Vol.2017-MBL-82 No.37
Vol.2017-UBI-53 No.37
2017/3/10

機能指向マッシュアップによるUGC型サービスの提供を実現するIoEフレームワークの提案

松野宏昭[†1] 丸島晃明[†1] 有村汐里[†2] 可児潤也[†3] 二村和明[†3]
峰野博史[†4] 飯田一朗[†5] 西垣正勝[†4]

概要：近年，センサやアクチュエータなどの様々なデバイスを連携させる，IoE（Internet of Everything）サービスの開発が進みつつある．より快適かつ便利なIoEサービスの実現には，高度なコンテキスト推定や複雑な制御が必要であるが，現在，インターネットに接続されるデバイスは台数・種類ともに増加し続けており，これらの組み合わせ数は膨大なものとなる．したがって，全てのユーザを取り巻くコンテキストに応じたIoEサービスを，個人や企業が単独で開発していくことは生産性の点で課題がある．本稿ではこれを解決するため，現在のスマートフォンアプリケーション市場のように，UGC（User Generated Contents）型のエコシステムを形成し，ユーザによるサービスの作成を促進するフレームワークを提案する．提案フレームワークでは，(i) デバイスが行うタスクを「機能」単位でAPI化しており，機能のマッシュアップによってユーザ（サービス開発者）は自由にサービスを作成することができる．また，(ii) 物理的なデバイスをカプセル化し，IoEサービスとデバイスの間に.セキュリティのチェック層を導入することで，サービス開発者にセキュリティ観点での判断を強いずに開発を可能にする．さらに，(iii) 仮想空間上のカプセルと物理空間内のデバイスを動的に接続する仕組みを提供しており，サービス開発者は機能のマッシュアップのみを考えて具体的なデバイスに依存しないサービスを開発することができる．

1. 背景

近年，インターネットに接続されたセンサやアクチュエータなどの様々なデバイスだけでなく，ヒューマンリソースをも連携させたIoE（Internet of Everything）サービスの開発が進みつつある．現在，インターネットにつながるデバイスは増加し続けており2020年には304億個まで増大すると予測されている[1]. これに伴い，一般的な住環境でも，家電やスマートトイといったデバイスがインターネットに接続されるようになり，特別な施設でなくともIoEサービスを実現することが可能になった．また，デバイスとクラウドを接続し，デバイスからのデータを収集，蓄積，分析や，リモート操作を可能にする，クラウドサービス[2]も一般向けに公開されている．

1.1 目的

一般的な住環境でのIoEサービスの利用目的には，コンテキストに応じて様々なデバイスの制御を自動化・高度化し，ユーザの利便性，快適性を向上するといった用途が挙げられる．また，オフィスや工場では，自動化・高度化による，業務の効率化といった用途が挙げられる．

今後，IoEサービスを生活の向上や産業の発展へ繋げるためには，高度なIoEサービスを普及させ，様々な環境で活用していくことが望ましい．しかし，このような世の中を実現するためには，IoEサービスの開発に関する3つの大きな課題を解決する必要があると我々は考える．

1.2 課題

(a) 膨大な数のデバイス，コンテキスト
ユーザや環境によって，利用可能なデバイスやクラウドサービスは多種多様であり，連携の組み合わせ数は膨大なものとなる．このため，全てのユーザを取り巻くコンテキストを想定し，それぞれに応じたデバイスやクラウドサービスを組み合わせる連携ルールを書き下すことは，個人や企業が単独で行うには難しい．

(b) ニーズとシーズのマッチング
同じIoEサービスに対して，ユーザによっては「気が利く」と感じたり，「大きなお世話」と感じたりするように，ユーザごとに必要とするIoEサービスが異なる場合がある．さらに，コンテキストの変化に応じて，ユーザのニーズも都度変化する可能性がある．このように，ユーザのニーズには多様性と変動性が存在するため，高度なIoEサービスの提供のためには，ユーザのニーズとIoEサービスのマッチングの判断を行う必要があるが，これを高精度に行うことは難しい．

(c) セキュリティ
物理的なデバイスを制御することが可能となるため，IoEサービスには，物理的な損壊や人命にかかわるリスクが存在する．危険な動作を防止するためにも，アクセス制御には，コンテキストに応じて権限を変更するような仕組みが必要である．また，様々なユーザがIoEサービスを利用するため，1つのデバイスを複数のIoEサービスが参照し，競合が発生する場合がある．したがって，これを検出・調停する仕組みが必要である．さらに，デバイスの構成や設定，ログから，病気などのセンシティブな情報が流出する可能性がある．このため，プライバシ対策も必要である．

†1 静岡大学大学院 総合科学技術研究科
Graduate School of Integrated Science and Technology, Shizuoka University
†2 静岡大学大学院情報学研究科
Graduate School of Informatics, Shizuoka University
†3 株式会社富士通研究所
Fujitsu Laboratories Ltd.
†4 静岡大学創造科学技術大学院
Graduate School of Science and Technology, Shizuoka University
†5 秋田県立大学
Akita Prefectural University

これらの課題を解決するため，本研究では，現在のスマートフォンアプリ市場のように，UGC（User Generated Contents）型のサービスアプリ開発エコシステムを形成し，ユーザによる IoE サービスの作成を促進するフレームワークを提案する．

課題(a) ,(b) ,(c) に対する，提案方式のアプローチを述べる．まず，課題(a) は，UGC 型エコシステムを形成し，数多くのユーザ（サービス開発者）によって，様々な目的に応じた IoE サービスアプリが開発されるようにすることで解決する．次に，課題(b) は，UGC エコシステムによって開発される無数の IoE サービスアプリを，アプリ市場を通じて流通させることによって，ユーザ（享受者）が自ら必要なアプリを動的に選択し，利用することで解決する．最後に，課題(c) は，フレームワークが，IoE サービス・デバイス間のセキュリティをチェックする機能を提供することで解決する．

本稿では特に，課題(a) を解決するための UGC 開発エコシステムを実現するために必要となるフレームワークについて検討した．フレームワークは主に，以下の 3 点のアプローチでサービス開発者の生産性を高める．

(i) IoE サービスにおいて各デバイスが担うタスクを「機能」単位で API 化することで，サービス開発者は機能のマッシュアップによって自由に IoE サービスを開発できる．デバイスの基本機能を API として切り出した上で，API をマッシュアップすることによって作成されるサービスについても API 化する仕組みを提供する．これにより，高機能な API（メタ API）を階層的に次々と生成し，サービス開発者どうしで共有することが可能となる．

(ii) 物理的なデバイスの機能（API）をカプセル化し，IoE サービスを担うそれぞれのデバイスを単位としたセキュリティチェック機構を提供する．サービスをカプセル同士の連携という概念で実現することで，複数エンティティによるデバイス利用の競合，エンティティ間のアクセス権限の制御，プライバシ情報の管理を効果的に実現可能である．これにより，サービス開発者は安全な IoE サービスを開発できる．また，メタ API ならび IoE サービス自身も同様の枠組みによってメタカプセル化する．

(iii) ユーザに IoE サービスを提供する物理デバイスは，ユーザの移動とともに変化する．ユーザの所在に応じて仮想空間上のデバイスカプセルと物理空間内のデバイスをその都度動的に接続する仕組みを提供することにより，サービス開発者から物理デバイスを隠蔽する．これにより，サービス開発者は API やメタ API を抽象度の高い形で取り扱うことが可能となり，具体的なデバイスに依存しない IoE サービスを開発できる．

以下では，2 章で関連研究を紹介し，各課題に対する解決状況を分析する．次に，3 章で関連研究を踏まえ，提案フレームワークが備えるべき要件の詳細について述べる．最後に，4 章でまとめと今後の方針について述べる．

2. 関連研究・技術

IoE サービス開発のためのフレームワーク・基盤に関する研究・技術を紹介し，課題(a) ,(b) ,(c) に対する解決状況を分析する．

2.1 SMuP

SMuP（Sensor Mashup Platform）[3]は，複数のセンササービスをマッシュアップして，仮想センササービスを開発可能な基盤である．これにより，複数のセンササービスを組み合わせて，高度なコンテキスト推定を行う仮想センササービスを開発できる．サービス開発者は，これを用いることで，複雑なコンテキスト推定のロジックを自ら記述せずとも，推定結果に基づくロジック（アクチュエータの制御）にのみ注力して IoE サービスを作成できる．しかし，アクチュエータの制御に関しては，マッシュアップをサポートしておらず，高度な制御を行うには，サービス開発者が複雑なロジックを記述する必要がある．

この研究は，マッシュアップによって作られた抽象度の高い API（仮想センササービス）を提供することで，サービス開発者の生産性を高めることに成功しており，課題(a) の解決に効果がある．センサだけでなく，アクチュエータの制御も含めて，マッシュアップを用いて抽象度の高い API を提供することができれば，さらに生産性を高めることが可能と考えられる．

2.2 AllJoyn

AllSeen Alliance の AllJoyn フレームワーク[4]では, Wi-Fi, Ethernet，Serial，PLC のいずれの通信規格でも IoE サービスから対応するデバイスを操作できる．AllSeen Alliance ワーキンググループは，デバイスの機能を，インタフェースという形で標準化しており．異なるデバイスであっても，共通の機能は同じ API（インタフェース）で利用することが出来る．これにより，デバイス構成が異なる環境でも，使用するインタフェースを備えたデバイスさえ存在すれば，同じ IoE サービスを実行できる．IoE サービス 1 つあたりの，適用範囲を広げることが出来るため，IoE サービスの流通を促進できる．

IoE サービスは，フレームワークが提供する機能を利用して，特定のインタフェースを備えたデバイスを検索し，連携させることが出来る．検索結果から連携させるデバイスを決定する部分は，サービス開発者が記述する必要がある．この処理をフレームワーク側で全て行うことが出来れば，より生産性を高めることが出来ると考えられる．

AllJoyn で標準化しているインタフェースは，例えば，照

明やテレビといった各種デバイスの最小の機能を抽象化したものである．したがって，サービス開発者は，デバイスの基本的な機能の組み合わせで IoE サービスを開発しなければならない．

AllJoyn に対応していないデバイスも，AllJoyn の枠組みで連携させるためのデバイスブリッジという仕組みがフレームワークから提供されている．IoE サービスの適用範囲が広がり，課題(a) に効果がある．

Android，iOS など様々なプラットフォーム向けのバージョンが用意されており，作成した IoE サービスをアプリ市場で公開できる．実際に，AllJoyn を用いたアプリケーションが公開されている．このように，UGC 型のエコシステムで IoE サービスを流通させることは, 課題(a) ,(b) に対して，効果があると考えられる．しかし，ユーザの周囲の環境に応じた検索機能・リコメンド機能が必要となる．この理由は, IoE サービスは, ユーザの身の回りの物理デバイスを，リソースとして使用するため，基本的に端末内で閉じている通常のアプリケーションと比べて, リソースが存在せず，実行できないといった場面が増えると考えられるためである．これは，インスト―ル済みの IoE サービスから，実行するものを選択する場面でも同様のことが言える．

セキュリティでは，デバイス毎にインタフェース単位のアクセス制御が可能である．暗号通信や認証の大部分の処理はフレームワークが機能を提供しているが，認証情報（PIN やパスワードなど）は，アプリケーションレベルで管理・提供しなければならない．したがって，コンテキストに応じたアクセス制御を行うには，サービス開発者が記述しなければならない．この機能をフレームワーク側で用意することで，生産性を高めることと，セキュリティをサービス開発者に判断させずに記述できるため，課題(a) ,(c) に効果がある．

2.3 HomeKit

HomeKit[5]は，HomeKit Accessory Protocol 対応デバイスを制御する iOS 向けの IoE サービスを開発するフレームワークである．AllJoyn と同じく，デバイスの機能の標準化をしている. また, デバイス管理を行うためのアプリ「Home」が用意されており，ユーザ（享受者）は，デバイスの導入・設定を容易に行える．アクセス制御は，クラウドのアカウントを用いて認証し，デバイスを制御可能である．また，遠隔から自宅のデバイスを操作するための仕組みも用意されている．

「Home」は, デバイスの管理・導入を容易にすることで，IoE サービスの普及に効果がある．また，クラウドと連携する事でデバイスを遠隔操作が可能となることで，より多くのコンテキストで IoE サービスを利用することが出来るため，課題(a) に効果がある．

2.4 エージェントベース分散処理基盤

竹内ら[6]が提案したエージェントベース分散処理基盤では，デバイス，ユーザ，場（空間），といった要素をエージェントとして統一的に抽象化している．特に,「場」を抽象化したエージェントは，その空間内の競合検出・調停を行う．これは課題(c) に効果がある．

IoE サービスは，ルールによるエージェントの連携で実現する．構成要素となるエージェントは，環境に応じて置き換えられるため，1 つの IoE サービスを様々な環境で利用可能である．さらに，実空間の動的な環境変化や負荷の増減といった状況に応じて，エージェントを物理ネットワーク上の異なる機器に再配置することで．高い拡張性を確保しつつ継続性のあるサービスを提供可能である．

この研究では，デバイスだけでなく人間（ユーザ）もエージェントとして抽象化して連携させている．人間を計算資源として扱うことで，高度なサービスが実現できる場合があり，この課題(a) の解決に効果があると考えられる

3. 提案フレームワークの要件

この章では, 1.2 節で述べたフレームワークの機能(i), (ii), (iii)について，それぞれの機能を実現するための要件をまとめる．

3.1 節「API の整備」では，(i)について述べる．様々な目的に応じた機能・抽象度の API を大量に用意することで，IoE サービスの開発を容易にするための要件について述べる．

3.2 節「セキュリティ」では，(i)について述べる．セキュリティに関する機能をフレームワーク側で用意することで，サービス開発者にセキュリティの判断を強いずに開発を可能とし，IoE サービスの開発を容易にするための要件について述べる．

3.3 節「動的バインディング」では，(i)について述べる．

表 1　IoE サービス開発に関係する要素

要素の種類	説明
*-able	物理デバイスの基本的な API や，それらをマッシュアップした抽象度の高い API の仕様（インタフェース）.
ドライバ	デバイス依存のプロトコルや命令セットを隠蔽し，物理デバイスの機能（API）を，*-able として実装するプログラム.
メタ API	*-able をマッシュアップしたサービスで，新たに別の*-able を実装するもの.
IoE サービス	*-able をマッシュアップしたサービス.

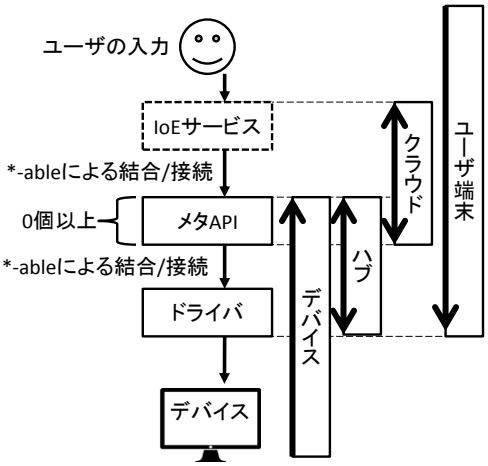
-able とは，IoE サービスを開発する際に利用できる API の外部仕様（インタフェース）である．本研究では，「～の機能を持っている」という意味で，「-able」という命名規則で表す．例えば，「ベルのような音を出せる」という API のインタフェースは，「Ringable」と名付ける．AllJoyn や HomeKit におけるデバイス機能の標準化は，デバイスの基本的な機能という，抽象度の低いものだけであったが，2.1 節で述べたように，センサやアクチュエータの機能を組み合わせた抽象度の高い API（*-able）を提供することができれば，生産性を高めることが可能である．

ドライバは，デバイス独自のプロトコルや命令セットを隠蔽し，物理デバイスの機能（API）を，*-able として実装するプログラムであり，基本的にデバイスベンダが開発する．メタ API は，*-able をマッシュアップして実現するサービスのうち，新たに別の*-able を実装するプログラムを指す．メタ API は，他のメタ API や IoE サービスから再利用できる．これにより，高機能な API（メタ API）を階層的に次々と生成していくことが可能となる．IoE サービスは，メタ API と同じく，*-able を用いて作成するプログラムであるが，こちらは新たに，*-able を実装しなくても良く，ユーザ（享受者）から利用されるためのものを指す．

IoE サービスやメタ API の開発者は，利用したい*-able のリストを記述するだけで，それらの *-able が全て揃っているという前提のもとにプログラミングができる．これにより，API やメタ API を抽象度の高い形で取り扱うことが可能となり，具体的なデバイスに依存しない IoE サービスを開発できる．このような記述によるプログラミングを実現するためには，IoE サービス，メタ API が，自身が利用する*-able を提供しているドライバ・メタ API と，物理的な隔たりを超えて，自動で接続されるような仕組みが必要である．ユーザ入力をデバイスへ伝える要素と配置可能性を**図 1** に示す．
この詳細は 3.3 節「動的バインディング」で述べる．

次に，ドライバ・メタ API が実際に API を提供するための仕組みを述べる．フレームワークが提供するセキュリティ機能やポリシ管理機能が，ドライバ・メタ API と共にカプセル化され，これにより，実際に API を提供することができる．ドライバやメタ API は，*-able を実装するコードだけを記述すればよく，エンティティ間のアクセス権限の制御やプライバシ情報の管理などは記述しなくてよい．これにより，ドライバ・メタ API の開発の負荷が低減されている．

図 2 にデバイスカプセル・メタカプセルの概要を示す，デバイスカプセルは単一のデバイスを仮想オブジェクト化したものである．メタカプセルは，複数デバイスを連携させて機能を提供するためのカプセルである．メタカプセルの機能を実行すると，連携先のデバイスカプセルへ，*-able の API コールが発信される．メタカプセルは，ハブやホームサーバといった，家や空間を代表するスマートデバイスで動作させ，空間内の競合管理やプライバシ管理なども行う．



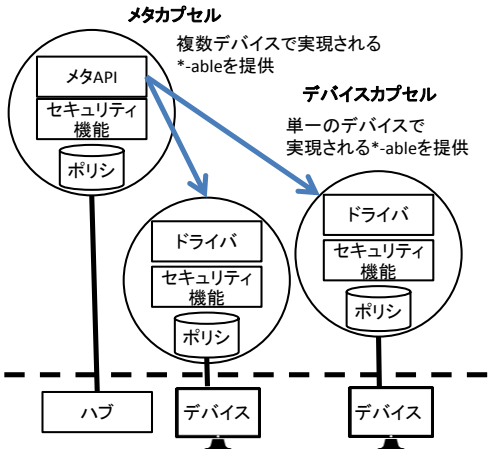


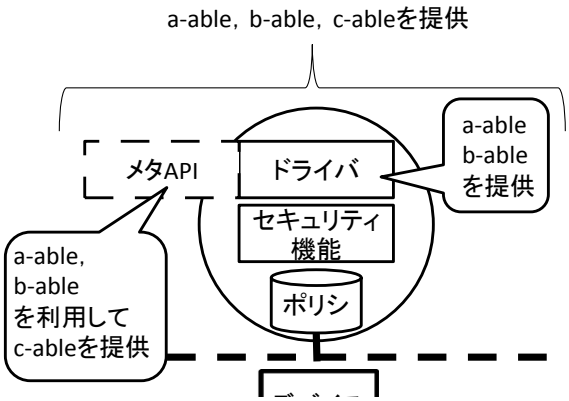
図 2　デバイスカプセル・メタカプセル概要

さらに，メタ API の利用形態について説明する，図 2 では，メタ API はメタカプセルに組み込まれ，連携サービスによる API として働いていた．メタ API の利用形態は，このほかにもデバイスへの動的な機能追加がある．**図 3** にメタ API によるデバイスの機能追加例を示す. a-able と b-able を提供するデバイスに対して，a-able と b-able を利用して c-able を実現するメタ API が組み込まれることで，デバイスが提供する機能に c-able を追加することが出来る．さらに別の利用形態を説明する．**図 4** の例では, a-able と b-able で c-able を実現するメタ API1 と，p-able と q-able で r-able を実現するメタ API2 をライブラリとしてメタ API の内部に組み込むことで，メタ API3 は，a-able，b-able，p-able および q-able を用いて x-able を実現するメタ API となっている．もしも，c-able と r-able を動的に利用する場合は，各々の実装が別のメタ API によるものとなる可能性があるが，この方法ならば，メタ API1 とメタ API2 による c-able と r-able の実装を確実に利用することが出来る．これは多様

性を消してしまっているようにも思えるが，動作を確認した実装を確実に利用することで，信頼性が上がるとも考えられる．また，図中の記述の通り，c-able と r-able が非標準で普及していない場合でも，連携する際に，標準化され普及している*-able を用いることができる利点がある．

3.1 API の整備

この節では，デバイスの基本機能だけでなく，それらをマッシュアップして実現するような，高機能・高抽象度の様々な*-able を大量に用意し，サービス開発者に提供するための要件を述べる．3.1.1 項「自由な*-able の定義」は，大量の*-able を定義していくための要件である． 3.1.2 項「自由なドライバの実装」は，デバイスが持つ機能を，より多様な*-able として提供可能にするための要件である．3.1.3 項「自由なメタ API の実装」は，高機能・高抽象度の API を効率良く実現するための要件である．3.1.4 項「*-able, ドライバ，メタ API の管理」は，開発者に大量の*-able,



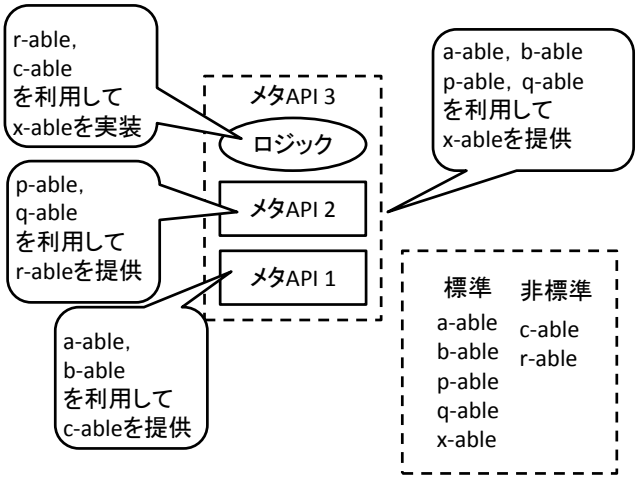


図 4　メタ API のライブラリ利用

3.1.1 自由な*-able の定義

サービス開発者が求める API は多種多様であり，これをフレームワーク設計者が網羅することは困難である．したがって，様々なユーザが自由に，機能の定義（*-able）を作成・共有できる仕組みが必要である．

3.1.2 自由なドライバの実装

デバイスベンダだけでなく，ユーザでもドライバを開発可能とする必要がある．しかし，*-able は大量に存在するため，デバイスベンダだけでは想定しきれない機能（利用方法）が存在し得る．例えば，「寝ている人の目を覚ます（Wakeup-able）」という機能が定義されているとする．目覚まし時計がこの機能を持っていることは簡単に想定されるが，テレビやラジオといったデバイスも，番組を大音量で流すことで人を起こすことができるため，この機能を持っていると判断することもできる．しかし，このような使い方は，デバイスベンダは想定できないかもしれない．こ

様々な目的・抽象度の機能の定義が存在するた
ベンダだけでなく，ユーザのアイデアも取り
する仕組みが必要である．また，2.2 節で述
のように，フレームワークの規格に対応して
スでも，利用可能とするためにも，ユーザに
開発を可能にすべきである．

メタ API の実装

，*-able を作成・実装していくためには．メ
々と作成・再利用されていく必要があるため，
が作成・共有できる仕組みが必要である．

ドライバ，メタ API の管理

によって，次々と大量に作られていく*-able,
タ API のうち，有用と認められたものを，標
させる仕組みが必要である．

インタフェースを標準化し，普及させること
を備えたデバイスが増え，これを用いて作成
I，IoE サービスの適用範囲が広がるためであ

ティ

，セキュリティに関する機能をフレームワー
意することで，サービス開発者にセキュリティの
いずに開発を可能とし，IoE サービスの開発を容
ための要件について述べる．
コシステムでは，サービス開発者と享受者が異な
キュリティ機構をサービス開発者に任せると，悪
開発者による攻撃や，設計ミスによる脆弱性など
がある．したがって，フレームワーク側で様々な
ティ機能を用意することで，セキュリティの確保
また，IoE サービス開発の負荷も低減される．

セス制御

キストに応じた，アクセス制御を行う必要がある，
カプセルで行うものと，空間（メタカプセル）で

行うものが必要である．デバイスレベルの例では，デバイスがビジー状態の間は，持ち主以外のユーザに対するアクセス権限を一時的に剥奪するといったもの，空間レベルの例では，深夜帯に，音の鳴るデバイスを禁止することで，睡眠が妨げることを防ぐといったケースがある，

3.2.2 競合管理

1 台の機器にたいして，矛盾した命令が同時に発行される機器競合と，空間の要素対して相反する命令が同時に発行される環境競合ともに処理可能な仕組みが必要である．特に，このフレームワークではマッシュアップによる高い層の命令にたいして，チェックを行う必要がある．

3.2.3 プライバシ管理

アクセス制御のポリシはプライバシ情報となり得る．例えば，病気・趣味など関する内容がポリシから読み取れる可能性がある．したがって，デバイスの貸し出しや，ゲストユーザによるデバイスの利用といった場面で，プライバシ情報となるようなポリシを公開せずに，デバイスのポリシ部屋のポリシを遵守する仕組みが必要である．

3.3 動的バインディング

この節では，IoE サービス特有の，リソース（物理デバイス）の可用性の問題や，移動性に対応する仕組みをフレームワークで用意することで，IoE サービスの開発を容易にするための要件について述べる

3.3.1 デバイスの自動設定

2.3 節，HomeKit の「Home」のように，デバイスの導入・設定を，ユーザ（享受者）が簡単にできる仕組みが必要である．この設定方法が煩雑だと，ユーザ（享受者）が IoE サービスの導入に抵抗を感じ，普及が困難になるためである．

3.3.2 デバイスの追従

デバイスには，物理的な移動によるネットワークの変更や，電源切れなどが発生する．竹内ら[6]は，エージェントの動的配置によって，こういった物理的な状態変化に対応できる基盤を提案していた．本方式においては，デバイスカプセルと，デバイス間の接続を維持する仕組みを，フレームワーク側で提供することで，ベンダやユーザによるドライバ開発の負荷を低減する．

3.3.3 動的バインディング

2.2 節より，IoE サービスと，サービスが利用する API を提供するデバイスとの接続を，フレームワーク側で行う必要がある．これにより，IoE サービス内でデバイスの検索や候補リストからの選択といった処理を不要とし，生産性を向上させる．また，IoE サービスが，純粋な機能のマッシュアップのロジックとなるため，API としての再利用が促進される．

4. おわりに

現在のスマートフォンアプリ市場のように，UGC(User Generated Contents)型のサービスアプリ開発エコシステムを形成し，ユーザによる IoE サービスの作成を促進するためのフレームワークを提案した．

今後は，実装を進めていき，動的バインディングのインテリジェンス化, 悪意のある IoE サービスへの対策の検討．マッシュアップによる IoE サービスの性能評価などを進めていく．

謝辞 本研究において貴重なご意見を頂きました，株式会社富士通研究所 司波章氏に厚く御礼申し上げます．

参考文献

[1] 総務省：総務省｜平成 28 年版 情報通信白書｜IoT 時代における ICT 産業動向分析，入手先〈http://www.soumu.go.jp/johotsusintokei/whitepaper/ja/h28/html/nc120000.html〉（参照 2017-01-26）
[2] IBM Bluemix：IBM developerWorks 日本語版 ：IBM Bluemix，入手先〈http://www.ibm.com/developerworks/jp/bluemix/〉（参照 2017-01-20）
[3] 坂本寛幸，井垣宏，中村匡秀：センササービスのマッシュアップを実現するサービス指向基盤の提案，電子情報通信学会技術研究報告，Vol.109，No.276，pp.23-28（2009）．
[4] AllSeen Alliance：AllSeen Alliance，入手先〈https://allseenalliance.jp/〉（参照 2017-01-20）
[5] Apple Developer：HomeKit デベロッパガイド，入手先〈https://developer.apple.com/jp/documentation/HomeKitDeveloperGuide.pdf〉（参照 2017-01-20）
[6] 竹内 亨，坂野遼平，馬越健治ほか：エージェントベース分散処理基盤の提案と BMI 応用サービスへの適用による評価，情報処理学会論文誌，Vo.55，No.2，pp.681-694（2014）．