IoT アプリケーションのソフトウェア・ハードウェアを単一コードベースで開発できる統合開発環境 f3.js

加藤 淳† 後藤 真孝†

概要：Internet of Things (IoT)アプリケーションのソフトウェア開発者にとって，センサやアクチュエータを配置する筐体の設計は CAD システムを用いる必要があり容易でない．本稿では，ソフトウェアの開発用インタフェース上でハードウェア設計まで一貫して行える統合開発環境 f3.js を提案する．f3.js では，センサやアクチュエータの制御コードと同じスコープに，レーザーカッターを用いた筐体設計のためのコードを書ける．また，レーザーカッターの切断面を組み合わせて立体形状を形成できるように，展開図の作成支援用 API を備えている．ユーザは展開図上にセンサやアクチュエータを配置する GUI プログラミングのようなコードで，新しいハードウェアインタフェースを設計できる．これらの仕組みを利用して，エンドユーザでも GUI を操作して IoT アプリケーションをパラメトリックにカスタマイズできるようになっている．提案手法の妥当性を，f3.js を用いた 5 名参加の小規模ワークショップで評価した．

f3.js: An Integrated Development Environment for Single Codebase Development of Both Software and Hardware of IoT Applications

JUN KATO† MASATAKA GOTO†

Abstract: During the development process of Internet of Things (IoT) applications, software developers need to learn how to use computer-aided design tools to create the enclosures. This paper proposes f3.js, an IDE that allows enclosure design within the code editor. It allows the programmer to write a single JavaScript codebase that defines not only the behavior of microcontrollers but also the layout of modules on laser-cut acrylic panels that form three-dimensional shapes. The IDE provides a set of GUI toolkit-like APIs to help creating development views for laser-cutting and GUI widgets that allow end-users to customize IoT applications by modifying parameters. Its five example applications are collected from a workshop with five participants.

1. はじめに

Internet of Things (IoT)アプリケーションは，センサやアクチュエータなどのモジュールをマイコンに接続して実世界で動作させることで，パーソナルコンピュータやスマートフォンのように標準化されたハードウェアでは難しい情報取得や提示を可能にするものである．その開発プロセスにおいて，ソフトウェアの開発とハードウェアの設計には通常別々のツールが用いられる．まず，ソフトウェア開発には，マイコン用の統合開発環境(Integrated Development Environment, IDE)が利用される．次に，ハードウェアの設計では，3D プリンタで 3 次元形状を直接出力したり，レーザーカッターで 2 次元の展開図を切断加工したりする必要がある．このような形状を効率的に設計するために CAD (Computer-Aided Design)システムが利用される．

ソフトウェア開発者が IoT アプリケーションのプロトタイピングにおいて筐体設計までは行おうとすると，3 つの問題に直面する．まず，CAD システムの学習コストが高く，学習しても別々のツールを使い分けるのは面倒である．また，筐体にはセンサやアクチュエータなどのモジュールを安定して把持するための穴などを開ける必要がある．箱型などのシンプルな形状を設計したいだけでも，CAD システムでのモデリングは容易でない．最後に，実際に組み立てて配線するために必要な機材が多く，手間も大きい．

† 産業技術総合研究所, National Institute of Advanced Industrial Science and Technology

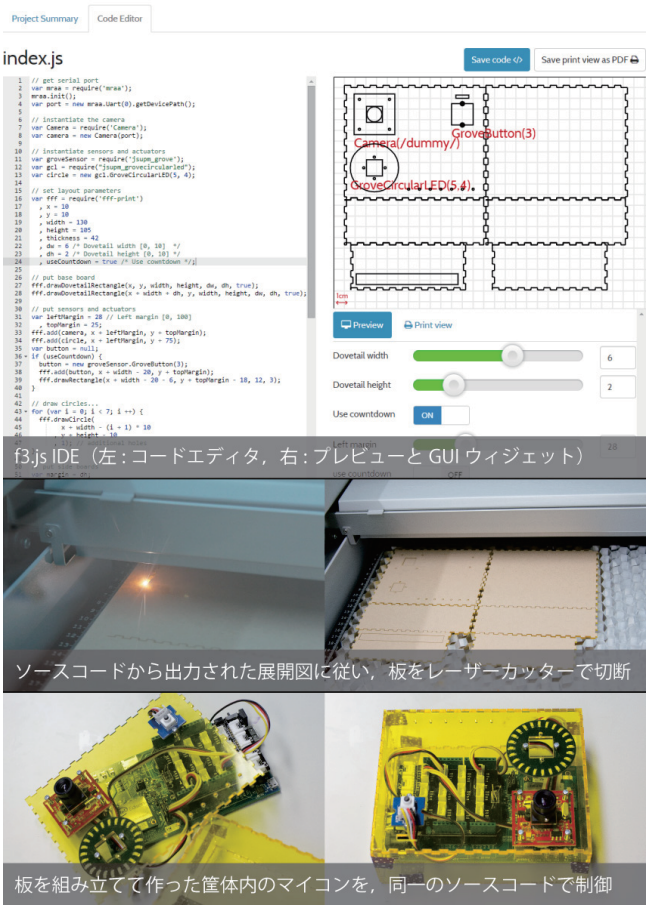

図 1. f3.js を利用した IoT アプリケーションの開発過程.

本稿では，GUI アプリケーションの開発プロセスを参考にした f3.js―**F**orm **F**ollows **F**unction() for **J**ava**S**cript―という IDE を提案し，上述した問題を解決・軽減することを目指す．GUI 設計では，プログラムのロジックと見た目は共にソースコードで定義されるため，ソフトウェア開発者は IDE を離れずにアプリケーション開発を行える．また，コードエディタの横に Interface Builder を配置して，インタフェースのプレビューを確認できる．これにより，開発者は実際にプログラムを実行することなくボタンなどのウィジェットの見た目を確認できる．

f3.js は，**図 1** に示したように単一の JavaScript ファイルを記述するだけで，IoT アプリケーションのソフトウェア(マイコンをどのモジュールと接続してどのような機能を実現するか)とハードウェア(モジュールをどのような立体形状のどの面に配置するか)を開発・設計できるようにする．ユーザは JavaScript プログラミングのみを学べばよく，学習コストを低く抑えられることが期待できる．また，f3.js が提供する API を利用すれば，GUI プログラミングのようなコードで新しいハードウェアのインタフェースを設計できる．設計した内容はコードエディタの横に常にプレビューが表示され，利用しているセンサやアクチュエータの外形と配置を確認できる．作成したレイアウトをもとにアクリル板を加工すると，センサやアクチュエータを設置するためのネジ穴が空くため，ユーザはネジとドライバ，接着剤だけで IoT アプリケーションを組み上げることができる．さらに，ソフトウェアとハードウェアが JavaScript の同一スコープ内で定義されるため，モジュールの有無，筐体上の配置や筐体の大きさ，これらのハードウェアに応じたソフトウェアの処理をパラメトリックに制御できる．f3.js は，この特長を活かし，プログラマでなくともスライダーなどの GUI ウィジェットを操作して IoT アプリケーションをカスタマイズ，出力できる機能を備えている．

以降，本稿では，関連研究との比較を行い，f3.js のインタラクションデザインと実装の概要を説明する．そして，著者 1 名を含む 5 名のプログラマによる試用例を紹介して，提案手法の妥当性を評価，議論する．

2. 関連研究

本章では，関連研究を紹介し，f3.js と比較することで，本研究の狙いと貢献を明らかにする．

2.1 ソフトウェアとハードウェアの同時開発支援ツール

2.2-2.3 節で紹介するように，さまざまな IDE が IoT アプリケーションのソフトウェア開発**または**ハードウェア設計のために提案されてきたが，両方を同時に行えるものは多くない．本節では，IDE に限らずソフトウェアとハードウェアを横断的に設計するための研究を紹介する．

2.1.1 IDE によるアプローチ

Microsoft Visual Studio は.NET Gadgeteer [2]と呼ばれるマイコンとセンサ・アクチュエータを繋ぐ配線を可視化する画面を持つ IDE である．画面上で配線を行うことで，対応するモジュールが特定の名前の変数に格納され，制御可能となる．また，市販の高級 CAD システムと連携して，利用しているモジュールを配置するための筐体制作を手助けする機能を試験的に実装している．

一方，本研究はソースコードのみで筐体設計まで可能であり，ツールを切り替えるメンタルワークロードがない．また，コードの変更結果が即座に反映される筐体のプレビュー画面を備えている．ただし，プレビュー画面上でマウスなどを利用した直接操作を行うことはできない．この点では，GUI アプリケーション開発に利用される Interface Builder を参考に改良が可能だと考えている．

Autodesk 123D Circuits [3]はブラウザ上で実行できる電子回路のシミュレーション環境で，実際に回路を組み上げなくても動作確認を行える．4 つのバリエーションがあり，Electronics Lab は Arduino マイコンと各種電子部品，Circuit Scribe は導電性インクと対応する独自モジュール，PCB Design はプリント基板と各種電子部品，MESH はソニー製 MESH モジュールのシミュレーションが可能である．

いずれもシミュレーションが主目的で，実際に試用できる新しいハードウェアのインタフェースを備えた IoT アプリケーションの作成には別途手間がかかる．本研究と最も近いのは Circuit Scribe で，配線やモジュールを点線で表した PDF ファイルを家庭用プリンタで印刷し，点線を導電性インクでなぞってモジュールを配置すれば IoT アプリケーションが得られる．同様のアプローチで配線の印刷までを自動化した PaperPulse [4]という研究もある．しかし，いずれも手描きで平面しかデザインできないため，本研究が可能にするプロシージャルな立体形状の設計とは一線を画する．なお，各種モジュールの機能をシミュレーションすることは f3.js でも技術的には可能であり，今後の課題である．

2.1.2 IDE 以外のアプローチ

IoT アプリケーションでは，求められる機能をどこまでハードウェアで実装し，残りをソフトウェアで処理するか選べることがある．その選択次第で，工数やコストを大きく削減できることがある．そこで，機能上の要求仕様とソフトウェア・ハードウェアが持つ制約条件を共に満たす設計を支援するための研究が多くなされている．

まず，ハードウェアの物理的な特徴を利用して，既存の電子部品が持つ一部の機能を代替する研究がある．Printed Optics [5]は，ハードウェアの設計を工夫して光路を制御し，光センサを用いてボタンや加速度センサなどの機能を実現できることを示した．PipeDream [6]は 3D プリンタで印刷された物体内部にパイプ状の空洞を開けて，音センサにより接触などのインタラクションを検知できることを示した．

また，電子部品を既存の物体に付与して機能を後付けする研究がある．Sauron [7]は低価格な画像センサを付与して

簡単な画像処理によるイベント検知を可能にした．Midas [8]は既存の物体に導電性のシートを貼りつけてタッチイベントの検知を可能にした．また，Mechanical Hijacking [9]は，人間用に設計されたボタンなどのユーザインタフェースの周りにアクチュエータを配置して，人間用のユーザインタフェースをプログラムから制御できるようにした．

これらの既存研究は，いずれも特定のセンサやアクチュエータを活用する手法であり，汎用の IoT アプリケーション用 IDE を目指す f3.js とは相補的な役割を果たすものである．例えば，Printed Optics をライブラリ化してレーザーカッターの展開図作成に利用したり，Sauron の画像処理部分をライブラリ化してイベント検知に利用したりできる．

2.2 2 次元・3 次元形状のための IDE

DressCode [10]はデザイナーが 2 次元的な形状を自由に描ける Domain Specific Language (DSL)のための IDE である．ソースコードを入力すると即座に形状を見ることができるライブプログラミングを実現している．OpenSCAD [11]は 3D プリンタに出力できる 3 次元形状に関して同様のことができる DSL および IDE であり，CAD システムの GUI が持つような設計支援機能を API として提供している．ShapeJS [12]は IDE ではないが，同様の目的で JavaScript 用の API を提供しているライブラリである．

f3.js は，DSL ではなく汎用言語の JavaScript を用いて，IoT アプリケーションの筐体を設計可能な API を提供している IDE である．上述した既存研究との最大の違いは，ハードウェアの形状のみならずソフトウェアの開発まで可能な点にある．その目的に供するためには，DSL ではなく汎用言語を利用することが妥当と考えた．

また，センサやアクチュエータなどのモジュールを配置した新しいハードウェアのインタフェースを開発するためには，2 次元的な形状設計機能は役不足である一方，3 次元形状のモデリングは容易でない．3 次元のコンピュータグラフィクスに関する基礎知識が要求される他，モジュールの設計情報を取り込んでネジ穴などの正しい向きを計算する必要がある．さらに，3D プリンタによる成型では，中に配線を通す中空の構造を作りづらいことがある．プリンタの精度が不十分で，ネジ穴が設計通り開かないこともある．

そこで本研究では，高精度に面を切り出せるレーザーカッターを利用して，複数の面を組み合わせて筐体を作成することとした．そのために，面同士を繋ぐ「ありつぎ」を出力したり，面を押し出して作った立体形状から展開図を作成したりする API を提供している．この展開図上にモジュールを配置することで，画面上にボタンを配置するように，GUI プログラミングと同様の使い勝手で筐体を設計できることを目指している．これは，絶対座標系が存在しない 3 次元の実世界において，天井カメラの画角内で床面上に座標系を張り，GUI プログラミングのように物体やロボットの移動を支持できるようにしたプロトタイピング用ツールキット Phybots [13]と同様の設計方針である．

2.3 フィジカルコンピューティングの開発支援ツール

Phidgets [14]はセンサやアクチュエータなどのモジュールをパソコンに USB 接続できるようパッケージ化し，はんだ付けのいらないフィジカルコンピューティングを実現したライブラリである．パソコンを使わずマイコンのみで動作するような IoT アプリケーションの開発は，Arduino [15]の登場以降，急速に広まった．マイコンの IDE と仕様をオープンソースで公開したことにより，デファクトスタンダード化している．Grove システム[16]は，Phidgets と同様に各種モジュールをマイコンへ接続しやすいようパッケージ化したものである．このようにデファクトスタンダード化，モジュール化が進むと，ハードウェアの設計情報を容易に入手できるようになり，3D プリンタやレーザーカッターによる筐体設計が捗る．本研究では，Web 上に公開されている Grove モジュールの設計情報を元に，ネジ穴や配線口を容易に開けられる API を実装し，提供している．

近年のマイコン用 IDE は，Codebender [17], mbed Compiler [18]のようにブラウザ上で動作するものが多くなってきている．また，多くのマイコンが無線 LAN 経由でインターネットに接続できる．例えば f3.js がサポートしている Edison の他にも，JavaScript で動作する Tessel.io [19]や Arduino Yún などがある．マイコン上で動作するプログラムの書き換えまで無線で行えるものは多くないが，Edison はその一例であり，今後増えていくことが考えられる．将来は，f3.js や TouchDevelop [20]のようにブラウザ上で動作する IDE を利用して，PC レスで IoT アプリケーションをライブプログラミングできるようになるだろう．

3. f3.js: Form Follows Function()

本章では，f3.js を用いて IoT アプリケーションのソフトウェアとハードウェアをシームレスに開発・設計するプロセスを紹介する．アプリケーションの具体例として，**図 1** に写っているカメラアプリケーションを取り上げる．このカメラは，ボタンが押されたら円周上に配置された LED がカウントダウンを始め，カウントダウン終了時に写真を撮影するようになっている．筐体右上にカメラが，右下に LED があり，左上にボタンがある．これらの振る舞いと筐体設計を共に定義している**両義的ソースコード**の具体例(全文)については http://f3js.org に掲載してある．

3.1 マイコン用のプログラム開発

f3.js は Intel Edison [1]マイコンを無線または有線で接続した Windows または Mac OS X 搭載パソコンで動作する Web サーバとして実装されており，アプリケーションのソースコードは**図 2** に示すように処理される．ユーザは Web ブラウザを搭載したパソコンやタブレットなどの端末から f3.js を利用する．f3.js にアクセスすると，パソコンのファイルやディレクトリ構造を閲覧できるファイルブラウザが

表示され，任意のディレクトリへ移動できる．ここで，アプリケーション開発のためのディレクトリを新規作成するか，すでにプロジェクトが保存されたディレクトリに移動すると，プログラミングを始めることができる．

Edison のプログラムは JavaScript のソースコードとして記述する必要があり，マイコンに接続したセンサやアクチュエータは Node.js モジュールを読み込んでインスタンス化することで制御できる．個々の Node.js モジュールが，いわゆるドライバの役割を果たしている．例えば，カメラは `var Camera = require('Camera')`のようにドライバモジュールを読み込み，`var c = new Camera(PORT_A)`のようにインスタンス化すると `c.takePicture('a.jpg')` のようにメソッドを読み込んで制御を行える．センサやアクチュエータ用の Node.js モジュールは UPM という GitHub 上のリポジトリで一元管理されており，誰でも pull request を送ることで新しいドライバを書いて他の人と共有できる．

3.2 センサ・アクチュエータの接続と動作確認

ユーザは，ソースコードを記述したら，Edison にセンサ・アクチュエータを接続して，f3.js 上の Install & Run ボタンをクリックすることで実機での動作を確認できる(**図 2** 右側)．もし動作に不具合があれば，ソースコードを修正して再度 Install & Run をクリックしてデバッグできる．

従来の IDE はここまでのプロセスのみを開発支援するケースがほとんどである．開発初期段階のプロトタイピングでは，ブレッドボードなどを利用して配線さえできていれば動作確認が可能なこともある．しかし，実用的な IoT アプリケーションをテストする際は，ある程度の強度を持つ筐体にセンサやアクチュエータの位置が固定されていないと，センサの値が不安定になったり，アクチュエータの出力が十分に伝わらなかったりすることが多い．例えば，カメラの場合は画角が固定できず，思い通りの撮像が得られるか確認できない．

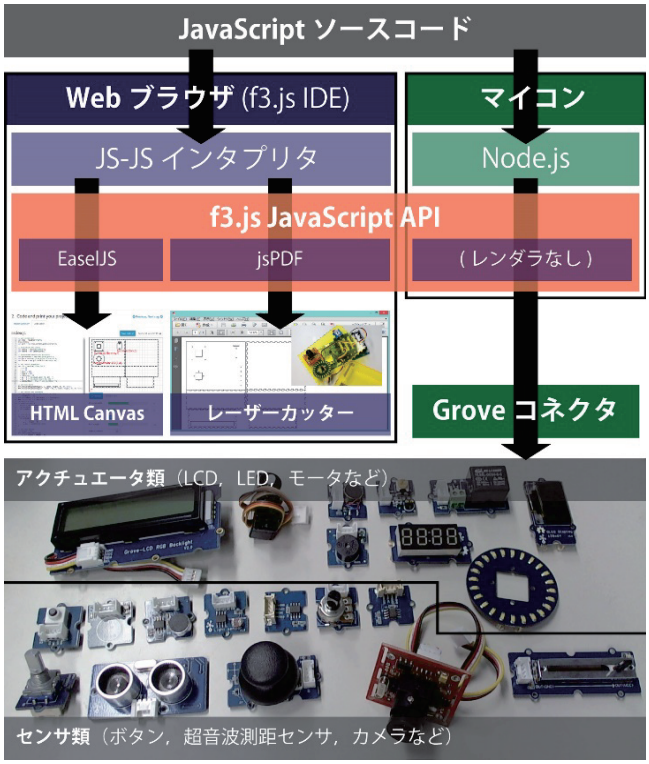

図 2. f3.js のシステム構成図．Web ブラウザでは独自の JavaScript インタプリタを，マイコンでは Node.js を用い，それぞれに共通 API を持つが実装の異なる f3.js ライブラリを提供することで**両義的ソースコード**を実現している．

3.3 レーザーカッターの切断面設計

f3.js では，マイコン用のプログラム開発だけでなく，筐体設計も同一のソースコード中で JavaScript を用いて行える(**図 2** 左側)．画面上では，左にソースコードが，右の HTML Canvas にレーザーカッターに出力されるレイアウトのプレビューが表示される．プレビューはソースコードを変更するたびに自動的に更新されるため，作成したい形状が得られているか随時確認しながら開発を行うライブプログラミングが可能である．また，プレビューには，レーザーカッターでアクリル板などを切断加工して筐体を作るために必要な切断線だけでなく，ユーザの利便性に配慮して筐体に載せるセンサやアクチュエータの名前や外形も重畳表示される．Print Preview ボタンをクリックすると切断線のみの表示に切り替わる．

f3.js では，マイコン用のプログラムと同一のソースコード中で筐体設計を行えるため，利用しているセンサやアクチュエータの種類や数といった情報を再度 CAD システムに入力するような手間がかからない．その代わり，一回インスタンス化したドライバのインスタンスを特別な API に渡すことで，対応するセンサやアクチュエータに適したネジ穴や配線口を開けることができる．具体的には，`var fff = require('fff-print')`のようにして API エンドポイントを取得し，以降，そのメソッドを呼び出すことで筐体設計を行うことができる．例えば，`add(c, 30, 40)`のようにカメラドライバのインスタンスを渡すことで，左上から 30mm, 40mm の位置を基準にしてカメラを固定するためのネジ穴と配線口を開けることができる．

筐体は通常，立体的な形状をしており，複数の切断面を組み合わせる必要がある．レーザーカッターの切断線は，切断面を複数並べた展開図となることが一般的である．このような展開図は従来手動で作成する必要があったが，f3.js では作成を容易にする API が用意されている．例えば `fff.drawDovetailLine(5, 5, 40, 40)`のように書くことで，左上から 5mm, 5mm の位置を開始点として 40mm, 40mm の位置まで凸凹した線が引かれる．これを別の面の凸凹した線と噛み合わせることを「ありつぎ(dovetail)」と呼び，ありつぎを複数組み合わせることで立体形状を作れる．しかし，正しくかみ合うありつぎを作ることは容易ではない．そこで，閉じた平面の領域を `var p =`

`fff.createPath(); p.moveTo(5, 5).dovetailTo(40, 40)….close()`のように作成してから `p.extrude(30)`のように 30mm 押し出し，展開図に含まれる閉じた平面の領域を配列で取得できる．

なお，多くのレーザーカッターは切断の他に表面を浅く彫る彫刻加工が可能であり，f3.js でも定義した平面の領域に対して `p.fill = true` のように記述すると彫刻を行うようになっている．さらに切断を行わない場合は `p.stroke = true` と書けばよく，GUI アプリケーションの開発と同様にしてレーザーカッターの動作を制御できるようになっている．

3.4 IoT アプリケーションの組み立て

切断面の設計が終わったら，Save as a PDF file ボタンをクリックすると，一般的なレーザーカッターに直接送信できる PDF ファイルが生成される．なお，一般的なレーザーカッターは，ある程度以上細い特定の色の線を切断し，特定の色の塗りつぶしを彫刻する仕様になっている．本システムは，切断線は 0.005mm の太さの黒い線として，彫刻面は赤色(RGB=255,0,0)の塗りつぶしとして出力しており，Trotec 社の Rajyet レーザー加工機で動作確認を行っている．切断加工が済んだアクリルなどの材質の板は，ありつぎ同士を組み合わせ，接着して立体形状にできる．

本システムが初めから対応しているセンサ・アクチュエータは，いずれもネジ穴のついたモジュールとして販売されている．これらは，ネジとビスを利用していずれかの板に固定することができる．また，これらのセンサ・アクチュエータは特定の形状のコネクタで配線でき，はんだ付けせずに組み立て可能である．このように，ネジ穴やコネクタが標準化されたセンサ・アクチュエータのモジュールに加え，その設計情報を提供する API があれば「特別な工具が不要で，まるでプラモデルのように IoT アプリケーションのプロトタイピングが可能」(5 章で紹介するワークショップ参加者のコメントより抜粋)となる．

3.5 エンドユーザによるカスタマイズ

f3.js は JavaScript のプログラムを実行することで切断面を生成しているため，与えるパラメタを変更してさまざまな形状をパラメトリックに生成することができる．f3.js では，この特性を活かして，変数宣言の直後に特定の書式でコメントを挟むと，パラメタを変更するための GUI ウィジェットを表示するようになっている．

例えば `var useCountdown = true /*` カウントダウンタイマーを使う `*/`という真偽値ではチェックボックスを，`var capacity = 40 /*` 容積`[0-100] */`という値域のついた数値の場合はスライダーを表示する．このようなウィジェットはプログラマでなくとも操作できるため，基本となるプログラムをプログラマが作成し，それをもとに誰もがニーズに合わせてカスタマイズを施した IoT アプリケーションを入手できるようになる．この際，形状と機能の両方を同一のソースコードで変更しているため，例えばカウントダウンタイマーを使う指定が入っていたらそのためのネジ穴を開けると共に，カウントダウンの処理を挟む，といった対応が容易にできるようになっている．

4. f3.js の実装

f3.js は，**図 2** に示したような処理系を実現するために，TypeScript で記述されたサーバとクライアントの組み合わせで実装されている．本章ではその概要と，利用できるハードウェアについて紹介する．

4.1 f3.js サーバ

f3.js サーバは Node.js を利用した HTTP サーバとして実装されている．ソースコードのほぼ全てがクロスプラットフォームで記述されているが，対応するマイコンへのプログラムの転送時に利用する子プロセスの立ち上げ処理や，OS 固有のシェル（Windows の cmd.exe や Mac OS X の bash など）を通したコマンド操作をサポートするために，OS 依存の Node.js ライブラリを利用している．

f3.js クライアントがサーバに接続した際は，f3.js は静的な HTML/CSS/JavaScript ファイルを配信する．また，同時に Socket.io ライブラリを用いて WebSocket によるストリーミング接続が開始される．この接続は，シェルと CUI ベースでインタラクティブに通信するため，また，マイコン上で動作するプログラムからの応答をリアルタイムに配信するために利用される．この他に，f3.js サーバは，クライアント側の要求に応じてサーバ上のファイルを一覧したり編集したりできるよう，ファイルマネージャの機能を提供する Web API を実装している．

4.2 f3.js クライアント

f3.js クライアントも TypeScript で開発され, JavaScript にコンパイルされたものが f3.js サーバにより静的ファイルとして接続端末のブラウザに配信される．**図 2** 左側に示したような筐体設計の機能をブラウザ上で実現するために，JavaScript の独自インタプリタを実装している．このインタプリタは JavaScript の基本的な文法は完全にサポートしているが，ブラウザが備えているような API はほとんど提供しない．その代わり，未定義の関数は無視して実行を続ける仕組みになっている．

また，このインタプリタ上で f3.js API が呼ばれた際に，筐体設計のプレビューを表示するために EaselJS という描画ライブラリを利用し，レーザーカッターへ送出できる PDF データを生成するために jsPDF という描画ライブラリを利用している．

4.3 対応ハードウェア

f3.js は，現時点で，JavaScript でプログラミングできるマイコンである Intel Edison [1]に対応している．Edison は組み込み Linux が動作する高性能なマイコンであり，f3.js からは TCP/IP (有線または無線)で SSH 接続して JavaScript の

ソースコードを直接転送する．Edison は Arduino Breakout Board Kit と呼ばれる基板を接続し，さらに Arduino 用の拡張基板である Grove シールドを搭載することで，I/O ポートが標準化された Grove システム[16]のモジュールを利用できるようになる．Grove モジュールは全てグラウンド・電源・RX・TX の 4 線で接続できるようパッケージ化されたセンサやアクチュエータであり，公式 Wiki で物理的なレイアウトの情報などが公開されている特徴がある．

f3.js では，これらの情報を利用して現在 21 種類の Grove モジュールに対してネジ穴や配線口を開けるための API を提供している．ユーザが他のモジュール用に API を実装することもできる．実際に，5 章の小規模ワークショップでは参加者により 3 つのセンサへの対応が追加された．

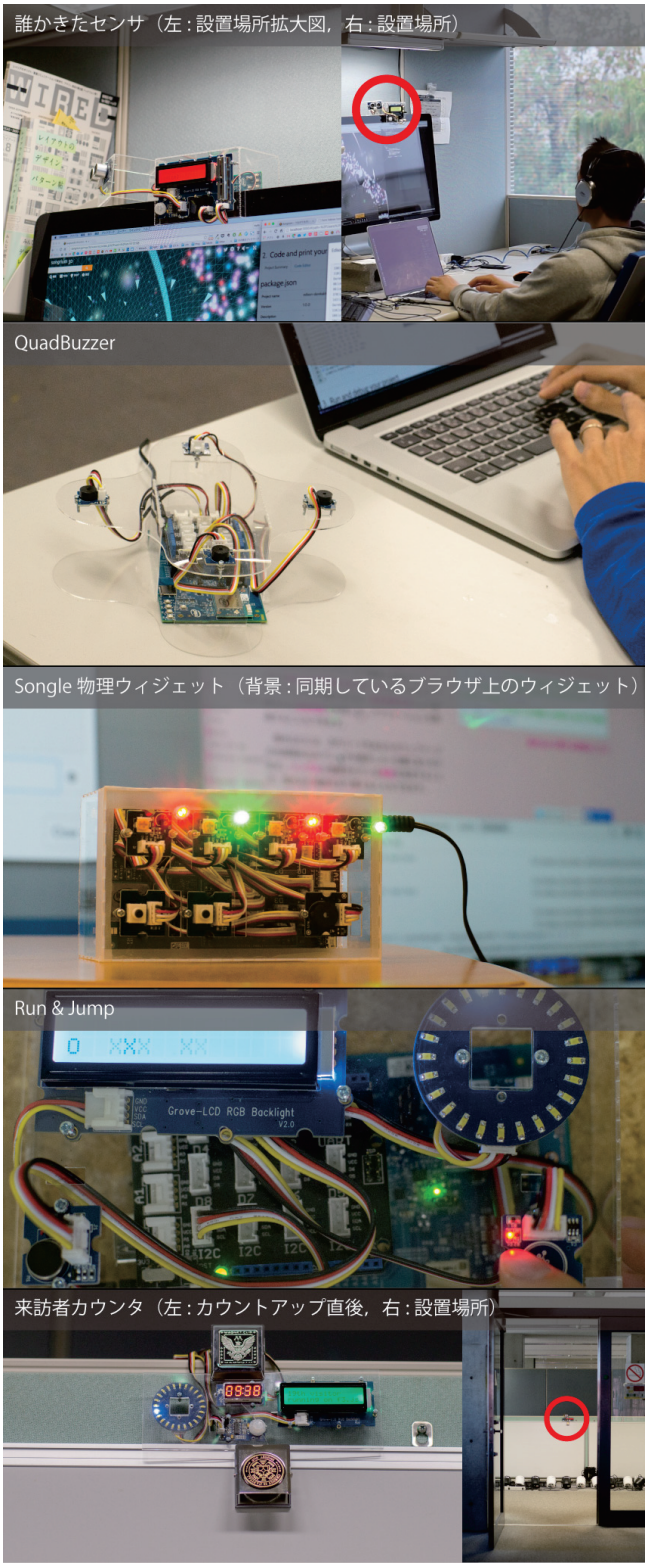

図 3. ワークショップで得られた作例．

5. 小規模ワークショップ

本章では，f3.js の評価のために行った小規模なワークショップの方法と参加者が作成した IoT アプリケーション(**図 3**)を紹介し，ワークショップ後に得られたコメントを引用しながら，f3.js の諸機能について議論する．

5.1 ワークショップの開催方法

ワークショップは，著者 1 名と 4 名の参加者で行った．

著者および 3 名の参加者は Arduino などを用いたハードウェアのプロトタイピングの経験があり，1 名は未経験であった．また，全員 JavaScript によるプログラミング経験者であった．

各々に Intel Edison の初期設定と f3.js の利用方法に関するイントロダクションを行い，その後の 1 週間で好きな時間にアイデア出しと実装を行ってもらった．著者はワークショップ用ブースに常駐し，随時，参加者からの Edison や f3.js に関する質問に答えた．ワークショップに参加した時間は人により 2 時間〜10 時間程度のばらつきがあったが，全員が自分好みの IoT アプリケーションを作成できた．

5.2 IoT アプリケーション

5.2.1 だれかきたセンサ (制作時間約 10 時間)

個人ブースで作業していると，誰かが相談ごとなどでブースの入り口にきても気づかず，いきなり肩を叩かれてびっくりすることがある．そこで，ブースに人が入ってきたことを検知して音と光でユーザに通知し，余裕を持って対応できるようにした．Twitter に流れるキーワードに反応して音を鳴らすエゴサーチ機能もある．ハードウェアのプロトタイピング未経験者によるもの．

筐体は，利用しているコンピュータのディスプレイ上部にぴったりはまるように設計され，バックライト付き LCD ディスプレイ，ブザー，音量調節つまみが取り付けられている．また，ディスプレイ裏面側に Edison が隠れるように収納されている．さらに，小さな箱に距離センサが取り付けられ，ブース入り口に向けて置かれている．

5.2.2 QuadBuzzer (制作時間約 6 時間)

PC/Mac とリアルタイムに通信し，マウスやキーボード入力，閲覧している Web ページのデータなど，あらゆるデータを 4 つのブザーが奏でる音楽に変換する楽器を作成した．

筐体は滑らかな曲線で縁取られた上面の 4 隅にブザーが対称的に配置され，Edison はその下に置かれている．

5.2.3 Songle **物理ウィジェット(制作時間約 8 時間)**

ブラウザ上で動作する音楽プレイヤーとリアルタイムに通信し，同期して光ったり，物理ボタンで音楽プレイヤーを操作したりできるようにした．

筐体は, Edison を格納した直方体上面にボタンや LED などのインタフェースを配置したシンプルな形をしている．

5.2.4 Run & Jump **(制作時間約 2 時間)**

バックライト付き LCD ディスプレイに表示された自機「o」をタッチ操作でジャンプさせ，ディスプレイ右端から流れてくる障害物「x」を避けるゲームを実装した．Songle 物理ウィジェットと同様に，直方体の上面にインタフェースが集中している．ゲームオーバー時には振動モータが振動して演出を盛り上げる．

5.2.5 来訪者カウンタ (制作時間約 3 時間)

人感センサにより部屋に入室した人数と時刻を記録する．また，入室時には何人目の入室か LCD に表示する他，常時デジタル時計が表示されている．筐体は一枚のアクリル板のみでできており，マグネットを使って磁性のある壁に貼り付けている．著者による制作物．

5.3 考察

ワークショップ参加者は全員 JavaScript を用いたプログラミング経験があったため，API リファレンスを参照しながら，途中で行き詰まることなく IoT アプリケーションのソフトウェア開発とハードウェア設計ができていた. 以降, 5.2.1-4 小節の(著者以外の)作者を P1-4 と表記する．本節の 5.3.1-4 小節は 1 章で述べた研究上の課題と次のように対応しており，課題がどのように解決されたか議論している．

1. 高い学習コストと IDE/CAD ツール切り替えのメンタルワークロードの高さ (5.3.1-2 小節)
2. モジュールを把持するための立体形状を平面から組み立てられるよう設計する難しさ (5.3.3-4 小節)
3. 組み立てて配線する手間の大きさ (5.3.5 小節)

5.3.1 両義的ソースコードによるシンプルな開発プロセス

一つのソースコードを書くとソフトウェアとハードウェアが生成されることに関しては,「ソースコードを書いてレーザープリンターで出力すれば終わりという，ほとんどの作業が PC 上で済むことが，自分のように電子工作よりプログラミングが得意な人にとってはかなりハードルを下げてくれる(P4)」など, 全員から肯定的な感想が得られた．

アプリケーションに関する情報をすべて両義的ソースコードで指示するシンプルなワークフローに起因すると考えられる「ユーザインタフェースが綺麗にまとまっていて使いやすかった(P4)」というコメントもあった．

5.3.2 ソフトウェア開発との親和性の高さ

「JavaScript(+ npm で公開されているライブラリ)を使って組めるのは Web エンジニアにとって Arduino 等他の環境と比べて敷居が低く(P2)」「Web ブラウザ上のユーザインタフェースから Install & Run できて，デバッグ出力もブラウザから確認できるため開発しやすかった(P3)」という好意的なコメントが得られた．

一方で「どんな API が使えるのかが分かりにくかった(P3)」というコード補完への要望や、「全画面化(P2)」などエディタに関する機能増強の要望があり，f3.js における文字ベースのプログラミング体験の重要性が確認できた．

5.3.3 2.5 **次元的な筐体設計のための** IDE

筐体を開発できるプログラミングについては，「最適な形状をすぐ作れるのがとても楽しかった．Arduino で開発したとき，無理やり名刺ケースをカットしようとしていびつな形になったり，アクリルにネジ穴を開けようとしてひび割れたりしたことがあったので，比べると大変楽(P4)」で，f3.js の支援機能で初めて筐体を容易に作れたというコメントが全員から得られた．とくに「平面から立体を作成する機能はとても助かった. なかったら無理(P1)」だったという．GUI プログラミングに近い API でさまざまな立体形状を作れる 2.5 次元的な筐体設計ができており，シンプルな直方体(P3,4)から曲線で縁取られた箱(P2)まで，バリエーションに富んだ形状が得られた．

また，ソースコードで筐体を設計する利点として，対称的な形状を効率的に生成できることが挙げられる他,「他の人が作ったものを，そのままコピー&ペーストできるのが便利(P3)」だという．また,「Edison にかぎらず，通常の電子工作でも使いたい．ふだんはガムテープで済ませてしまうショート防止の代わりに，よく使われるサイズのボードに合わせた筐体を(スライダーで大きさを調節して)作りたいと思う(P3)」そうで，GUI で形状などをパラメトリックにカスタマイズできる機能への需要が確認できた．実際，レーザーカッター用に好きなサイズの直方体の展開図を作成できるだけの Web サービス aが「1 日 100 回以上箱を生成している」など人気を博しているようである．

さらに，Grove モジュールに対応したネジ穴や配線口などが API で提供され，ユーザ自身でも自由に拡張できることについては,「部品の配置などのモジュールを共有できてよい(P3)」という好意的なコメントが得られた.「超音波センサの図面テンプレートをビスの位置など確認しながら作るのを楽しめた (P1)」ことから，f3.js を Web 上でホストして，さまざまな人が協力して強化していける TextAlive [21] のようにソーシャルな IDE にする将来展望が考えられる．

5.3.4 筐体設計のライブプログラミング

ソースコードが随時実行され，筐体設計のための切断線などがプレビューされる機能については「各モジュールを配置した形でケース図面を見られるのは完成予想図を想像しやすくて良かった(P3)」「ケース図面が表示されると早く

a Make-a-box コメント．Retrieved November 1, 2015 from http://makeabox.io

プリントアウトしたくなるため，完成までのモチベーション維持に良い．やはりケースまで完成するとやりとげた感がある(P1)」というように大いに歓迎された．また，「将来的には 1 行更新するごとに(マイコン上のソフトウェアも)すぐに更新して実行できるようなオプションがあるとよいかもしれない」というように，マイコンそのもののライブプログラミングに関する期待も見られた．

一方で，現状は切断線などの設計をすべてソースコードで記述しているが，「GUI で切断線を編集する機能までは要らないが，クリックするとその座標がソースに入力されるような補助機能は欲しい(P4)」「GUI で直接操作したい(P1,2,3)」というコメントがあった．関連研究でも触れたように Interface Builder のような直接操作の機能が必要であることが確認された．また， 123D Circuit [3]のように回路上のセンサやアクチュエータのシミュレーションもできれば,ライブプログラミングの体験が更に便利になるだろう．

5.3.5 プラモデルのような IoT アプリケーション開発

「入れ物の組み立てがプラモデル的で簡単だった(P3)」というコメントに象徴されるように，はんだ付け不要で接着剤だけを使ってハードウェアを組み立てられる工程はとても好評だった．その上で,「凹凸の辺を組み合わせるだけでなく，何らかの形で嵌めこむだけで自立するような，小学生向けの玩具のような仕組みができれば(P3,4)」という接着剤すら不要で組み立てられる仕組みへの要望が見られた．

また，「プリントアウトされた板の裏表が分からなくなる(P1,3)」問題が明らかになったが，隣り合わせになる面には相対する矢印を彫刻するなど，レーザーカッターの出力を工夫すれば組み立て工程をさらに支援できるだろう．

6. おわりに

本研究では，レーザーカッターで切断したアクリル板などを組み合わせ，センサやアクチュエータを配置して新しいハードウェアのインタフェースを備えたマイコンアプリケーションを制作できる統合開発環境 f3.js を開発した．f3.js において導入された**両義的ソースコード**は，センサやアクチュエータの利用情報をアプリケーションの形状と機能の記述において共有することで，開発にかかる労力を削減できる．筐体設計のための **2.5 次元的な API** は GUI プログラミングなどに親しんだソフトウェアエンジニアにも使いやすく，学習コストを下げることができる．さらに，ソースコードで全てを記述することにより,利用するセンサ・アクチュエータに応じてソフトウェアの振る舞いと筐体形状の両方を変えるなど，アプリケーションの**パラメトリックなカスタマイズ**を実現している．

今後は，これらの特徴を活かしながら，ワークショップで得られたフィードバックを元に改善を重ねていきたい．また，http://f3js.org で f3.js を一般公開し，多くの人に利用してもらいたいと考えている．

謝辞 本研究の一部は JST CREST の支援を受けた．また，5 章のワークショップには石田 啓介氏，川崎 裕太氏，井上 隆広氏，矢倉 大夢氏にご参加いただいた．ここに感謝の意を表する．

参考文献

1) Intel. Intel Edison. 2014. Retrieved April 1, 2015 from https://www.intel.com/content/www/do-it-yourself/edison.html
2) Nicolas Villar, James Scott, Steve Hodges, Kerry Hammil, and Colin Miller. .NET gadgeteer: a platform for custom devices. In Proc. of Pervasive'12, 216-233.
3) Autodesk. Autodesk 123D Circuits. 2015. Retrieved November 1, 2015 from http://www.123dapp.com/circuits
4) Raf Ramakers, Kashyap Todi, Kris Luyten. PaperPulse: An Integrated Approach for Embedding Electronics in Paper Designs. In Proc. of CHI'15, 2457-2466.
5) Karl Willis, Eric Brockmeyer, Scott Hudson, and Ivan Poupyrev. Printed optics: 3D printing of embedded optical elements for interactive devices. In Proc. of UIST '12, 589-598.
6) Valkyrie Savage, Ryan Schmidt, Tovi Grossman, George Fitzmaurice, and Björn Hartmann. A series of tubes: adding interactivity to 3D prints using internal pipes. In Proc. of UIST '14, 3-12.
7) Valkyrie Savage, Colin Chang, and Björn Hartmann. Sauron: embedded single-camera sensing of printed physical user interfaces. In Proc. of UIST '13, 447-456.
8) Valkyrie Savage, Xiaohan Zhang, and Björn Hartmann. Midas: fabricating custom capacitive touch sensors to prototype interactive objects. In Proc. of UIST '12, 579-588.
9) Scott Davidoff, Nicolas Villar, Alex S. Taylor, and Shahram Izadi. Mechanical hijacking: how robots can accelerate UbiComp deployments. In Proc. of UbiComp '11, 267-270.
10) Jacobs, J., and Buechley, L. Codeable Objects: Computational Design and Digital Fabrication for Novice Programmers. In Proc. of CHI'13, 1589-1598.
11) OpenSCAD. 2015. Retrieved November 1, 2015 from http://www.openscad.org/
12) Shapeways. ShapeJS. 2015. Retrieved November 1, 2015 from http://shapejs.shapeways.com
13) Jun Kato, Daisuke Sakamoto, Takeo Igarashi. Phybots: a toolkit for making robotic things. In Proc. of DIS'12, 248-257.
14) Saul Greenberg and Chester Fitchett. 2001. Phidgets: easy development of physical interfaces through physical widgets. In Proc. of UIST'01, 209-218.
15) Arduino. 2015. Retrieved November 1, 2015 from http://arduino.cc
16) SeeedStudio. Grove System. 2015. Retrieved November 1, 2015 from http://www.seeedstudio.com/wiki/GROVE_System
17) Codebender. 2015. Retrieved November 1, 2015 from https://codebender.cc
18) mbed Compiler. 2015. Retrieved November 1, 2015 from https://developer.mbed.org/compiler
19) Technical Machine. Tessel. 2014. Retrieved November 1, 2015 from https://tessel.io
20) Sebastian Burckhardt et al.. It's Alive! Continuous Feedback in UI Programming. In Proc. of PLDI'13, 95-104.
21) Jun Kato, Tomoyasu Nakano, Masataka Goto. TextAlive: Integrated Design Environment for Kinetic Typography. In Proc. of CHI'15, 3403-3412