Vol.2016-GN-97 No.18
Vol.2016-CDS-15 No.18
Vol.2016-DCC-12 No.18
2016/1/21

スマート端末と周辺デバイスを簡単につなげる Web ドライバ技術

二村和明[†1]　伊藤栄信[†1]　坂本拓也[†1]　中村洋介[†1]

概要：従来，スマート端末で周辺デバイスを利用するためには，OS や周辺デバイスごとに専用アプリケーションが必要で，利用者はアプリケーションのインストール，開発者は OS や周辺デバイスごとのアプリケーション開発が必要となり，利便性と開発コストの点で課題があった．そこで，Web アプリケーションから直接周辺デバイスの制御を可能にして，クラウドサービスと周辺デバイスの接続を自由に組み合わせられる Web ドライバ技術を開発した．これにより，サービス事業者やデバイスメーカーは OS に依存しないアプリケーションやドライバ開発が可能になるとともに，サービス利用者は，周辺にあるデバイスを即座にスマート端末につないで活用可能になる．

キーワード：スマートフォン，周辺デバイス，Web runtime,，HTML5，JavaScript，デバイスドライバ

Web Driver Technology for Freely Connecting a Smartphone to Peripheral Devices

KAZUAKI NIMURA[†1]　HIDENOBU ITO[†1]
TAKUYA SAKAMOTO[†1]　YOUSUKE NAKAMURA[†1]

Abstract: Conventionally, connecting peripheral devices to smartphones requires a dedicated application on the smart device for each device, with different versions for different OS. This gave rise to issues of usability as users needed to install applications, and development cost that was necessary for developers to create applications for each OS and device. We have developed technology that enables Web applications run on smartphones to control devices directly, so that cloud services and devices can be easily connected and combined. This enables service providers and device makers to develop applications and drivers that are not tied to a particular operating system, and enables service users to instantly connect these devices to their mobile smart device.

Keywords: Smartphone, Peripheral device, Web runtime, HTML5, Java Script, Device driver

1. はじめに

スマートフォンやタブレットなどのスマート端末が急速に普及しており，多くの人が常時無線接続可能なコンピューティング環境を利用するようになった．無線接続可能な周辺デバイスも多数存在し，これらをスマート端末から頻繁に活用する利用シーンが今後増えると考えられる．しかし，スマート端末で，周辺デバイスを利用するためには，OS や周辺デバイスごとに専用のドライバアプリケーションが必要で，利用者はアプリケーションのインストールが必要となり，必ずしも利便性が良いとは言えない状況にある．また，スマートフォンのようにユーザが常に携帯して移動し，さまざまな周辺デバイスに遭遇する状況で，いちいち周辺デバイス利用のための準備を行うことは煩わしい作業となり，折角の利用タイミングを逃すことにもなりかねない．このような利用シーンで動的に周辺デバイスを利用できるプラグアンドプレイ的な機能があれば，ユーザ負担を減らすとともに周辺デバイスを使ったサービスの利用機会を増やすことが可能になるであろう．

周辺デバイスの活用は，スマート端末上のアプリからだけでなく，クラウドサービスと連携して活用する場合も考えられる．例えば，周辺デバイスで得られたデータをクラウドストレージサービスに保存し，別のサービスから活用するといった使い方である．このようなクラウドサービスは HTML や JavaScript などの Web 技術で構築される場合が多い．Web 技術はオープンかつ OS 非依存であり，マルチプラットフォーム上でインターオペラビリティを保ったシステムを作ることが可能であるという特長を持つ．もし周辺デバイスを HTML/JavaScript により利用できるようになると，同じ言語でクラウドサービスと周辺デバイスを直接つなげることができるようになることから，サイバーとフィジカルの世界をより接近させることが可能になる．

一般的にクラウドサービス開発とデバイスおよびデバイスドライバ開発のライフサイクルが異なるため，お互いの依存関係が疎になることが望ましい．すなわち，サービスが利用を期待するデバイスへのアクセス機能を必要に応じて後から付加して利用できるように，サービス記述とデバイス記述を分離・結合できるようにするような仕組みを確立しておくことが，今後の Web サービス開発に対する重要な変革をもたらす鍵になると考えられる．

そこで我々は，HTML/JavaScript で書かれた機器制御モジュール（本稿では Web ドライバと呼ぶ）を動的に挿抜でき，必要なタイミングで Web アプリと紐付けて実行するこ

†1 (株)富士通研究所
Fujitsu Laboratories Ltd..

Vol.2016-GN-97 No.18
Vol.2016-CDS-15 No.18
Vol.2016-DCC-12 No.18
2016/1/21

とが可能な Web ドライバアーキテクチャ(WDA)を提案する．これは，サービス，OS，周辺デバイスの間を疎な関係にするとともに,動的に紐付けて利用できる仕組みである．これによって，以下の三者にメリットがもたらされる．先ずユーザは，煩わしい操作や設定を必要とすることなく，その場にあるデバイスを，スマートフォンから簡単に利用することが可能になる．デバイス開発業者は一つのドライバを書くだけであらゆる端末で動作させることができるようになる．そして，クラウドサービス業者は，スマート端末経由でクラウドと周辺機器を結びつけたサービスを提供することができるようになる．

本論文では，Web ドライバアーキテクチャを実現するシステムの基本体系を示すと共に，実装方法についても示した. さらに市販の周辺デバイスを対象に評価を行うことで，その実現性を実証すると共に，パフォーマンス評価により実用性についても検証も行った．

2. 課題

前述をまとめると，解くべき課題は，さまざまなサービス，端末 OS，周辺デバイス，さらには端末と周辺デバイスを繋ぐ通信方式が混在するヘテロジーニアスな環境で，サービス開発と機器制御部（デバイスドライバ）の開発を分離し，利用時に統合することができるようにすることである．このためには，図１に示すような２点を考慮する必要があると考えられる. 以下ではこれらについて説明する．

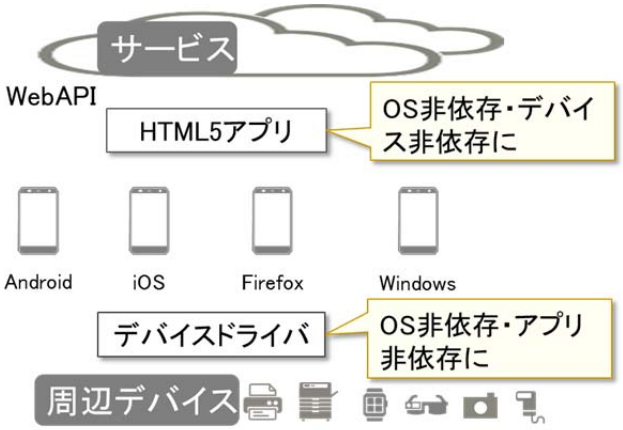

図 1　課題
Figure 1　Issue.

2.1 サービスの OS 非依存・デバイス非依存

サービスを開発する視点では，OS 非依存・デバイス非依存性が課題となる(図 1 の上部)．サービス開発者がデバイスを取り込み利用することになった場合には，サービスが期待する周辺デバイスの機能のみが重要であり，それ以外の要素は考慮することなく周辺デバイスを利用できると使いやすい．例えば，クラウドサービスがある場所に置かれた温度センサーから温度を取得したい場合，温度のリクエストに対して，温度データさえ返ってくれば良く，その際に，温度センサーのメーカや種類，その接続ネットワークや OS の種類といった現実を意識する必要があるようでは困ることになる．

すなわち，サービス開発者の視点では，先進的なサービスを生み出すことに集中することが重要視され，開発したサービス(Web アプリ)が OS 非依存かつデバイス非依存で動作可能であれば，一つアプリを開発するだけで全てを対象にできることになり，開発効率が最も良くなる．

このうちアプリ（サービス）の OS 非依存については，HTML5 アプリによる開発で実現・解決されている．しかしデバイス非依存性まで含めた解決はこれまでなされていない．このため，次々と開発される周辺デバイス情報を把握し,それらを開発に反映する際に困難が伴うことになる．

このアプリ（サービス）のデバイス非依存の課題は，以下 2.2 の周辺デバイスのアプリ非依存と等価の課題であり一緒に考えることができる．

2.2 周辺デバイスの OS 非依存・アプリ非依存

周辺デバイスを開発する視点では，OS 非依存・アプリ非依存性が課題となる(図１下部). 周辺デバイス開発では，現状,一つの周辺デバイスに対してデバイスドライバを OS 毎に開発する必要がある．さらにデバイスドライバ部が，現状，アプリと一体化しているため，デバイスを自由にアプリとつなげることができず，両者の開発が一緒に行われる必要があり開発効率は悪くなる．もし，一つのデバイスドライバの開発により，周辺デバイスを OS やアプリを選ぶことなく動作させることができれば，デバイスメーカーの開発効率を最大化できることになる．

たとえば，温度センサーを開発した際に，そのデバイスの制御を行うデバイスドライバを一つ書くだけで，OS が変わっても対象とする温度情報をだけを取得することができ，デバイスドライバへのアクセスがアプリに開放されていて，アプリと自由につなげることができるようになっていると良い．さらに，温度センサーは何らかのネットワークにより接続されていると考えられることから，デバイスドライバはそのネットワークに対する制御も含めて扱うことができれば良い．

この周辺デバイスのアプリ非依存の課題は，2.1 のアプリのデバイス非依存と等価の課題であることから，2.2 の 2 つの課題を解くことが鍵となる．

3. 解決策

課題解決のため，我々は，以下のような HTML/JavaScript をベースとした Web ドライバアーキテクチャ(Web Driver Architecture: WDA)を提案する．

3.1 構成

提案手法では，OS 上の Web 実行エンジンである Web ラ

Vol.2016-GN-97 No.18
Vol.2016-CDS-15 No.18
Vol.2016-DCC-12 No.18
2016/1/21

ンタイムを前提に，それに対して OS が持つ汎用通信インタフェースと WebApp 層をつなぐインタフェースブリッジ機能を新たに追加する．このインタフェースブリッジは OS 毎に異なる汎用通信インタフェースの差異を吸収する為に用いる．そして，JavaScript/HTML で書かれる Web ドライバを WebApp 層に配置して，WebApp 層からインタフェースブリッジにアクセスすることで周辺デバイスを直接制御可能にする[16]．Web ドライバは，Web アプリと一緒の WebApp 層で動作させる．これにより，デバイスアクセスを OS に依存せずに扱うことが可能になり，Web ドライバを一つ作るだけで，様々な OS での動作を可能にする【2.2 OS 非依存の解決】．また Web ドライバは，動的にインストールすることが可能であり， WebApp 層に挿入し，Web アプリと共に動作させることができる【2.2 アプリ非依存の解決】．これにより，Web アプリと周辺デバイスの組み合わせを柔軟に変えて利用することが可能になる．よって，ユーザが異なる Web サービスに乗り換える際にも容易に切り替えが可能になる．さらに Web ランタイムは，OS の違いと OS バージョン毎の違いの両方を吸収するためにも用いる．これにより，OS に依存しないドライバ開発が可能になり，例えば頻繁な OS バージョンアップへの対応が不要になる．

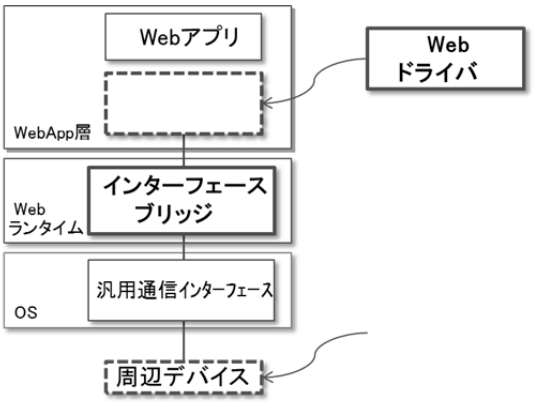

図 2　Web ドライバアーキテクチャ
Figure 2　Web Driver Architecture (WDA).

3.2 動作

提案手法の動作例は次の通りである．ここではスマート端末の上に提案手法が搭載されているものとする．

ⅰ）検出：スマート端末を持ったユーザが周辺デバイスに近づくと，スマート端末がその周辺デバイスを検出する．

ⅱ）デバイス情報取得：周辺デバイスに関する情報を取得するため，周辺デバイスに接続して，どのようなデバイスなのか，どのような機能を提供しているのか特定する．

ⅲ)Web ドライバ配備： 取得したデバイス情報を元に，対応する Web ドライバをダウンロードし，WebApp 層に配置して，Web アプリと紐付けを行う．

ⅳ)デバイス利用： Web アプリから Web ドライバにアクセスすることで，周辺デバイスを利用する．

Web ドライバはクラウドサービス上のストレージに置かれていることを想定しているが，一旦ダウンロードされれば OS の中で保持し，再利用することも可能である．Web アプリも Web ドライバと同様にクラウドサービスから持ってきても良い．

ここで Web アプリと Web ドライバの関係には２つのパターンが考えられる．

- 周辺デバイス発見後，その周辺デバイス利用に必要な Web アプリをダウンロードまたはローカルにある Web アプリを起動すると共に，Web ドライバを入手して周辺デバイスにアクセスする方法．これは，周辺デバイス情報をクラウドサービスに渡して，適切な Web アプリと Web ドライバを入手する場合を含んでいる．
- 予め Web アプリがダウンロードされ立ち上がっており，Web アプリが期待する周辺デバイスが発見された場合に Web ドライバを入手して，あるいは保持された Web ドライバをメモリに展開して周辺デバイスにアクセスする方法．

何れも提案手法の実現性には問題が無く，どちらを用いるかは利用シーンに依存する．しかし，以下では後者を前提として実装・評価等を行っている．

4. 実装

提案手法を図 3 に示したような構成で実装を行った．以下では，端末，Web サーバそれぞれの実装について説明する．

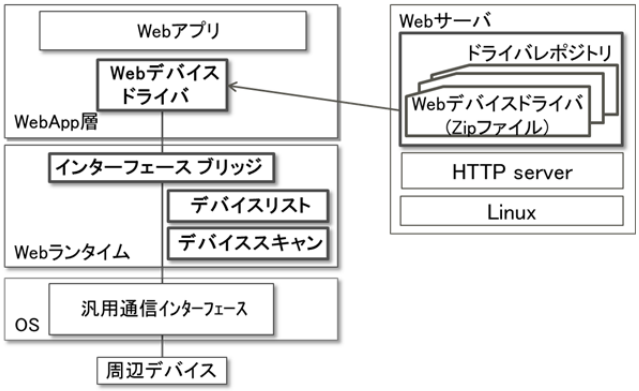

図 3　Web ドライバアーキテクチャの実装
Figure 3　Implementation of Web Driver Architecture.

4.1 端末

以下を Android および iOS を対象に実装を行った．

Web ランタイムには，Apache Cordova[3]を使用した．Cordova は通常 HTML5 を使ったネイティブアプリ開発に用いられ，一つのソースコードから Android や iOS など向けの専用アプリを生成することができるものである．通常

Vol.2016-GN-97 No.18
Vol.2016-CDS-15 No.18
Vol.2016-DCC-12 No.18
2016/1/21

Cordova で生成されたアプリは, Android や iOS のアプリマーケットに登録され，ダウンロードして利用される．ここではインタフェースブリッジ，デバイスリスト，デバイススキャンの機能を Cordova Plugin として実装し，Cordova 本体と Plugin を Web ランタイムとして端末にインストールして利用する．さらに Web ランタイム上に Web アプリと Web ドライバを動的に取り込めるように実装した．

4.1.1 インタフェースブリッジ

インタフェースブリッジは，OS が持つ Bluetooth LowEnergy(BLE)，Classic Bluetooth(BT2.1) [2]，Wi-Fi の 3 つの通信インタフェースにアクセスできるよう実装を行い Web ランタイムに組み込んだ．BLE 上に関しては，使用するプロファイルとして Generic Attribute Profile (GATT) [4]を対象とする．GATT は論理的な通信機能を定義しており，デバイスへの読み書きおよびデバイスからの通知ができる．何れの通信手段も，表 1 に挙げた機器を使って問題なく動作することを確認した．

4.1.2 デバイスリスト

デバイスリストは，Web アプリが利用を期待するデバイス一覧を持つ機能である．デバイスリストは，デバイス ID (did), 発見に使用する比較情報 (cmp), Web ドライバの URL から構成される (extra)．図 4 はデバイスリストの例を表している．

```
[
  {
    "did" : "/hrm",
    "cmp" : { "type": "ble",
              "Complete_List_of_16bit_Service_Class_UUIDs":
              ["0x180d"] },
    "extra" : { "download_url" :
              "http://www.xxx.com:8080/driver/test1.zip" }
  }
]
```

図 4　デバイスリストの例
Figure 4　Example of a Device List.

- did: デバイス種別を一意に識別するための ID(device id)である．この例では“ /hrm”を指定している．この ID は，Web アプリが利用するデバイスを Web ドライバに宣言する際にも利用する．
- cmp: 発見に使用する情報で，無線種別，デバイス特定に使用するフィールドと値を指定する．この例では，type=”ble”を指定し，bluetooth low energy であることを表している．Wi-Fi を指定する場合は”wifi”を記述する．
  Complete_List_of_16bit_Service_Class_UUIDs”は，UUID のタイプを表し，具体的な UUID 例として”0x180d”を指定している．BLE の場合，Advertisement パケットの任意のフィールドを指定可能で，通常は Service UUID（Bluetooth SIG で規定）か Bluetooth Address のどちらかを用いる．
- extra: 追加情報を表し，ドライバのダウンロード URL を定義する．

これらを定義し，スキャンサービスに追加することで，デバイス検出時のフィルタとして利用することができる．

4.1.3 デバイススキャン

デバイススキャンは，デバイスリストを元に，利用できるデバイスが存在しないかどうか，3 つの無線通信インタフェース上のデバイスをスキャンし，デバイス発見を一元管理することが可能な機能である．このユニバーサルなデバイススキャンにより，利用デバイスとして特定されたデバイスに関して, 後述のドライバレポジトリから Web ドライバをダウンロードし，WebApp 層に展開する．
デバイススキャンは，Web アプリから指定された ID に対応するデバイスを探索する．

4.1.4 Web アプリ

Web アプリは，利用者にデバイス・サービスを提供するユーザインタフェースなどを備えるアプリケーションである. 今回の実装ではヘルスケア情報を取得するアプリ, LED ランプを制御するアプリ，カメラを制御・表示するアプリを開発した．これらを動かした状態で，それぞれのアプリが期待するデバイスが発見された場合に，Web ドライバと結合して，利用者に周辺デバイスのサービスを提供する．Web アプリは，Web ドライバの提供する API を呼び出すことでデバイスとのアクセスを行う，

4.1.5 Web ドライバ

Web ドライバは，周辺デバイス一つにつき，一つ開発を行う．Web ドライバは，ドライバ情報ファイル，デバイス制御 API を提供する JavaScript プログラム，アプリ上に表示されるデバイスアイコンから構成される．これらをパッケージ化して，Web サーバに置く．デバイス発見により Web サーバからダウンロードされた Web ドライバは，WebApp 層に展開・配置され Web アプリから利用される．図 5 は，ドライバ情報ファイルの例を表している．

```
{
        "id" : "ble_bp",
        "title" : "BLE_BP",
        "version" : "1.0",
        "icon" : "bloodpressure.png",
}
```

図 5　ドライバ情報ファイルの例
Figure 5　Example of a Driver Information File.

- id: ドライバの文字列記述名であり，BLE 通信の血圧計(Blood Pressure)を表す”ble_bp”を指定している．
- title: ドライバレポジトリで用いるドライバ説明情報であり，例では”BLE_BP”を指定している．
- version: ドライバのバージョン番号であり，例では”1.0”を指定している．
- icon: ドライバのアイコンファイルであり，png ファ

イルのファイル名を指定している．デバイス発見時に，アイコン表示するために用いる．

これらの定義の内，ドライバ記述に必須なのは，”id”と”version”であり，その他はオプション機能である．

4.2 ドライバレポジトリ

ドライバレポジトリは，Web ドライバの保存・ダウンロードの管理を行う機能である．これを Web サーバ上に実装した．Web サーバには，Linux(Ubuntu) 上に HTTP Server(Apache)を載せたものを用いた．ドライバレポジトリに対するドライバダウンロード依頼の内容には，デバイスの ID を入れることができるようになっており，これを確認することで，不用意な利用を防止することが可能である．また，ドライバレポジトリに管理機能を拡張することで，設定条件に基づいたデバイスの発見，ドライバの挿抜・実行をコントロールすることができるようにした．これにより，デバイスリストの書き換えることで，意図しない周辺デバイスの利用をデバイス発見時に制限することが出来る．

5. 評価

本章では，ユーザ・開発者・運用者それぞれの観点によるメリットを示し，実装した提案手法を Android および iOS スマとフォンと市販の機器を利用して動作確認すると共に，従来型の Native アプリケーションとのパフォーマンス比較を行うことで有効性を検証する．

5.1 理論評価

理論評価では，ユーザ観点，開発者観点，運用者観点のそれぞれから評価を行う．本技術は，ユーザの移動に応じて利用機器が変化するような利用シーンで大きな効果が得られる．そこで，ここでは，患者の身辺での検査，病院でのベッドサイド検査，患者自身が自宅などでおこなう自己検査など患者の近いところで行われる診察・検査の総称であるポイントオブケア(Point of care)を想定する．例えば，この自己検査に用いるヘルスケア機器を，看護師が訪問して扱うような在宅医療サービスが今後出てくる可能性がある．

5.1.1 ユーザ観点

ユーザ観点では，これまで周辺デバイスを利用する際に，それぞれ専用のアプリをインストールする必要があった．これを例えばヘルスケアサービスのようなアプリを使って，血圧を測定し，データを蓄積するような場合を考える．機器が壊れてユーザが別の会社の機器に買い換えたような場合であっても，この違いを Web ドライバが吸収し，ヘルスケアサービスを使い続けることができる．また，ユーザがスマートフォンを Android から iPhone に，もしくは iPhone から Android に買い換えた場合にも，その機器へのアクセスモジュールである Web ドライバは同一であるため，問題なく使い続けることが可能である．

5.1.2 開発者観点

開発者観点では，デバイスごとに一つのドライバを開発するだけで，複数の端末で動作させることが可能である．例えば，血圧計用の Web ドライバを一つ開発すれば，様々な OS 上で動作させることができる．今回 Web ランタイムとして用いた Cordova は Windows や Linux 等でも動作可能であり，これらの OS 上で追加開発することなく動作させることができる．

5.1.3 運用者観点

運用者観点では，クラウドサービス業者が，スマート端末経由でクラウドと周辺機器を自在に結びつけるサービスを提供することが可能になる．

例えば，ヘルスケア情報を蓄積・解析する Web サービスを一つ開発すれば，様々な OS を経由して，様々な周辺デバイスを Web サービスから直接掴んで動作させることができるようになる．

5.2 動作評価

実装した提案手法が，周辺デバイスのハードウエアへの変更等を必要とせず，Web ドライバのみ開発すれば利用できることを確認するため，市販の周辺デバイスを使って評価を行う．図 6 は評価環境の構成を示している．ここで，ドライバレポジトリ(Web サーバ)とスマート端末(Android, iOS スマートフォン)は，Wi-Fi によって接続されている．また，スマート端末と周辺デバイスは，Wi-Fi, BT2.1(Classic Bluetooth), BLE(Bluetooth Smart)の何れかを使って接続されるものとする．

図 6　評価の構成

Figure 6　Structure of evaluation.

周辺デバイスは，表 1 に示したように，Continua 対応機器(血圧計，体重計，体温計)，LED ランプ，カメラを使用した．周辺デバイスからの情報取得動作を確認するため，Continua 機器(BT2.1, BLE 接続)を用いた．また，周辺デバイスへの制御動作を確認するため，LED ランプ(BLE 接続)を用いた．また Wi-Fi による情報取得・制御動作を確認するためカメラを用いた．

Vol.2016-GN-97 No.18
Vol.2016-CDS-15 No.18
Vol.2016-DCC-12 No.18
2016/1/21

表 1　Web ドライバ開発し動作確認したデバイス
Table 1　Devices which develop the Web device and confirm the operation.

デバイス種類	メーカ・型格	通信手段
血圧計	A&D UA-651BLE	BLE
血圧計	Omron HEM-7081-IT	Classic Bluetooth (BT2.1)
体重計	A&D UC-352BLE	BLE
体温計	A&D UT-201BLE	BLE
LED ランプ	Nexturn	BLE
カメラ	SONY DSC-QX30	Wi-Fi

5.2.1 Continua 機器

Continua Health Alliance では, 健康機器の通信規格の標準を定めている．これに準拠した 3 種類 4 つの Continua 機器を用意し，それぞれに対応する Web ドライバを開発した．そして, Android 5.0.2(Arrows F-04G), iPhone 6 Plus 上にヘルスケア機器情報を取得する Web アプリを動作させた状態で，以下の機器を接続し，その機器に対応する Web ドライバを取得して動作することを確認した．また，OS が変わっても同一の Web ドライバで機器にアクセスできることも確認した．

血圧計：血圧計は，ユーザが腕帯を装着し血圧計のボタンを押すと血圧計測が始まり，最高血圧，最低血圧，脈拍の３つの情報を得ることができる．測定が終了すると，血圧計は BLE または BT2.1 を使ってその結果数値の報告を行おうとする．それをアドバタイズ情報として端末が感知した際に，血圧計の Web ドライバを読み込んで，結果を取得すると共に，Web アプリでその結果を表示できることを確認した．BT2.1 に関しては，HDP(Healthcare Device Profile) のプラグインを Web ランタイム側に実装し, Web ドライバはこれを前提に開発したものを用いた．
図 7 は，計測におけるスマートフォン画面をキャプチャしたものである．

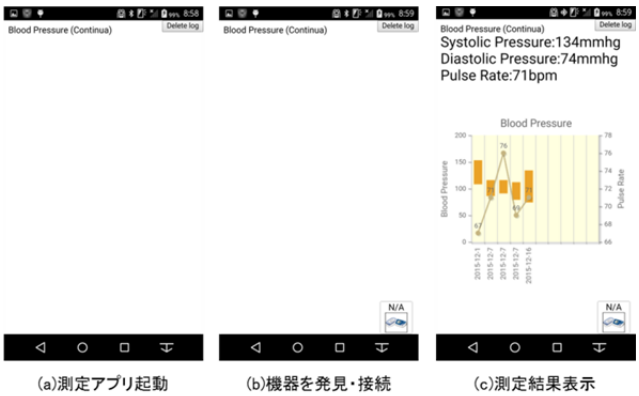

図 7　血圧計による動作確認
Figure 7　Operation Check using Broad Pressure Monitor.

(a)は HTML で書かれたヘルスケア情報を測定するためのアプリケーションである．(b)は血圧の測定が終わり，血圧計からスマートフォンに通信要求が行われ，血圧計を制御するための Web ドライバが組み込まれた状態を表している．右下に表示されているのが，血圧計 Web ドライバのアイコンであり，これによって通信が確立されたことが分かる．(c)は測定結果の表示を表している．最高血圧 134，最低血圧 74，脈拍 71 が表示され，過去の測定結果と共にグラフ表示もされる．

体重計/体温計：体重計と体温計はほぼ同じ動きをする．体重計は，ユーザが体重計に乗り測定し終えた後に，BLE 経由で結果報告のアドバタイズ信号が発行される．体温計は，ユーザがスイッチを押して体温を測定し終えた後に，BLE 経由で結果報告のアドバタイズ信号が発行される．これを端末内のデバイススキャンが検知して，体重計または体温計の Web ドライバを読み込み, 体重計情報または体温計情報を取得して，Web アプリに渡し，Web アプリでその結果を表示することを確認した．
体温計では，体温計内のファームウエアによって，デバイスアクセス時の応答時間に若干違いがあった．しかし，開発した Web ドライバでは, そのような差異に左右されることなく動作することを確認できた．

5.2.2 LED ランプ

評価に用いた LED ランプは, BLE で制御が可能である．そして, この LED ランプの On/Off や RGB 色調を変えるコマンドを発行する Web ドライバを開発した. LED ランプの電源を入れることで，Web ドライバをスマートフォン上にダウンロードし, これに対して Web アプリから制御コマンドを発行することで，LED ランプの制御ができることを確認した．

5.2.3 カメラ

Wi-Fi 接続の周辺デバイスを用いた場合の検証を行うため，Wi-Fi 機能付きのカメラを用いて動作検証を行った．スマホのデバイススキャンが Wi-Fi の SSID からカメラを見つけると，Web ドライバをダウンロード，カメラアプリと連動して，ユーザが自分のスマホを使ってリモートで外部カメラをコントロールしながらスマホとは別のアングルから写真を撮ることができることを確認した．Wi-Fi で，Web ドライバで行うべきことは Proxy としての機能であるため，BLE/BT2.1 と比較すると複雑な記述を行うことなく Web ドライバを記述可能である．

5.2.4 パフォーマンス評価

Web ドライバの動作パフォーマンスを確認するため，周辺デバイスとして Broadcom 社 WICED Sense を使用し, Web ドライバ/Web アプリを開発した．また，このデバイスを利用するネイティブアプリ(Android の Java アプリ)を開発し，Web ドライバとの比較対象とした．スマートフォンには Nexus 5(Android 5.1.1)を用いた. 図 7 は評価の構成である．スマートフォンには, Web アプリまたは Java アプリを置き，

Vol.2016-GN-97 No.18
Vol.2016-CDS-15 No.18
Vol.2016-DCC-12 No.18
2016/1/21

3.2 で示した 4 つの動作に対応する，8 つの基本的な処理について評価を行った．

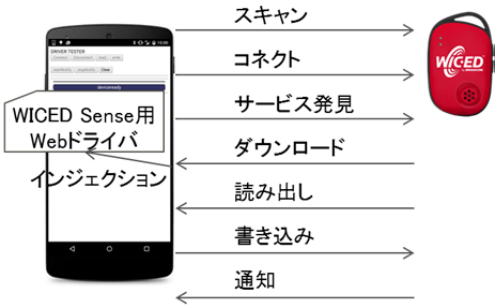

図 8　評価の構成

Figure 8　Structure of evaluation.

表 2　処理時間の比較(ネイティブアプリ vs Web ドライバ /Web アプリ)

Table 2　Comparison of transaction time between native application and web device driver / web app.

動作	処理	時間[msec]	
		Web ランタイム/Web ドライバ	ネイティブアプリ(Java)
i)検出	スキャン	126	15
ii)デバイス情報取得	コネクト	422	427
	サービス発見	3246	3231
iii)Web ドライバ配備	ダウンロード	788	-
	インジェクション	404	-
iv)デバイス利用	読み出し	433	400
	書き込み	440	393
	通知	24	<1

表 2 は周辺デバイスへのアクセスに要する時間の測定結果である．Web ランタイム/Web ドライバの方が時間を要している項目があるが，Web ドライバには Web 層と繋ぐ役割/コードまで含まれている点に留意されたい．仮にクラウドサービスから周辺デバイスを制御することが目的であるとすると，このネイティブアプリはローカルでの処理だけを行っているため，この目的を満たしていないことになる．すなわち，表 2 はあくまで従来のローカルアプリと提案手法を意図し比較したものとなっている．

また，表 3 は，各処理における Java 層と JavaScript 層の行き来の回数をカウントしたものである．

表 3　呼び出し回数

Table 3　Number of times of calling.

処理	回数	
	Cordova Exec	Callback from native
検出	0	0
コネクト/サービス発見	4	4
読み出し	4	4
書き込み	4	4
通知	0	1

Cordova Exec は Java Script 層から Java 層へのアクセスを表しており，1 回の呼び出しでおよそ 11mS 程度かかる．また，Callback from native は，Java 層から Java Script 層への応答を表しており，1 回の呼び出しでおよそ 6mS 程度かかる．これらの結果からそれぞれの処理について検証する．

スキャンは，BLE デバイスを検出する機能である．この結果では，Web ドライバ/Web アプリがネイティブアプリよりも約 110mS 余計に時間を要している．これは BLE をスキャンして検出した後の callback(Web 層への送信)およびスキャン結果を JSON 形式で生成するところまで Web ランタイムが行っているために追加時間を要していることによる．

コネクトは，GATT への接続処理を行っている．これについては，ほぼ同じ処理時間となっている．

サービス発見は，周辺デバイスが持つサービスを特定する機能である．この処理については，両者ともに 3 秒以上を要している．理由は，評価に利用した周辺デバイス WICED Sense が，加速度，角速度，電子コンパス，気圧，温度湿度などの多数の機能を備えているため，それぞれに対する確認時間が積み重なったことによる．このサービス発見の時間は，例えば，機能が単純な LED ランプを使って測定すると，46mS と短くなる．

ダウンロードおよびインジェクションは，Web ドライバのダウンロード時間と Web ドライバの Web 層への配備時間を表している．合わせて 2 秒程度の時間がかかっているが，ユーザはこれにより，準備を手間なくできるようになるメリットを得ることができるようになる．なお，ネイティブアプリでは，通常デバイスドライバに相当する部分は最初からアプリに組み込まれているため，ダウンロードやインジェクションの測定は行っていない．

読み出し/書き込みは，周辺デバイスからの情報の読み出し/情報の書き込みを行う機能である．これらの処理については，両者で差分が無い．表 3 を見ると，読み出し/書き込み共に 8 回の呼び出しが行われているため遅くなっている．

通知は，周辺デバイスからの単発での情報送信を行う機能である．この処理については，Web ドライバが約 23mS 余計に時間を要している．この原因は，ネイティブ層から Web 層にアクセスする呼び出しが 1 回行われていることと，ドライバからアプリへの送信に残りの時間が割かれている為である．

6. 関連研究・技術

以下では本提案に関連する研究および技術との比較を行う．

W3C Web Bluetooth[9] により，Web 技術から直接 Bluetooth デバイスにアクセスする仕組みが検討され，Firefox, Chrome ブラウザで試験的な実装が進められている．これにより Bluetooth 用のインタフェースブリッジに関し

ては，標準的なものが準備される可能性がある．本提案方式との差分は，本提案で，アプリケーションとデバイスアクセス部分を分けたこと，Web ドライバを動的インストールすることが可能な点である．標準機能が用意されると，今回利用した Cordova Web ランタイムが必要無くなるかもしれない．よって，本提案手法がより広く利用される可能性があると考えている．

Java Script はスクリプト言語であるため，そのパフォーマンスが課題として指摘される[5].これを高速化するための取り組みも進められている [6]．さらに WebAssembly では，C や C++などで書かれたバイナリーフォーマットを，JavaScript に組み込むことでパフォーマンスアップする取り組みが進められている[17]．今後，周辺デバイスを扱うため Cyber-Physical 間のパフォーマンス最適化を行う改善手法が研究課題に挙がる可能性も考えられる

GoT API[7]は，スマートフォン上のブラウザやアプリから周辺デバイスにアクセスするための仕様で公開されている．この手法では，周辺デバイスにアクセスするための機能をアプリとして実現するため，周辺デバイスごとに一つのアプリが必要となる．一方で，本提案では，アプリのアプリストアからのインストールをする手間を省略することが可能となる．

OSGi ベースで IoT にアクセスする Plug and play に関する研究や[13], IoT での Plug and Play の消費電力について研究が行われているが[14]，JavaScript までは扱っていない．SPOT では，スマートフォンから，XML を使って，様々な Wi-Fi 機器の制御を行う手法を提案し，家庭向けの機器制御を行っている．本提案方式との違いは，JavaScript でなく XML を使って実現している点，SPOT では Bluetooth は対象としていないが本提案では対象にしていること，またドライバの動的インストールが無い点が差分である．

W3C WoT[10]では，IoT[1]を Web からの利用について議論を行っている．基本形として HTTP[11][12]や CoAP[18]などのプロトコルで機器が通信できることを前提としているが，現状の Bluetooth 機器をレガシーデバイスと呼び，Adapter と呼ばれる層でこれらを制御する構成をとろうとしている．本提案方式は，Bluetooth に関しては，この Adapter に相当し，Wi-Fi 機器は HTTP によるアクセスに相当するものと考えられる．

7. おわりに

従来，スマート端末で，周辺デバイスを利用するためには，OS や周辺デバイスごとに専用アプリが必要で，利用者はアプリのインストール，開発者は OS や周辺デバイスごとのアプリ開発が必要であった．そこで，本稿では，Web アプリから直接周辺デバイスの制御を可能とする Web ドライバアーキテクチャの提案を行うと共に，提案手法を実装，実デバイスを使って動作を検証した．これにより，デバイスメーカーは OS に依存しないデバイス制御が可能になるとともに，サービス利用者は，周辺にあるデバイスを即座につないで活用可能になりクラウドからのデバイス利用がより簡単にできるようになることを示した．

参考文献

1) Gubbia, J.: Internet of Things (IoT): A vision, architectural elements, and future directions, Future Generation Computer Systems, Volume 29, Issue 7 (2013)
2) Bluetooth special interest group, Bluetooth Core Specification 4.2 (online), available from 〈https://www.bluetooth.org/en-us/specification/adopted-specifications〉 (accessed 2015-12-13).
3) Apache Cordova (online), available from <https://cordova.apache.org/> (accessed 2015-12-13)
4) GATT specifications Services, Bluetooth special interest group (online), available from <https://developer.bluetooth.org/gatt/services/Pages/ServicesHome.aspx> (accessed 2015-12-13)
5) Charland, A. and Leroux, B.: Mobile application development: web vs. native, Communications of the ACM 54.5 , pp. 49-53 (2011)
6) Zhu, Y. and Reddi, V.: High-performance and energy-efficient mobile web browsing on big/little systems, 19th IEEE International Symposium on High Performance Computer Architecture, pp. 13-24 (2013)
7) GotAPI (online), available from <http://en.device-webapi.org/gotapi.html> (accessed 2015-11-19)
8) Kliem, A., Koner, M., Weissenborn, S. and Byfield, M.: The Device Driver Engine-Cloud enabled ubiquitous device integration, IEEE Tenth International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), pp. 1-7 (2015)
9) WebBluetooth, W3C Web Bluetooth Community Group (online), available from https://www.w3.org/community/web-bluetooth/ (accessed 2015-12-13)
10) Web of Things, World Wide Web Consortium (online), available from http://www.w3.org/WoT/ (accessed 2015-12-13)
11) Dominique, G.: A web of things application architecture-Integrating the real-world into the web, PhD Thesis, ETH Zurich (2011)
12) Guinard, D., Trifa, V. and Wilde, E., A Resource Oriented, Architecture for the Web of Things, Internet of Things, (2010)
13) Vlad, S.: Towards a restful plug and play experience in the web of things, IEEE international conference on Semantic computing, pp. 512-517, (2008)
14) Yang, F., Matthys, N., Bachiller, R., et al.: μ PnP: plug and play peripherals for the internet of things, The tenth European Conference on Computer Systems, ACM, p.25 (2015)
15) Moazzami, M., Mashima, D. and Chen, W.: SPOT: Smartphone-Based Platform to Tackle Heterogeneity in Smart-Home Systems, The 21st annual international conference on Mobile Computing and Network (2015)
16) Ito, H. and Nimura, K.: Customizable Web-Based System to Federate Smart Devices and Peripherals, Computer Software and Applications Conference Workshops, 38th IEEE COMPSAC, pp. 584-589 (2014)
17) WebAssembly, WEBASSEMBLY COMMUNITY GROUP (online), available from https://www.w3.org/community/webassembly/ (accessed 2015-12-13)
18) The Constrained Application Protocol (CoAP), Request for Comments:7252 (online), available from http://tools.ietf.org/html/rfc7252 (accessed 2015-12-13)