZUMA KANAMORI[†1]　　EIICHI HAYAKAWA[†2]

The Development of Web services using sensors become popular by higher performance and network interface available on embedded systems. Since the use of different programming languages and development environments on the nature of the target device, development efficiency gets lower in conventional embedded system development. In this study the unique lightweight language suitable for Web development is utilized in entire embedded system development including sensor handling and web services. We use JavaScript not only web services but device handling to describe the entire system in the single language environment. JavaScript based device control library on the embedded system is presented. The library provides different abstraction level API such as hardware dependent layer and functional layer. We utilize JSON to the data communication mechanism between embedded system and web service.

1. はじめに

組込みコンピュータにネットワーク機能が搭載されたことにより、Web サービスと連携した Web of Things[1]アプリケーション(以下、WoT アプリケーションと呼ぶ)開発が盛んになっている。

センサ機器から得られたデータを多様な Web サービスと連携することによって、新たな付加価値を持ったサービスを提供できる。しかし、WoT アプリケーションの開発には、Web アプリケーション開発と組込みシステム開発が必要となり、その場合、複数の開発環境を用いることから、開発効率が低いという問題がある。

組込みシステム開発では、広く C 言語を用いているが、Web 開発に適した軽量言語を制御ソフトウェアの開発言語として用いることが可能になりつつある。従来の C 言語を用いた組込みシステム開発のスタイルの替わりに、Web 環境を利用して、軽量言語によって記述可能な制御ソフトウェアを開発することによって、Web ベースによる開発環境の生産性の高さやネットワークサービスとの親和性を活かして WoT アプリケーション開発が可能となると考えられる。

本研究は、軽量言語による Web ベースの組込みシステム開発環境、特に JavaScript ベースのセンサライブラリの検討およびプロトタイプ実装について報告する。具体的には、Web サービスサイドおよび組込みシステムサイドで記述可能な JavaScript を用いて、センサ制御が行えるセンサ API ライブラリと、サーバサイドでセンサ制御プログラムが管理可能なデータ通信機構を実現する。センサ API ライブラリは、センサ機器ごとの仕様の違いを吸収し、軽量言語によって記述可能な、抽象化されたセンサ制御 API を利用することができる。このセンサ API ライブラリを組込み機器ごとの制御ライブラリとして提供する。データ通信機構は組込み機器ごとに記述してあるセンサ制御プログラムを Web サーバで管理し、必要に応じて組込み機器へ転送する。転送されたプログラムは組込み機器側で実行されることで、プログラムの再利用が可能になる。

この開発環境を利用することで、WoT アプリケーションの組込みシステム部分が軽量言語によって開発が可能であることを示すとともに、Web ベースの環境の組込みシステム開発への適用可能性を示す。

[†1] 拓殖大学大学院　工学研究科
Graduate School of Engineering, Takushoku University
[†2] 拓殖大学　工学部
Faculty of Engineering, Takushoku University

Vol.2015-SLDM-170 No.28
Vol.2015-EMB-36 No.28
2015/3/7

2. 問題分析

センサネットワークにおいてセンサデータや通信方式などの異なる環境を吸収して複数の Web サービスと連携するための研究がある[2]。これらの研究の問題を次に述べる。

2.1 開発言語の問題

一般的に、組込みシステム開発には広く用いられている C 言語が使われる。使われる理由は組込み機器自体にリアルタイム性や、低リソースでの高速な処理を求められているが、アプリケーションによっては上記の制約を必ずしも満たす必要のないシステムもある。これまでの組込みソフトウェア開発に用いられた C 言語で機能を実装するのではなく、軽量言語を用いて開発する手法が提供されている[3][4]。この部分で、組込みシステムと Web サービスとの間で言語が異なり、プログラマは複数の言語によってプログラミングを行うという負荷がかかる。

2.2 センサボード仕様の多様性の問題

現在、Raspberry Pi や Intel Edison などの高性能で安価なマイクロコントローラを入手可能になっている。また、Arduino などの手軽な IDE を持った環境も存在する。WoT アプリケーションでは、仕様の異なるセンサボードを利用して Web サービスと連携したサービスを提供する。これらのセンサボードは ARM プロセッサおよび Linux を搭載していて、ユーザ層ではほぼ同じプログラミングインタフェースを利用できる。

その一方で、入出力部分については各ボードによって異なることが多い。また、AD 変換機能を含まないチップセットや、I2C、SPI、GPIO などセンサとの通信も多様であり、この部分は OS レベルでは吸収されていない。

2.3 Web に適した通信機構の問題

センサデータを Web サービス側が利用する際には、取得したセンサデータを適切なデータ形式に変換してから利用するのが一般的である。変換するデータ形式には様々な種類が存在するため、Web サービスごとに異なるデータ形式を採用した場合、データ変換を行う処理を別々に実装する必要があり、開発の手順が増加し複雑化する。また、Web サービスがセンサデータのやり取りを行うための通信機構にも、それぞれ対応する仕様に対応しながら開発を行わなければならない。また、Web サービスではクライアントは Web ブラウザを想定することが多いが、組込みシステムをクライアントとした場合、ブラウザが動作していないために、ユーザとのインタラクションが変わっている。また、クライアント側でもサーバが動いていることから、対称的、自律的な構造を備えている。

3. 設計方針

本研究で対象とするシステムの設計方針は次のとおりである。

(1) Web に適した軽量言語として JavaScript による Web サービスサイドおよび組込みシステムサイドを記述

本研究では、クライアントのブラウザ上で動的な機能を提供するために用いられる JavaScript を、Web サービスサイドおよび組込みシステムサイドに拡張して利用可能にする。Web サービスサイドについては、Google Chrome 上の JavaScript インタプリタから派生した Node.js[5]を用いて、MEAN スタック（MongoDB、Express、AngularJS、Node.js）上で Web サービスを記述することは一般的に行われている。これにより、両方のサイドをすべて JavaScript および HTML で記述可能になる。

本研究では、これを拡張して、組込みシステムボード上で Node.js を稼働させ、センサに関する処理はすべて JavaScript で記述する。これによって、すべて JavaScript で記述可能になり、生産性の向上が期待できる。Node.js はイベント駆動で、ノンブロッキング I/O を採用している。また、Node.js にはネットワークアプリケーションが開発しやすくするモジュール群などがあり、システムの負荷によるリアルタイム性の低下を避けることも可能になる。JavaScript を用いた場合、システム性能の低下が懸念されるが、現在のセンサボードは CPU 性能およびメモリ量は言語の実行に十分であると考えられる。性能面については後述する。

(2) ボードに共通のセンサ API の提供

2 章の問題分析で述べたように、ボードごとのセンサの使用ポートやハードウェア間の通信、チップの性能の違いを解消するために、共通のセンサ API を用意する。これによって、使用ハードウェアの違いによるプログラムの変更を極力減らすようにする。具体的には、ポート番号や通信方法については、プログラム本体から分離して、定義および変更可能にする。

(3) JSON を中心としたデータ通信機構の提供

本システムでは、両サイドの間の通信は JSON をベースとする。JSON は JavaScript と親和性の高いデータ表現であり、一般性が高く、Web 上での利用もしやすい。また、REST などを用いることで組込みシステムサイドの状態保持を容易にすることができる。

4. 設計

4.1 システムモデル

本システムでは、MVC モデル 2 に沿ったシステムモデルを導入する。MVC モデル 2 は、Web アプリケーションの設計において、Model、View、Controller の三つの要素に

分離し開発を行う、MVC モデルを参考にした設計手法である。MVC はそれぞれ、次の役割を担う。

・ Model

フォームなどから入力されたデータを保持し、データベースの接続/操作などを行う要素である。従来の MVC モデルでは Model が行ったデータ変更を View へ通知する責任があるが、MVC モデル 2 では、それができないので View が直接 Model の状態を取得する。

・ View

ユーザからの入力やイベントなどの HTTP リクエストに対する実行結果の表示を行う要素である。Model からデータの取得、更新を行う。

・ Controller

ユーザからのリクエストメソッドに合わせて Model や View を呼び出し、処理を実行する要素である。リクエストメソッドは Controller が受け取り、そこから必要に応じて Model、View が呼び出される。

MVC モデルおよび MVC モデル 2 は基本的にユーザインタフェースを持つデスクトップアプリケーションに対応したものである。そのため、MVC モデルをそのまま、Web のソフトウェア開発モデルとしては利用できない。これを WoT アプリケーションに適用するために、センサを扱えるようなコンポーネントを用意して、モデルを拡張した。

ここでは、WoT アプリケーション開発モデルに必要なものは、次の要素であると考えられる。

・ データに関する手続きを行う Model
・ ユーザへのデータ表示を行う View
・ Model、View 要素の通知、処理を行う Controller
・ センサの入出力を行う Sensor

まず、Model、View、については MVC モデル 2 と同じである。Controller は新しく Sensor 要素を用意したため、この Sensor 要素からの通知されるデータやイベントなどを処理する必要がある。具体的には、定期的に Sensor から送られてくるセンサデータの View への送信や、Web ページからの入力をセンサに通知する役割を担う。Sensor 要素はセンサボードなどの組込みシステムサイドで動作してセンサ入出力を取り扱う。これは、組込みシステムサイドでの View に相当する。

4.2 システム構成

本開発環境では、Web ブラウザ機能を持ったサーバサイドと、組込みボードからなるクライアントサイドから構成している。いずれのサイドでも JavaScript の実行環境である Node.js が動作する。サーバサイドは Node.js 上で動作するフレームワークであり Express[6]をベースに構成する。

組込みシステムサイドで動作するプログラムは、Web サービスサイドで用意して、これを組込みシステムサイドへ送ることでプログラムをデプロイして、実行できるようにする。これによって、組込みシステムサイドに開発環境を用意する必要がなくなり、ノードが増えた場合のプログラムの管理を容易にする。

組込みシステムサイドでは、次の要素を提供する。

・ Controller：Controller はサーバからの要求を受け取り、各実装コードを探して、それらへ処理を渡す。
・ Sensor：Controller において、センサからのデータの取得、およびアクチュエータへのデータの送信を行う。通常の MVC の View 部分に相当する。
・ Model：性能上もしくは機能的な要求によって、組込みシステムローカルなストアが必要な場合、この Model を呼び出す。通常は、サーバの Model を利用し、データはサーバへ送る。

また、Web サービスサイドは、次の MVC の要素を提供する。

・ Model：Sensor から渡されたセンサデータ、ユーザから入力されたフォームデータなどを保持/更新し、データベースを利用している場合は接続などの操作を行う。
・ View：Controller から渡されたセンサデータを Web ページ上に表示する。
・ Controller：Controller はユーザからの入力データやイベントを受け取る。 Sensor からのデータは制御と共に View へ渡す。 ユーザからのデータもしくはイベントの種類によって、Sensor、Model へ処理を依頼する。

このように各要素の役割を明確にすることで、目的の制御処理を実装する部分が開発者にわかりやすくなる。また、各要素が独立性を保つことで、プログラム全体を修正する箇所が少なくなる。

4.3 センサ制御機構

4.3.1 Sensor

Sensor はユーザから入力されたデータを元に、センサ制御を行い、センサの状態を読み取る役割を担う要素である。

センサ制御については Web 開発で用いられている言語によって制御可能にする。理由としては、開発言語を統一化することで、Web 開発を得意とする開発者であれば、慣れている言語で開発が可能になる。その際には、Web 開発言語でセンサ制御可能な制御ライブラリを用意する。

センサライブラリは、階層構造を持つ。低レベル部分では、各センサ類を制御するのに必要となる次の三つの機能を提供する。

・ ディジタル入出力
・ アナログ入出力
・ イベント通知

Vol.2015-SLDM-170 No.28
Vol.2015-EMB-36 No.28
2015/3/7

ディジタルおよびアナログ入出力は、センサやアクチュエータの入出力の基本となる部分である。また、割込みは JavaScript のイベントメカニズムにマッピングする。ハードウェアの性質については、これらに対して対応する定義データベースを用意する。ポート番号や ADC の精度、通信方式などは、チップ名やハードウェア名から検索できるようにして、ボードで使用しているハードウェアに依存しないようにする。

また、機能による高レベルの入出力ライブラリも用意する。これによって低レベル入出力だけではプログラムの記述が冗長になり、機種依存性が高くなることを防ぐことができる。例えば、LED や温度センサ、サーボモータという抽象レベルでの記述を可能にすることで、機種に依存する部分を隠蔽する。これらの階層的なライブラリは、JavaScript のクラス機構を用いて実現する。これによって、ライブラリの利用性や可読性の向上を可能にする。

今回提案した構成では、クライアントも Node.js が動作するサーバとして機能している。クライアント上の資源は、クライアントマシンの URL にマッピングしているので、外部から URL を用いて指定することができる。

4.4 Web サービス機構

ここでは Web サービスサイドの要素について述べる。

4.4.1 Model

Model はユーザが入力したデータやセンサデータなどをサーバ上で管理し、データベースを利用している際には、その操作手続きをする役割を担う。これによってセンサデータを一貫して扱うことができるようになる。センサデータのフォーマットは各センサによって大きく異ることから、方針に述べたとおり JSON をベースとする。これを KVS(Key-Value Store)に格納することで、低オーバヘッドで永続性をもたせる。

4.4.2 Controller

Controller はユーザからの入力データやイベント、Sensor からのセンサデータの入出力の結果を各要素に通知する役割を担う。

ユーザからの入力データやイベントの受け取りには HTTP 通信を利用する。理由としては、Web ページからの入力データやボタン等によるイベントは HTTP プロトコルを用いてサーバとやり取りするのが一般的なためである。

Sensor からのデータは、定期的に Controller へ渡す場合がある。この場合、新しいセンサデータの更新を Web ページに表示する際に、HTTP 通信ではサーバ側からクライアントへ更新データを載せた Web ページを送ることができない。そのため、ユーザが Web ページ上にあるボタンなどの UI から HTTP 通信の GET メソッド等を使って更新データを取得するしかない。これでは、リアルタイムにセンサデータを見ることができないので、相互通信可能な通信プロトコルを用いることでサーバ側からでもクライアントへセンサデータの送信が可能となる。

5. 実装

この章では、WoT アプリケーション向け開発モデルを用いての実装について、どのように検証システムを実装したかを述べる。

5.1 検証アプリケーション

本研究では、提案開発モデルを用いてアプリケーションのプロトタイプ実装を行い、その動作を検証した。

実験では、Web サービスからのイベント、センサからの入力、センサデータの出力機能があるシステムを想定した。

プロトタイプアプリケーションとして、ユーザがスイッチを押すと光センサと温度センサのデータを Twitter 上にツイートし結果を出力する。また、Twitter にユーザが特定のツイートすることでカメラモジュールから研究室内を撮影し、予め設定しておいたメッセージと撮影した画像を添付したメールを結果として返す機能を開発した。

実験で使用するマイコンボードとして RaspberryPi[7]を OS に Raspbian を採用した。

5.2 アプリケーションの構成

組込みシステムサイドでは、Sensor および Controller を実装している。

Sensor は(1)スイッチ、(2)温度センサ、(3)光センサ、(4)カメラモジュールの入出力データを Controller に通知する。(1)が押された場合、一度(2)、(3)のセンサデータを取得してから、そのデータを Controller へ渡す。(4)は Controller の通知からカメラモジュールを起動し、撮影データを Controller へ返す。

Web サービスサイドの Controller は通知イベントハンドラで Twitter からの通知を受け取り(4)の制御を Sensor に依頼する。(1)の入力で Sensor から(2)、(3)のデータを受取り、ユーザへの表示を View へ依頼する。

Model は Sensor から受け取った温度センサ、光センサのデータを View へ表示するためのデータに加工する。また、Controller から要求があった場合、カメラモジュールで撮影した画像とメールで送るためのユーザ情報を渡す。

View は Controller からの表示依頼を受け、ユーザへメッセージと撮影画像を添付したメールを GMail に送る。同じく(2)、(3)のデータを Twitter へツイートする。

Vol.2015-SLDM-170 No.28
Vol.2015-EMB-36 No.28
2015/3/7

図 1　プロトタイプアプリケーション

クラス

Temperature({pin: pin番号, time: ミリ秒指定})
Light({pin: pin番号, time: ミリ秒指定})
Button({pin: pin番号, event: 入力状態の指定})
Camera(ファイル名, 静止画/動画の設定)

メソッド

on(イベント, 処理内容)
Shooting(ファイル名, 静止画/動画の設定)

図 2　センサ制御の記述例

図 1 に実験環境を示す。研究室内の環境を監視するために(1)温度センサ、(2)光センサ、(3)カメラモジュールが設置されている。温度センサには高精度 IC 温度センサ LM61BIZ、光センサには CDS セル 5mm タイプ、カメラモジュールには RaspberryPi Camera Board を使用した。また、ユーザが研究室内の環境を取得するための(4)スイッチが設置されてある。スイッチにはタクトスイッチを使用した。(1)、(2)、(3)、(4)は RaspberryPi に接続されており、この RaspberryPi 上で実行可能である制御ライブラリからセンサの制御を行う。各センサは RaspberryPi 上に実装してある汎用 I/O(GPIO と呼ぶ)と接続することで入出力を行うことが可能である。

5.3 動作検証

WoT 向けアプリケーションフレームワークを用いて、アプリケーションを作成し、実行すると、Web サービスと通信を行いデータ出力できることを確認した。その動作結果を述べる。

研究室内の温度と部屋の明るさは、RaspberryPi の GPIO に接続したあるスイッチを押すことで、センサデータの表示と予め設定しておいたメッセージを結果としてツイートしている。この結果から Twitter 上でセンサデータの表示が可能であることを確認できた。

カメラモジュールから研究室内を撮影した画像を取得するには、ユーザが Twitter 上に「人」という特定の単語を含めたツイートをすると RaspberryPi に接続されているカメラモジュールから研究室内の撮影を行う様子である。この例では、カメラモジュールから撮影した画像と予め設定しておいたユーザ情報をメールに添付して転送している。ユーザはメールに添付してある撮影画像を見ることで研究室内を確認できる。この結果から、Twitter 上のイベントからセンサの出力を表示できることを確認できた.

図 2 は、センサの制御記述例である。Temperature は、温度センサの制御を行うクラスである。Temperature クラスの第一引数にはピン番号を指定し、第二引数にはセンサデータを周期的に読み込む時間をミリ秒単位で指定する。第二引数は省略可能で、このプロトタイプアプリケーションではスイッチによる入力のタイミングでセンサデータを取得している。Light は、光センサの制御を行うクラスである。温度センサと同じ記述方法で制御可能である。Button は、タクトスイッチの制御を行うクラスである。第一引数は Temperature、Light と変わらない。第二引数にはスイッチが押された時の状態を指定する。Temperature、Light、Button クラスには on メソッドが用意されている。この on メソッドには、イベント名と処理内容を記述する。Camera は、カメラモジュールを制御するためのクラスである。shooting メソッドの第一引数には撮影した画像のファイル名を指定し、第二引数には静止画の撮影か、動画の撮影かを指定する。撮影した画像はカレントディレクトリに置かれる。戻り値として保存したディレクトリのファイルパスを返す。

本動作検証では、WoT アプリケーション向け開発モデルを使うことで、WoT アプリケーションの開発が可能となり、その動作も確認できた。

6. まとめ

本報告では、軽量言語による Web ベースの組込みシステム開発環境、特に JavaScript ベースのセンサライブラリの検討およびプロトタイプ実装について述べた。MVC をベースとした WoT アプリケーションのモデルを提示し、記述に必要となるセンサライブラリ群を開発した。また、そのライブラリの有効性を示すためのアプリケーションを作成し、記述の容易性について確認した。

Vol.2015-SLDM-170 No.28
Vol.2015-EMB-36 No.28
2015/3/7

今後の課題としては、より多くのアプリケーション記述にこのライブラリを用いて、有効性の検証を行うことである。また、ライブラリ構造の見直しや、性能面での改善についても合わせて検討していく。

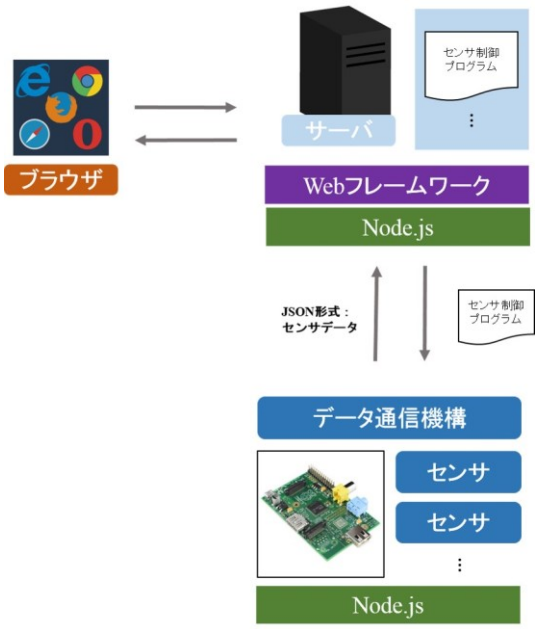
参考文献

1) Dominique Guinard and Vlad Trifa,「Towards the Web of Things:Web Mashups for Embedded Devices,」 Proc. of the Int’l Conf. on the 6th Networked Sensing Systems(INSS), pp.1-4, 2009.
2) 小玉哲平、小板隆浩(2010/11)「センサマッシュアップ構築支援機構の提案」電子情報通信学会
3) 熊谷康太、石川拓也、本田晋也、高田広章(2014/3/15)「組込みシステム向け軽量スクリプト言語へのアクセス制御機構の導入」情報処理学会研究報告
4) 松本亮介、岡部寿男(2013/12/13)「mod_mruby：スクリプト言語で高速かつ省メモリに拡張可能な Web サーバの機能拡張支援機構」インターネットと運用技術シンポジウム 2013
5) Node.js<http://nodejs.org>
6) Express< http://expressjs.com/>
7) RaspberryPi<http://www.raspberrypi.org>

正誤表

頁	箇所	正
2	4.2 システム構成　11 段目	ノードが増えた場合のプログラム の管理を容易にする。 図 1 にシステム構成を示す。

図 1　システム構成



頁	箇所	正
4	4.3.1　Sensor　21 段目	ボードで使用しているハードウェアに依存しないようにする。センサライブラリの構成を図 2 に示す。

図 2　センサライブラリ