情報処理学会論文誌　コンシューマ・デバイス & システム　Vol.7　No.1　10–20 (Jan. 2017)

コンシューマ・システム論文

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[1,2,a)]　伊藤 栄信[1]　坂本 拓也[1]　中村 洋介[1]　西垣 正勝[2]

受付日 2016年6月30日, 採録日 2016年10月31日

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 型デバイス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[1,2,a)]　Hidenobu Ito[1]　Takuya Sakamoto[1]　Yousuke Nakamura[1]
Masakatsu Nishigaki[2]

Received: June 30, 2016, Accepted: October 31, 2016

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，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増えつつあり，これらをスマート端末から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アプリケーションが必要であるとともに，利用者はそのインストール作業が必要となり，必ずしも利便性が良いとはいえない状況にある．また，スマートフォンのようにユーザがつねに携帯して移動し，様々な周辺デバイスに遭遇する状況では，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く，せっかくの利用タイミングを逃すことになりかねない．このような新たな利

[1] 株式会社富士通研究所
Fujitsu Laboratories Ltd., Kawasaki, Kanagawa 211–8588, Japan
[2] 静岡大学創造科学技術大学院
Graduate School of Science and Technology, Shizuoka University, Hamamatsu, Shizuoka 432–8011, Japan
[a)] kazuaki.nimura@jp.fujitsu.com

用シーンにおいて，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新しいサービスの創生をすることにつながるものと考えられる．

周辺デバイスの活用は，クラウドサービスとスマート端末上のアプリケーションを連携させて行うことが考えられる．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・記述される場合が多い．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．このため，周辺デバイスも，HTML/JavaScript により利用できるようになると，同じ言語でアプリケーション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．

ここで，クラウドサービスと連携するアプリケーション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ことが一般的であるため，アプリケーションと周辺デバイスを連携させるにあたっては，お互いの依存関係の制約が疎になることが望ましい．すなわち，アプリケーションが必要となったデバイスをいつでも必要に応じて追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仕組みを確立しておくことが，IoT 時代のクラウド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 型デバイスドライバと呼ぶ）を必要なタイミングでスマート端末上の Web ランタイム上で動的に挿抜でき，Web アプリと動的に紐付けて実行することが可能な Web 型デバイスドライバアーキテクチャを提案する．これは，クラウドサービス，端末 OS，周辺デバイスの間の依存関係の制約を疎にするとともに，必要に応じて動的に紐付けて利用できる仕組みを提供するものである．これにより，以下の三者に対してメリットがもたらされる．

- 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，スマート端末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，1 つのドライバを書くだけであらゆるスマート端末でデバイスを動作させることが可能になる．
- 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るサービスを提供することが可能になる．

以下では，Web 型デバイス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ともに，実装方法について示す．また，市販の周辺デバイスを対象に評価を行うことで，実用性について検証を行った結果を示す．

2. 課題

本稿が対象とする課題は，様々なクラウドサービス，端末 OS，周辺デバイス，さらにはそれらをつなぐ通信方式が混在するヘテロジーニアスな環境で，Web アプリの形態で提供されるサービスの開発と周辺機器を制御するデバイスドライバの開発を分離し，利用時に必要に応じて組み合わせ利用できるようにすることである．**図 1** はサービスからの周辺デバイス利用時の構成を示している．ここで，周辺機器は，Bluetooth，Wi-Fi など無線で接続される機器，Web アプリは，HTML，CSS，JavaScript で構成されるアプリ，デバイスドライバは，デバイスにアクセスするためのソフトウエアモジュールを表している．Web アプリで記述されたサービスから周辺デバイスを組み換えて利用するには，サービスと周辺機器を操作するデバイスドライバを容易につないだり離したりすることが必要である．このためには，クラウドサービス・端末 OS・周辺デバイスの間に位置する 2 つの依存関係（クラウドサービスの OS 非依存・デバイス非依存，周辺デバイスの OS 非依存・アプリ非依存）を考慮する必要がある．以下ではこれらについて説明する．

2.1 サービスの OS 非依存・周辺デバイス非依存

サービス開発者がクラウドサービスを利用する端末アプリを開発する場合，端末 OS ごとに異なるアプリを開発するのには手間がかかる．また，サービス開発者が，クラウドサービスにデバイス操作を取り込むことが必要になった場合，クラウド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扱いやすい．たとえば，クラウドサービスが，ある場所に置かれた温度センサから温度を取得したい場合，温度取得のリクエストに対して，温度データさえ返ってくればよい．その際に，温度センサのメーカやセンサの種類，その接続ネットワークや OS の種類といった設定を行う必要があるようでは，その活用までに大

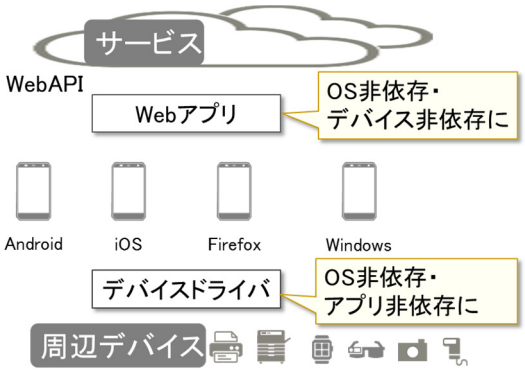
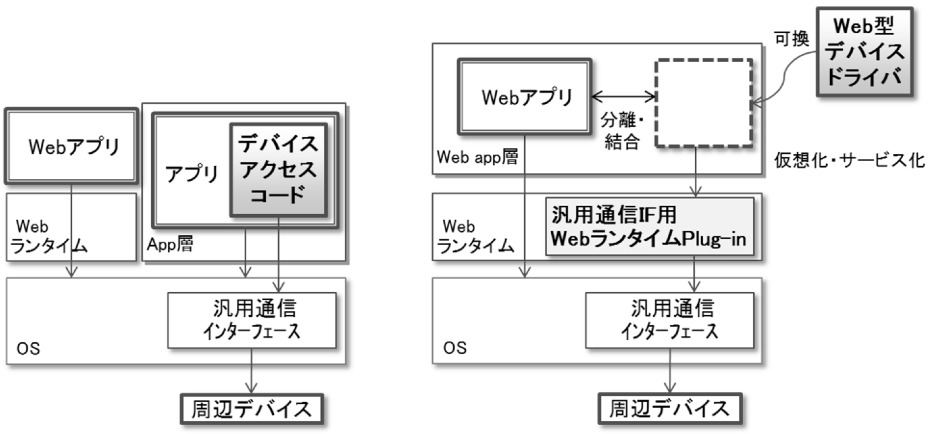


図 1　サービスからの周辺デバイス利用時の課題
Fig. 1　Issue on use of the peripheral device from services.



(a) スマート端末（現在）　　(b) Web型デバイスドライバアーキテクチャ（提案）

図 2　Web 型デバイスドライバアーキテクチャ
Fig. 2　Web type Device Driver Architecture (WDDA).

変な手間が予想される．

すなわち，サービス開発者の視点では，先進的なクラウドサービスを生み出すことに集中することが重要視される．開発したサービスが OS 非依存かつデバイス非依存で動作可能であれば，1 つのアプリを開発するだけですべてを対象にできることになり，OS やドライバごとにアプリを個別に作り込む（互換性に配慮した形でのコードの修正と動作確認）ための煩雑かつ非本質的な工程が不要となるため，開発効率が格段に向上する．

このうちクラウドサービスの OS 非依存については，図 2 (a) 左上部のように，Web アプリによる開発，ランタイム上での実行によるアプリの仮想化により実現・解決されている．しかしクラウドサービスの周辺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．

2.2　周辺デバイスの OS 非依存・サービス非依存

周辺デバイスの開発者は，通常，1 つの周辺デバイスに対してデバイスドライバを OS ごとに開発する必要がある．また，現状のスマート端末では，図 2 (a) 右上部のように，デバイスアクセスコードが，アプリと一体化され両者の開発が一緒に行われることから，デバイスとアプリの関係を自由に後から組み替え直すことができない．もし，1 つのデバイスドライバを開発するだけで，周辺デバイスをあらゆる OS のスマート端末上で，クラウドサービスといつでも自由に組み合わせて利用することができれば，デバイスメーカは開発効率を最大化できることになる．

すなわち，周辺デバイスが OS 非依存かつサービス非依存で動作可能であればよい．しかし，これらの解決はいずれもこれまでなされていない．

2.3　要件

上記において，周辺デバイスの OS 非依存性，クラウドサービスの周辺デバイス非依存性が解決されていないことを示した．これら課題解決に向けた要件は以下のようになると考えられる．

(1) 周辺デバイスへのアクセスを仮想化できること．
これは，周辺デバイスの OS 非依存性を解決するため，クラウドサービスの OS 非依存化においてアプリの仮想化を行ったのと同様に，周辺デバイスにおいても仮想化を実現できるようにすることを表している．

(2) クラウドサービスとデバイスドライバを分離・可換・結合できること．
これは，クラウドサービスの周辺デバイス非依存性を解決するため，サービスとデバイスドライバを分離できるようにするとともに，必要に応じて入替え・結合できるようにすることを表している．

これらを満たすことによって，クラウドサービス開発者および周辺デバイス開発者の開発効率を同時に最大化するアーキテクチャを確立できることになる．また，これらが実現されることで，ユーザの利便性も向上することになる．

3.　解決策

前述の要件 (1) は，サービスの OS 非依存化において実績があり，アプリの仮想化に用いられる Web ランタイムを拡張して実現することが最適であると考えられる．Web ランタイムのレイヤをベースとする効果として，周辺デバイスへのアクセスもサービスとして仮想化することが可能になる．また，要件 (2) については，この仮想環境（Web ランタイムレイヤ）上でサービスとデバイスドライバの両者を扱うことにより，分離・可換・結合を可能にするシンプルな最適解が実現されると考えられる．このために，我々は，HTML/JavaScript でデバイスアクセスの記述を行う Web 型デバイスドライバアーキテクチャ（Web type Device Driver Architecture: WDDA）を提案する．これ

は図 2 (b) に示したように，HTML/JavaScript で書かれた Web 型のデバイスドライバを Web ランタイム上で動作させるとともに，必要なタイミングで動的に挿抜可能にし，Web アプリと動的に紐付けて実行することを可能にする手法である．以下では，この構成と動作を示す．

3.1　構成

提案手法では，デバイスドライバ機能を Web 層で実行するため，OS 上の Web 実行エンジンである Web ランタイムの中に，OS が持つ汎用通信インタフェースと Web app 層をつなぐ汎用通信インタフェース用の Web ランタイムプラグイン機能を新たに追加する．そして，JavaScript/HTML で書かれる Web 型デバイスドライバを Web app 層に配置して，Web app 層から汎用通信インタフェース用 Web ランタイムプラグイン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 [1]．

すなわち，この汎用通信インタフェース用 Web ランタイムプラグインにより，周辺デバイスに対するアクセスのサービス化がなされ，要件 (1) 周辺デバイスへのアクセスの抽象化が解決できることになる．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 型デバイスドライバを 1 つ作るだけで，様々な OS で動作させることが可能になる．また，Web 型デバイスドライバを，動的にインストールし Web app 層に挿入できるようにし，Web アプリとともに動作させることができるようにすることで，要件 (2) サービスとデバイスドライバの分離・可換・結合を解決できることになる．

よって，Web アプリと周辺デバイスの組合せを柔軟に変えて利用することが可能になり，開発効率を最大化可能なエコシステムが確立できることになる．これにより，たとえば，ユーザが異なる Web サービスに乗り換える際の切替えが容易になる．また，Web ランタイムは，OS の違いおよび OS バージョンごとの違いを吸収するために用いることができるので，OS に依存しないドライバ開発に加え，頻繁な OS バージョンアップへの対応が不要になる．

3.2　動作

提案手法の動作フローを説明する．動作は以下の 4 つに大別される．これらは，後述の**図 3** の機能を利用して動作する．

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：スマート端末は，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，それがどのようなデバイスで，どのような機能を提供しているのか特定する．

iii) Web 型デバイスドライバ配備：取得したデバイス情報をもとに，対応する Web 型デバイスドライバをダウンロードし，これを Web app 層に動的に配置して事前にダウンロードしておいた Web アプリと結合する．ここで，Web アプリと Web 型デバイスドライバを利用の時点でダウンロードして動的配備する手法 [14] や，事前にダウンロードした Web アプリと Web 型デバイスドライバを端末内に保持して利用する手法も状況に応じて使い分ける．

iv) デバイス利用：Web アプリから Web 型デバイス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．

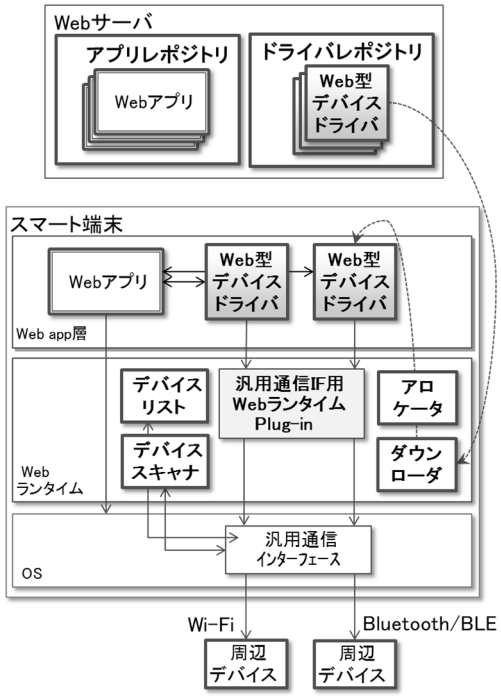


図 3　Web 型デバイスドライバアーキテクチャの実装
Fig. 3　Implementation of Web type device driver architecture.

4.　実装

ここでは，提案手法の実装について示す．提案手法は，図 3 に示すように，スマート端末上の機能と，Web 型デバイスドライバをクラウドで保持するドライバレポジトリにより構成される．以下では，それぞれの詳細について説明する．

4.1　スマート端末

スマート端末上には，図 3 のように，周辺デバイスへのアクセスを仮想化する汎用通信インタフェース用 Web ランタイムプラグイン，これを使ってデバイスにアクセスする Web 型デバイスドライバ，デバイスを発見する際に参照するデバイスリスト，デバイス発見時に機能するデバイススキャナ，Web 型デバイスドライバをダウンロードし Web app 層に配置するダウンローダおよびアロケータ，Web 型デバイスドライバに指示を送る Web アプリが存在する．以下のサブセクションではこれらについて詳述する．

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

図 4　ドライバ情報ファイルの例
Fig. 4　Example of a driver information file.

```
[ {
    "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 5　デバイスリストの例
Fig. 5　Example of a device list.

これらの機能は，Android および iOS（iPhone）端末を対象に実装し，Web ランタイムには，Apache Cordova [2] を使用した．Cordova は HTML5 を使ったネイティブアプリ開発に用いられ，1 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．通常 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．本稿では汎用通信インタフェース用 Web ランタイムプラグイン，デバイスリスト，デバイススキャナ，ダウンローダ，アロケータの機能を 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．

4.1.1　汎用通信インタフェース用 Web ランタイムプラグイン

汎用通信インタフェース用 Web ランタイムプラグインは，OS が持つ Bluetooth Low Energy（BLE），Classic Bluetooth（BT2.1）[3]，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 ランタイムへの組み込みを行った．BLE に関しては，使用するプロファイルとして Generic Attribute Profile（GATT）[4] を対象とした．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を受けることが可能である．

4.1.2　Web 型デバイスドライバ

Web 型デバイスドライバは，以下から構成され，周辺デバイス 1 つにつき，1 セット開発を行う．

- 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

これらはパッケージ化して，Web サーバに置く．

図 4 は，ドライバ情報ファイルの例を表している．

ここで，それぞれの定義は，以下を意味している．

- id：ドライバの文字列による記述名である．例では，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の説明情報である．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を定義する．例では，png ファイルのファイル名を指定している．これは，デバイス発見時にアイコン表示するために用いる．

これらの定義のうち，ドライバ記述に必須な項目は，“id” と “version” であり，その他はオプション機能である．

4.1.3　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 Web ランタイムおよび OS に持たせる機能である．**図 5** はデバイスリストの例を表している．

デバイスリストは，以下から構成される．

- did：デバイス ID．デバイス種別を一意に識別するための 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 型デバイスドライバに宣言する際にも利用する．
  cmp：発見のため使用する比較情報．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，無線種別として type=“ble” を指定し，Bluetooth Low Energy であることを表している．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．
  Wi-Fi を指定する場合は “wifi” を記述する．Complete_List_of_16bit_Service_Class_UUID 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．
- extra：追加情報を表し，Web 型デバイスドライバのダウンロード URL を定義する．

これらを定義し，デバイススキャナに追加することで，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．

4.1.4　デバイススキャナ

デバイススキャナは，Web アプリから指定されたデバイスリストをもとに，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（利用できるデバイスが存在するかどうか）を一元管理することが可能な機能である（3.2 節の動作 ii) における周辺デバイスを特定する作業を担う）．

4.1.5　ダウンローダ

ダウンローダは，デバイススキャナによるデバイス発見を受け，後述のドライバレポジトリから Web 型デバイスドライバをダウンロードして保持する機能である（3.2 節

の動作 iii) における Web アプリと Web 型デバイスドライバのダウンロードを行う作業を担う）．

4.1.6　アロケータ

アロケータは，Web 型デバイスドライバを Web app 層に動的に配置して，Web アプリと結合する機能である（3.2 節の動作 iii) における Web アプリと Web 型デバイスドライバの Web app 層への動的配置を行う作業を担う）．

4.1.7　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ための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．Web アプリを動作させた状態で，アロケータ機能が Web 型デバイスドライバと動的に結合し，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 型デバイスドライバの提供する API を呼び出すことでデバイスへのアクセスを行う．一般的には，図 3 に示したように，Web アプリはアプリレポジトリに置かれており，必要に応じてスマート端末にダンロードして利用する形態となる．

4.2　ドライバレポジトリ

ドライバレポジトリは，Web 型デバイスドライバの保存やダウンロードの管理を行う機能で，Web サーバ上に実装した．Web サーバとして，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 を指定す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，デバイスリストに指定した設定条件に基づき，端末のデバイス発見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，デバイスリストを書き換えることで，デバイス発見時に意図しない周辺デバイスの利用を制限することができる．

5.　評価

本章では，提案手法の理論評価と動作評価を行う．

5.1　理論評価

ここでは，提案手法を導入した際に，ユーザ・開発者・運用者それぞれがどのようなメリットを享受できるのか評価を行う．周辺デバイス利用の一具体例として，血圧計をとり上げて説明する．

5.1.1　ユーザ観点

ユーザ観点では，これまで血圧計などの周辺デバイスを利用する際に，ユーザは周辺デバイス開発業者が提供する専用のアプリをインストールし利用していた．しかし，機器が壊れて別の会社の機器に買い換えたような場合には，新たなアプリを利用することになり，日々測定していたデータなどを引き継ぐことが難しくなる．提案手法では，この違いを Web 型デバイスドライバが吸収し，1 つのアプリをユーザは使い続けることができる．

5.1.2　周辺デバイス開発者観点

周辺デバイス開発者は，これまで血圧計などの周辺デバイスを開発する際に，サポートする OS の数だけアプリを開発する必要があった．提案手法では，周辺デバイスごとに 1 つのドライバを開発するだけで，様々な OS 上で周辺デバイスを動作させることが可能になる．本稿で Web ランタイムとして用いた Cordova は Android や iOS だけでなく，Windows や Linux などでも動作させること可能であり，本手法の適用範囲を広げることで，これらの OS 上で統一的に動作させることができる．

5.1.3　クラウドサービス開発者観点

現時点では，クラウドサービス開発者が周辺デバイスまで考慮して開発や運用を行うことはなされていない．提案手法により，Web サービスを 1 つ開発すれば，クラウドと周辺機器を自在に結びつけて Web サービス側からこれらを直接動作させることができるようになる．たとえば，ユーザが血圧計をスマートフォンに接続した際に，血圧データを管理するための Web アプリケーションと血圧計を駆動するための Web 型デバイスドライバをクラウド側からスマート端末に送るといったヘルスケア Web サービスを容易に開発できるようになる．

5.2　動作評価

ここでは，開発した提案手法を，動作評価した結果を示す．周辺デバイスは，独自で開発したものを利用し動作させることも可能であるが，世の中に多数存在する市販の周辺デバイスをそのまま利用できるかどうかを評価のポイントとした．そこで，様々な周辺デバイスを使い，Web 型デバイスドライバを導入して周辺デバイスが扱えるのか検証を行った．また，Web 層にデバイスドライバを配置することにより，パフォーマンスへのインパクトがあるのかどうかについても評価を行う．

5.2.1　市販周辺デバイスでの評価

ここでは複数の市販の周辺デバイスを使って，Web 型デバイスドライバのみ開発するだけで，周辺デバイスのハードウエアへの変更を必要とすることなく，複数の OS から利用できることを確認する．

図 6 は評価環境の構成を示している．この構成では，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）のいずれかを使って接続される．端末として，Android 5.0.2（Arrows F-04G），iPhone 6 Plus を用い，この上に開発した Web ランタイムを入れて評価を行った．

図 6　評価機器の構成
Fig. 6　Structure of devices evaluation.

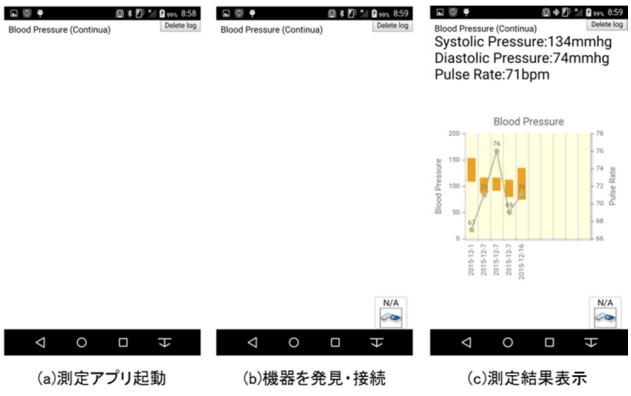


図 7　血圧計による動作確認
Fig. 7　Operation check using broad pressure monitor.

表 1　Web 型デバイスドライバを開発したデバイス
Table 1　Devices which are developed the Web type device driver and confirmed the operation.

通信手段		デバイス種類	メーカ・型格
BLE	センシング	血圧計	A&D UA-651BLE
		体重計	A&D UC-352BLE
		体温計	A&D UT-201BLE
		ロケーションバッジ・バイタルセンシング	Fujitsu, ユビキタスウエア
	アクチュエーション	LED ランプ	Nexturn
Classic Bluetooth (BT2.1)		血圧計	Omron HEM-7081-IT
Wi-Fi		カメラ	SONY DSC-QX30

周辺デバイスとして**表 1** に示した機器を用い，それぞれの動作評価を行った．周辺デバイスは通信手段により 4 つ（BLE センシング機器，BLE アクチュエーション機器，Classic Bluetooth 機器，Wi-Fi 機器）に分類される機器を利用した．

以下では，それぞれの動作結果を示す．

(I) BLE センシング機器

BLE センシング機器として，Continua Health Alliance 準拠の健康機器（血圧計，体重計，体温計）とロケーション・バイタルセンシング機器を利用し，それぞれに対応する Web 型デバイスドライバを開発し評価を行った．そして，それぞれの機器からの情報を取得する Web アプリを動作させた状態で，以下のように動作することを確認した．

- 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 を使って，その結果数値の報告をスマート端末に対して行おうとする．これを，アドバタイズ情報としてスマート端末が感知した際に，血圧計の Web 型デバイスドライバを読み込んで，結果を取得するとともに，Web アプリ上に血圧測定結果を表示できる．

図 7 は，血圧測定を行った際のスマートフォンのキャプチャ画面を示したものである．

(a) は，ヘルスケア情報を測定する HTML で書かれたアプリケーションが起動した状態を表している．この状態では何も情報は表示されていない．

(b) は血圧の測定が終わり，血圧計からスマートフォンに対してアドバタイズ通信要求が行われ，血圧計を制御するための Web 型デバイスドライバがダウンロードされ組み込まれた状態を表している．(b) の右下に小さく表示されているのが，血圧計 Web 型デバイスドライバの存在を表すアイコンであり，これによってドライバレポジトリから Web 型デバイスドライバがダウンロードされたことが確認できる．

(c) は，Web 型デバイスドライバが血圧測定結果のデータを血圧計から取得し，Web アプリにより測定結果を表示した状態を表している．Web アプリには，最高血圧 134，最低血圧 74，脈拍 71 が表示され，過去の測定結果を含めたグラフ表示がなされている．

これらが Android，iOS の両方で動作することを確認した．

- 体重計/体温計：

体重計と体温計も，基本的に血圧計と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．また体温計は，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．これらを端末内のデバイススキャナが検知して，体重計または体温計の Web 型デバイスドライバを読み込み，体重計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．

- ロケーションバッジ・バイタルセンサ：

ロケーションバッジ・バイタルセンサは，ユーザの転倒などのイベントを判断し，その結果を BLE によりスマート端末に通知可能な周辺デバイスである．動作確認では，周辺デバイスから得られるセンシング情報を，Web 型デバ

イスドライバを経由して，Web アプリに表示できること，また，これらが Android，iOS の両方で動作することを確認した．

(II) BLE アクチュエーション機器

BLE アクチュエーション機器として，LED ランプを用いた．この LED ランプは，BLE による制御が可能で，LED ランプの On/Off や RGB 色調を変えることができる周辺デバイスである．動作確認では，LED ランプの電源を入れることで，Web 型デバイスドライバがスマートフォン上にダウンロードされること，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．

(III) Classic Bluetooth 機器

Classic Bluetooth 機器として，血圧計を用い評価を行った．Classic Bluetooth（BT2.1）に関しては，ヘルスケア向けプロファイル HDP（Healthcare Device Profile）が端末内に標準搭載されたものを用い，これとつなげるプラグインを Web ランタイム側に実装した．Web 型デバイスドライバはこれに合わせたものを開発して用いた．動作確認により，(I) で示した血圧計の結果・図 5 と同様に動作することを確認した．

(IV) Wi-Fi 機器

Wi-Fi 機器として，カメラを用いた．このカメラは，スマートフォンなどと Wi-Fi 接続して写真を撮ることなどが可能な周辺デバイスである．動作確認では，カメラの電源を on し，スマート端末上のデバイススキャナ機能が Wi-Fi の SSID からカメラを見つけると，カメラ用の Web 型デバイスドライバをダウンロードし，Web アプリがカメラデバイスと連動すること，そして，ユーザがスマート端末を使ってリモートでこの Wi-Fi カメラをコントロールし，写真撮影などの制御ができることを確認した．Wi-Fi 周辺デバイスにおいては，Web 型デバイスドライバで行うべきことは限られており，Proxy としての機能を記述するだけでよい．このため，BLE/BT2.1 と比較すると，Web 型デバイスドライバの記述はより簡易で済む．

5.2.2　パフォーマンス評価

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ている [5]．本提案手法の目的は，課題にあげた様々な非依存性の解決であるが，デバイスドライバを HTML/JavaScript で記述するため，パフォーマンスに対する影響は懸念材料となる．

そこで，周辺デバイスに対する Web 型デバイスドライバおよび評価用 Web アプリを開発するとともに，同じ機能を備えるネイティブアプリ（Android Java アプリ）を開発し比較を行うことにした．これは図 2 (a) 右上と図 2 (b) の比較にあたる．ここでは，周辺デバイスとして Broadcom 社 WICED Sense（BLE で動作）を使用し，評価のスマートフォンには Nexus 5（Android 5.1.1）を用いた．

図 8　評価の構成
Fig. 8　Structure of evaluation.

表 2　処理時間の比較（ネイティブアプリ vs Web 型デバイスドライバ/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 型デバイスドライバ	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 型デバイスドライバ配備	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

図 8 は評価の構成を表している．スマートフォンには，開発した Web アプリ/Web 型デバイスドライバおよび，Java アプリを置き，提案手法のフローに関連する基本的な処理について評価を行った．ここでアプリは，5.2 節動作評価であげたセンシングデバイスのような，多数のセンシングデータをスマホに送信するものを想定した．準備にあたるスキャン，コネクト，サービス発見，ダウンロード，インジェクションを 1 回，通知を 50 回，読み出し・書き込みを 9 回行うテストアプリにより評価を行い，これらを 20 セット試行した平均値を算出した．**表 2** はこの周辺デバイスへのアクセスに要する時間の測定結果を示している．以下では，それぞれの処理項目に対する考察を行う．

i) 検出

- スキャン：スキャンは，BLE デバイスを検出する機能である．結果より，Web 型デバイスドライバ/Web アプリがネイティブアプリよりも約 110 msec 余計に時間を要していることが分かる．これは Web 型デバイスドライバ/Web アプリでは，BLE をスキャンして検出した後に，callback（図 3 における OS 層から Web app 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ことによる処理時間が影響を与えてい

るものと考えられる．

ii) デバイス情報取得

• コネクト：コネクトは，GATT への接続処理を行う機能である．結果より，処理時間はいずれもほぼ同じであることが分かる．

• サービス発見：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．結果より，両者ともに 3 秒以上を要していることが分かる．この理由は，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．参考までに，このサービス発見の時間は，機能が単純な LED ランプを使って測定した場合には，46 msec と短くなっていた．

iii) Web 型デバイスドライバ配備

• ダウンロード/インジェクション：ダウンロードおよびインジェクションは，ダウンローダとアロケータにより Web 型デバイスドライバのダウンロードと Web 型デバイスドライバの Web 層への配備を行う機能である．結果から，ダウンロードとインジェクション合わせて 1.2 秒程度の時間がかかっていることが分かる．

従来は，周辺機器を利用する際に，アプリのダウンロード手続きが必要で，ユーザの慣れやダウンロードスピードなどの状況に依存するが，1 分ほどの時間がかかっていた．これに対し，提案手法の適用により自動化されたことによって，たかだか 1 秒強の所要時間のみで同等の処理を行うことが可能となった．ユーザは準備に対する手間を省くことができるメリットがあり，ネイティブアプリにはなかった大きなメリットがもたらされる．なお，この機能は，ネイティブアプリでは，あらかじめアプリに組み込まれているか，さらに別のネイティブアプリをインストールしてアプリ間連携させて利用されるため，比較対象となる測定結果は存在しない．

iv) デバイス利用

• 読み出し/書き込み：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/書き込みを行う機能である．結果から，これらの処理については，両者で差分がないことが分かる．これは，Web ランタイム/Web 型デバイスドライバの処理よりも，ワイヤレス通信の I/O 待ち時間が大半を占めているため，方式の差異が表れないものと考えられる．

• 通知：通知は，周辺デバイスからの単方向 1 回の情報送信を行う機能である．結果から，この処理については，Web 型デバイスドライバが約 23 msec 余計に時間を要していることが分かる．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行われていることによるものであると考えられる．

ユーザビリティへの影響が最も大きいと考えられる項目が利用時の処理時間であることから，これらの結果が最もインパクトが大きいと考えられる．総じていえることは，デバイス利用に関しては，通知のような単純な機能では，Web ランタイム/Web 型デバイスドライバの方が時間を要している．一方で，ワイヤレスデバイスにアクセスして結果を待つような読み出し/書き込みのような項目に関しては差がない．すなわち，無線制御の周辺デバイスでは，従来のブラウザ表示目的に比して，ネイティブと Web の差が顕著に表れなくなることが分かる．

Java Script に対するパフォーマンス改善のための関連技術として, Chrome Browser の V8 エンジンでは, Crankshaft JIT（Just In Time）コンパイラによる最適化実行が行われている．また，Firefox Browser では，高速化したい Java Script 処理の前後に，型変換を行わずに処理することを宣言する記述子を宣言することで処理を高速化する Asm.js のような取り組みもある．さらに，WebAssembly [6] 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 に直接組み込むことで，パフォーマンス改善を図る取り組みが進められている．これらの取り組みと提案手法は補完関係にあり，提案手法はこれらの高速化の手法をそのまま利用することができる．現状，これらの取り組みは Web アプリをスマート端末で実行するためのパフォーマンス改善を対象としており，無線アクセスに要する時間など周辺デバイス利用時のすべてのパフォーマンス最適化に寄与するか明らかではないが，今後これらの技術の発展により，パフォーマンス改善されることが考えられる．

5.2.3　提案手法のインパクト

これまで Web アプリからデバイスを制御する場合には，デバイス自体に Web サーバ機能および WebAPI を提供し，Web アプリからデバイスにアクセスさせることが一般的であった．これにはデバイスの改造が必要であり，これがサービス開発者による周辺デバイスの自由な利用を阻害する要因となっていた．今回の提案は，これを解決する機能の提供にあたる．従来は，サービスアプリとデバイスドライバを一体化して開発する必要があり，デバイス開発者とサービス開発者の連携が必須であった．提案手法の適用によって，サービス開発者単独でデバイスを扱うことができるようになる．これは，サービス開発の手間だけでなくコストの削減にも大きく寄与するものと期待される．

システムパフォーマンスの評価で用いたアプリのバイナリサイズは，ネイティブアプリが 1 MB，提案手法では 3.4 MB であった．提案手法では，Web ランタイムとして用いた Cordova などによりバイナリサイズが増える．使用メモリサイズは，ネイティブアプリが 63 KB，提案手法では 180 KB でった．アプリのバイナリサイズ，メモリ使用サイズが提案手法によって 3 倍程度になるが，現在のスマホのストレージ容量，メモリ容量は 64 GB，2 GB 程度の機種があり，提案手法による増分と比べて十分大きいため

影響は軽微であると考えられる．

6. 関連研究・技術

以下では，提案手法に関連する研究および技術をまとめ比較を行う．

W3C Web Bluetooth [7] では，Web アプリ層から直接 Bluetooth デバイスを利用できるようにするブラウザのための仕組みが検討されており，今後無線外部デバイスへのアクセスを HTML/JavaScript で記述するスタイルが広まる可能性がある．また，Web Bluetooth は Chrome ブラウザに対する試験的な実装も進められている．これは図 3 の Web ランタイムと Bluetooth 向けの汎用通信インタフェース用 Web ランタイムプラグインの機能提供と等価であると考えられる．これが様々なブラウザに標準実装されるようになると，Bluetooth 用の汎用通信インタフェース用 Web ランタイムプラグインに関しては，この標準機能に置き換え開発コードを減らすことができるようになると考えられる．一方で，Web Bluetooth の本格的な実装がいまだ始まっていないため正しい判断を下すことは難しいが，Web Bluetooth は従来の Web アプリと同様に，アプリとデバイス制御を 1 つのコードで書く実装スタイルをとるものと予想される．すなわち，図 3 の Web ランタイムの改造によるアプリケーションやドライバの分離，動的にダウンロードし，結合する仕組みは提供されないことになる．

GoT API [8] では，スマートフォン上のアプリやブラウザから周辺デバイスにアクセスするための仕様を定めている．この手法では，周辺デバイスごとに 1 つのネイティブアプリを実装する形態をとるとともに，このネイティブアプリとデバイス利用アプリの間に，Web サーバに相当するもう 1 つのネイティブアプリを挟む構成をとる．また，新たな周辺デバイスを利用するために，開発者はこのデバイスドライバに相当する周辺デバイスごとのネイティブアプリを OS ごとに開発することが必要になる．これらのデバイスドライバアプリ，Web サーバアプリ，デバイス利用アプリは，端末 OS ごとのアプリマーケットに置かれ，ユーザはこれらをダウンロードし，端末上で 3 つを連携動作させ利用する．これらは図 3 の端末側の Web 型デバイスドライバ，アロケータ，Web アプリ，に相当する．提案手法では，ネイティブアプリは Web ランタイム 1 つであり，アプリ，デバイスドライバに相当するコードは Web ランタイム上に動的に配置されるため，これらのインストール手続きの手間を省略することが可能である．

SPOT [9] では，スマートフォンから様々な Wi-Fi 機器の制御を行うために，XML により記述されるドライバを用いる手法を提案している．これは XML による静的記述がベースとなるため，プログラマブルな動作記述までは対象としていない．よって，本提案が対象とする BLE デバイスに対するアクセスや，ドライバの動的インストールまではサポートしていない．SPOT では，図 3 の Web サーバ側の機能は存在せず，端末側は 1 つのアプリに SPOT の機能が搭載される．

W3C WoT (Web of Things) [10] では，IoT [11] に対する統一的な Web API を提供することで IoT プラットフォーム間のインターオペラビリティを実現するための議論を行っている．Web of Things に関しては，HTTP をベースとした研究 [12], [13] などがなされているが，W3C WoT では IoT を統一的に扱う Scripting API と呼ぶ Web API の規定にフォーカスする可能性が高い．そして HTTP や Bluetooth のような通信機能が Protocol Binding され，この API の下で透過的に動作する．これは，図 3 の端末側機能に対して，さらに Web サーバを追加搭載することに相当し，外部からのリクエストを受け付けられるようになる．本提案手法は，ここでの Protocol Binding を動的に行うための仕掛けと位置づけられると考えている．WoT と GoT API の Web サーバとの役割の違いは，GoT API がアプリとドライバの間に Web サーバを配置し，Web サーバはアプリからアクセスを受け付けるものであるのに対し，WoT では，Web サーバとデバイスドライバの間にアプリが置かれ，Web サーバが外部からのアクセスを受け付ける点にある．WoT では，図 3 の Web サーバ側の機能については規定がなされず，任意の運用にな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 型デバイスドライバアーキテクチャの提案を行うとともに，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，デバイスメーカはデバイスドライバを 1 つ書くだけで様々な OS に接続されたデバイスの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スマート端末か即座につないで活用可能になることを示した．

参考文献

[1] 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).
[2] Apache Cordova (online), available from ⟨https://cordova.apache.org/⟩ (accessed 2015-12-13).
[3] 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).
[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, *Comm. ACM*, Vol.54, No.5, pp.49–53 (2011).
[6] WebAssembly, WEBASSEMBLY COMMUNITY GROUP (online), available from ⟨https://www.w3.org/community/webassembly/⟩ (accessed 2015-12-13).
[7] WebBluetooth, W3C Web Bluetooth Community Group (online), available from ⟨https://www.w3.org/community/web-bluetooth/⟩ (accessed 2015-12-13).
[8] GotAPI (online), available from ⟨http://en.device-webapi.org/gotapi.html⟩ (accessed 2015-11-19).
[9] Moazzami, M.W.: SPOT: Smartphone-Based Platform to Tackle Heterogeneity in Smart-Home Systems, *The 21st annual international conference on Mobile Computing and Network* (2015), available from ⟨https://github.com/SPOT-SMARTHOME/SPOT⟩ (accessed 2016-04-30).
[10] Web of Things, World Wide Web Consortium (online), available from ⟨http://www.w3.org/WoT/⟩ (accessed 2015-12-13).
[11] Gubbia, J.: Internet of Things (IoT): A vision, architectural elements, and future directions, *Future Generation Computer Systems*, Vol.29, Issue 7 (2013).
[12] Dominique, G.: A web of things application architecture-Integrating the real-world into the web, Ph.D. Thesis, ETH Zurich (2011).
[13] Guinard, D., Trifa, V. and Wilde, E.: A Resource Oriented, Architecture for the Web of Things, *Internet of Things* (2010).
[14] 伊藤栄信，二村和明：スマート端末向けセキュアアプリ管理・実行基盤技術，雑誌 FUJITSU，Vol.64, No.5, pp.510–515 (2013)，入手先 ⟨http://www.fujitsu.com/downloads/JP/archive/imgjp/jmag/vol64-5/paper08.pdf⟩（参照 2016-05-14）.

二村 和明　（学生会員）

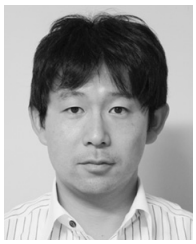
1969 年生．1994 年東京電機大学大学院情報通信工学専攻修士課程修了，同年富士通株式会社入社．1997 年から富士通研究所勤務．2016 年より静岡大学創造科学技術大学院博士課程．IoT サービス・プラットフォームの研究開発に従事．

伊藤 栄信

1968 年生．1993 年大阪府立大学大学院工学研究科数理工学専攻博士前期課程修了．同年（株）富士通研究所入社．IoT サービス・プラットフォームの研究開発に従事．

坂本 拓也

1973 年生．1997 年神戸大学大学院自然科学研究科情報知能工学専攻博士前期課程修了，同年富士通株式会社入社．2000 年から富士通研究所勤務．IoT サービス・プラットフォームの研究開発に従事．



中村 洋介

1977 年生．2000 年横浜国立大学工学部電子情報工学科卒業．平成 14 年同大学大学院工学研究科電子情報工学専攻修士課程修了．同年（株）富士通研究所入社．IoT サービス・プラットフォームの研究開発に従事．

西垣 正勝　（正会員）

1990 年静岡大学工学部光電機械工学科卒業．1995 年同大学大学院博士課程修了．日本学術振興会特別研究員（PD）を経て，1996 年静岡大学情報助手．同講師，助教授の後，2010 年より同創造科学技術大学院教授．博士（工学）．情報セキュリティ全般，特にヒューマニクスセキュリティ，メディアセキュリティ，ネットワークセキュリティ等に関する研究に従事．2013～2014 年情報処理学会コンピュータセキュリティ研究会主査．2015 年より電子情報通信学会バイオメトリクス研究専門委員会委員長．