博　士　論　文

Study on the Architecture of the Fusion of
Human-Centric Applications and the Internet of Things
（ヒューマンセントリックアプリケーションと
インターネットオブシングスの統合アーキテクチャに
関する研究）

淺野　智之

論文要旨

現在、Internet of Things（IoT）やユビキタスコンピューティングと呼ばれる、コンピューター応用のモデルが全世界で注目されるようになってきている。情報処理系システムをつなぎ、その結果として各システムのユーザーである人間同士がコミュニケーションをするために利用されていたきたのが今までのインターネットである。これに対し、IoT ではコンピューター組込みのモノ（デバイス）を接続し、モノの情報取得やモノのコントロール、さらにはモノの間でのコミュニケーションによる自動連携動作などの実現を目的にしている。

この現実空間中の人間やモノもすべてネットワークで繋ぐような概念自体は 1980 年代から Highly Functionally Distributed System (HFDS)などによって提案されていたが、当時は HFDS を実現するための半導体技術やインターネットをはじめとするネットワークインフラストラクチャーの整備が不完全であったため、実現されていなかった。現在はこれらの技術が発展し、IoT やユビキタスコンピューティングが実現可能になってきている。

しかし、一方実現性が高まる中で実際的な問題がクローズアップされている。インターネットがそうしたように、国や所属や応用分野の枠を超えてモノをつなぎその制御を可能とするためには、セキュリティだけではなく、権限の与え方（アクセスコントロール）を適切に取り扱える仕組みが用意されている必要がある。なぜならば、例えばデバイスの所有者のみネット経由で遠隔でそのデバイスを制御できるが、他者からは全くアクセスできないといった単純なセキュリティでは、単なるネット経由リモコン程度の応用しか実現できないからである。逆に適切なアクセスコントロールが実現できれば、家庭の機器の消費電力やさらにはその制御権限の一部を電力会社にオープンにして、細かい電力のデマンド・サイド・コントロールを可能にし、地域全体の省エネや、災害時の適切な電力管理も可能になる。

また、モノのベンダーによって API の仕様が異なる場合が多いため、ベンダーの枠を超えた IoT サービスを実現するためにモノ毎の API 仕様をプログラム開発者が理解してシステムを構築する必要があり相互運用性が低いという問題がある。

本論文ではこの問題を解決するためのアーキテクチャを提案した。特徴として以下の 2 点が挙げられる。

1） モノと人間からなるサービスを連携させ総体として扱う為には、利用者を含む

関連プレーヤーの位置や場所の取り扱いが重要であるため、位置情報や場所情報を取り扱うための仕組みを提供する

2) 多くのサービスを接続する際にはアプリケーションの作りやすさの観点から API の標準化が重要であるが、既存のデバイスの API を再設計することは負担が大きいため、クラウドでリバースプロキシを実現し、既存のデバイスに手を加えなくとも標準化された API でモノにアクセス可能な仕組みを提供する

以上のような観点で提案したアーキテクチャ Huot について評価を行い、有効性を示した。

謝辞

本研究を進めるために長期にわたって丁寧にご指導くださった坂村健教授、越塚登教授、小林真輔様、湧田雄基助教、別所正博様には深く感謝いたします。また、関連プロジェクトを遂行するためにご協力くださった山田純様、矢代武嗣様をはじめとする YRP ユビキタス・ネットワーキング研究所の皆様、ならびにユーシーテクノロジ株式会社、坂村・越塚研究室の皆様にも御礼申し上げます。

目次

図一覧

表一覧

第 1 章

序論

1.1. 背景

1.1.1. Internet of Things の基盤技術の発展

現在、Internet of Things（IoT）やユビキタスコンピューティングと呼ばれる、コンピューター応用のモデルが全世界で注目されるようになってきている [1]。情報処理系システムをつなぎ、その結果として各システムのユーザーである人間同士がコミュニケーションをするために利用されてきたのが今までのインターネットである。これに対し、IoT ではコンピューター組込みのモノ（デバイス）を接続し、モノの情報取得やモノのコントロール、さらにはモノの間でのコミュニケーションによる自動連携動作などの実現を目的にしている [2]。

この現実空間中の人間やモノもすべてネットワークで繋ぐような概念自体は 1980 年代から図 1.1 に示すような超機能分散システム Highly Functionally Distributed System (HFDS)などによって提案されていたが、当時は HFDS を実現するための半導体技術やインターネットをはじめとするネットワークインフラストラクチャーの整備が不完全であったため、実現されていなかった[4]。現在は、コンピューターに関連する技術が発展することにより、情報処理系システムだけではなく、様々なモノに埋め込まれたコンピューターでもインターネットに接続している。

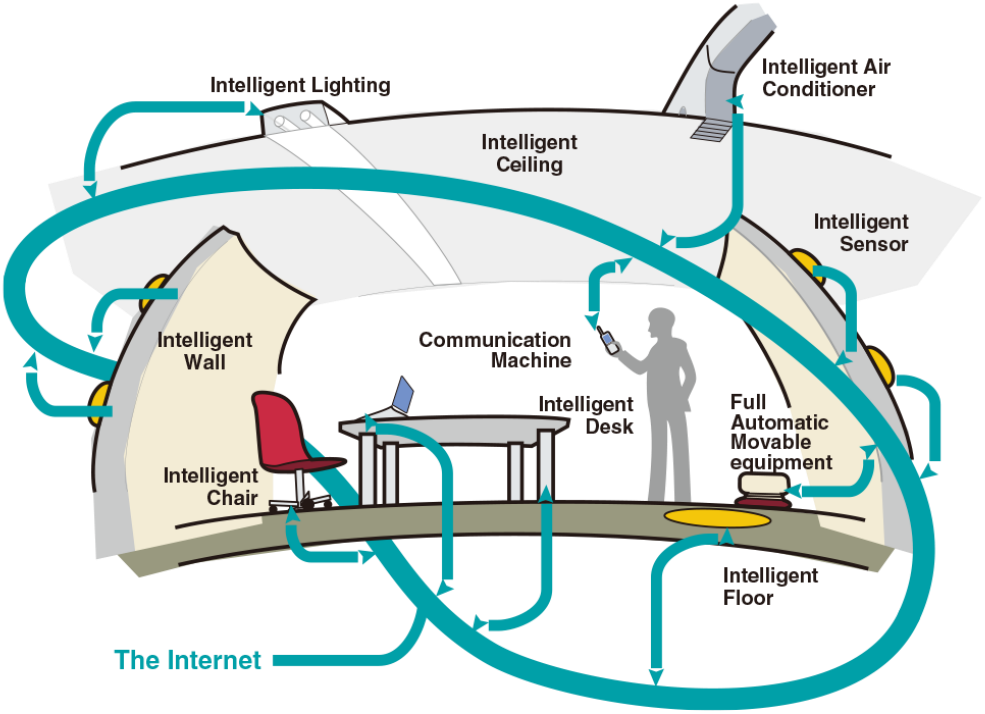


図 1.1 TRON プロジェクトが目指す超機能分散システム
Highly Functionally Distributed System （ [3]を元に修正した [4]から引用）

現在広く普及しているインターネットプロトコルである IPv4 は、IP アドレスが 32bit であり、2^{32} 個しかないことから、インターネットで利用する IP アドレスであるグローバル IP アドレスと、プライベートなネットワークで利用するプライベート IP アドレスの 2 種類を相互に変換することで、プライベートなネットワークをインターネットへ接続している。この IPv4 に代わる技術として普及しつつあるのが IPv6 である。IPv6 では、IP アドレスが 128bit であり、2^{128} 個の IP アドレスがあるため、ネットワークに接続されているすべての機器を直接インターネットに接続することが可能となった。

一般論としてモノに組み込まれるコンピューターシステム（組み込みシステム）は、情報処理系システムと比較すると計算資源に制約がある場合が多い。例えば CPU について着目すると、情報処理系システムでは現在の主流は 64bit CPU であるが、組み込みシステム系ではシェア 60%を誇る TRON 系 OS である T-Kernel 2.0

が 32bit CPU を対象としており、さらにμT-Kernel 2.0 は 16bit や 8bit CPU を対象としているように、計算資源に対する想定が全く異なっている [5]。また組み込みシステムは設置形態に制約があることがある。組み込みシステムは家電製品のように情報処理系システムと同様に電源に接続するデバイスに組み込まれる場合だけではなく、バッテリーで駆動させる必要がある場合もあり、さらにネットワークにも有線での接続が難しい場合がある。このようなシステムでは低消費電力無線通信を採用されている。従来は IrDA や Bluetooth や ZigBee などによって、インターネットにつながらない範囲内で完結するような通信方式が採用されていた。しかし現在では、6LoWPAN と呼ばれるプロトコルによって、情報処理系システムだけではなく、組み込みシステムも IPv6 で直接インターネットと接続されるようになりつつある。

IoT はこのような技術背景に基づいて実現性が高まっているが、それに合わせて実際的な問題がクローズアップされている。インターネットがそうしたように、国や所属や応用分野の枠を超えてモノをつないでその制御を可能とするためには、セキュリティだけではなく、権限の与え方（アクセスコントロール）を適切に取り扱える仕組みが用意されている必要がある。なぜならば、例えばデバイスの所有者のみインターネット経由で遠隔でそのデバイスを制御できるが、他者からは全くアクセスできないといった単純なセキュリティでは、単なるインターネット経由のリモコン程度の応用しか実現できないからである。逆に適切なアクセスコントロールが実現できれば、家庭の機器の消費電力やさらにはその制御権限の一部を電力会社にオープンにして、細かい電力のデマンド・サイド・コントロールを可能にし、地域全体の省エネや、災害時の適切な電力管理も可能になる。

また、モノ同士が IPv6 によって相互に通信可能となったとしても、モノの提供する機能や能力、接続方法などが分からなければ、自動でモノ同士が連携することはできない。現状では、モノの API が標準化されていないため、モノを作ったメーカー毎に API の仕様が異なる場合が多く、あるサービスを実現するためにメーカーの枠を超えて相互連携させるようなことは実現できない場合が多い。また API の仕様を標準化して相互運用性を高めるためには、モノの種類に合わせて API や機能等の仕様を策定し、策定した仕様に則ってモノを作る必要があるため、メーカーが独自の機能を追加することは難しくなる。メーカーは自社製品に付加価値をつけて市場に出すことが競争原理に基づいた考え方であるため、このような標準化を行うことが難しい。

モノ同士や、モノとその他の情報を組み合わせて、実世界にフィードバックを行うようなサービスを実現するためには、これまでに述べたような理由から API が標準化されていないため、モノ毎の API 仕様をプログラム開発者が理解してシステムを構築する必要があり、相互運用性が低いという問題がある。現状の IoT モデルではその名の通り「モノ」の連携しか考えられていないものがほとんどであり、モノ以外が実現するサービスが混在する今後の社会アーキテクチャにおいて、これらを総体として取り扱うためのモデルにはまだ取り組まれていない。

本研究の究極的な目標は、HFDS を実現するための基盤技術として、ユーザーの属性情報を組み合わせてモノが連携動作するためのプラットフォームを実現することである。

1.1.2. 個人の属性情報と IoT

IoT では、モノとモノが協調動作するための仕組みが様々に検討されているが、HFDS を実現するためには、モノとモノが連携する仕組みだけを構築するだけではなく、言語や年齢、身体的な属性、嗜好など、様々な個人の属性情報を取得し、個人に合わせたサービスを実現する仕組みが必要である。様々なサービス間で個人の属性情報を流通させることの重要性は、政府主導で進められている MyData [6]や midata [7]、アカデミック主導で進められている Project VRM [8]など、ユーザーの属性情報を共有するための様々なプラットフォームが実現されていることからも明らかである。このようなプラットフォームのことを Personal Data Store（PDS）と呼び、ユーザーの属性情報を蓄積し、必要に応じてサービスへ提供することができる。なおユーザーの属性情報の提供はユーザーの主導の下で管理されるべきであり、どのサービスに対していつ、どの情報を提供するかを適切に管理する仕組みを VRM（Vendor Relationship Management）システムと呼ぶ。一般的には PDS と VRM を兼ねたシステムとなっている場合が多い。

個人の属性情報は、情報源の種類に応じて、以下のように分類が可能である。

- ユーザーが自分自身で登録する属性情報
  - 例えば、氏名や性別、利用可能言語など
- ユーザーが実生活でサービスを利用することによって自動的に生成される属性情報
  - 例えば、通院履歴や決済履歴など

- ユーザーが保持するデバイスから収集する属性情報
  - 例えば、位置情報や消費カロリーなど

HFDS を実現するためには、このような個人に関する様々な属性情報を活用し、ユーザーに合わせてサービスを提供できるための仕組みが重要である。先述の個人の属性情報のうち、IoT を実現する上で最も重要な属性情報は、ユーザーが保持するデバイスから収集する属性情報である。この理由として、IoT が実現されることでユーザーの周辺環境やユーザー自身が保持するデバイスはネットワークに接続され、常にユーザーの属性情報を取得できる環境が構築されるため、IoT の実現に当たって考慮する必要があるためである。

1.1.3. Human-centric なアプリケーションと IoT

Human-centric なアプリケーションとは、IoT やクラウドコンピューティングなど、様々な計算能力を持つコンピューターシステムを組み合わせるようになっている今日のコンピューターシステムのように複雑化する高度なシステムを利用して構築された、システムの利用者であるユーザー側の視点に立って利用しやすくなるように意識して設計されたアプリケーションである。例えば図 1.2 は、Human-centric なアプリケーションを中心として、周囲の様々なサービスが連携して動作してる様子を表した例である。

図 1.2 Human-centric なアプリケーションの例 （ [9]より引用）

HFDS を実現する上で必要となる要件は、ユーザーの要望に応じて、または要望を自動で予測して情報を提示したり、環境をコントロールしたりすることである。また、このような情報提示を実現する仕組みや環境をコントロールするための仕組みは、ユーザーは意識するべきではなく、それぞれが実現する機能単位で分割されたサービスとして見えるようになっているべきである。さらには、ユーザーはこのようなサービスを任意の単位で組み合わせて、プログラミングできるようになっているべきである。

こういった要件を実現するためには、サービスが提供する機能が GUI や CLI などのような直接ユーザーに見える形で提供されることではなく、API として提供されていることが重要である。API として提供されていれば、アプリケーションの開発者がユーザーの要望に応じて、例えば、荷物を持っていて両手がふさがっていて一切端末などのボタンを操作できない状態の場合や目の見えない方向けに、API で取得した結果を音声で案内することを実現したり、本来はユーザー同士が情報伝達をするためのチャットシステム内の発言をトリガーに環境をコントロールするようなアプリケーションを実現したりすることが可能となる。さらには、こうした自由に API を利用して組み合わせる事ができる環境を構築することで、ユーザー自身が自分でアプリケーションを DIY（Do It Yourself）で構築できるような可能性も広が

る。

このように、Human-centric アプリケーションを実現するためには、サービスが提供する機能が適切に API として提供されることが重要である。特に Internet of Things のコンテキストでは、API は Web API として一般的な RESTful な API として構築されていることが求められる [10]。さらに、先述の通り、ユーザーの視点に立ってアプリケーションを構築するためには、ユーザーの個人の属性情報を組み合わせて個人に適応した形でサービスを実現することが求められる。従って、Human-centric アプリケーションを実現するためには、ユーザーの個人の属性情報を扱えることと、ネットワークに接続されたモノの API が統一的に操作できるようになっていることが重要であると言える。

1.2. 目的と研究の概要

1.2.1. ユーザー属性情報連携システム

本研究の目的は、Human-centric アプリケーションと Internet of Things の統合を実現するためのアーキテクチャの構築のための方法論を明らかにすることである。このために必要なこととして、ユーザー属性情報を様々なサービスで個別に保有するのでは無く、一カ所で管理することで、ユーザーが様々なサービスを利用する都度、ユーザー属性情報を入力する必要がなくなり、さらにどの情報をいつどのサービスへ提供したか、どのサービスにはどの情報を提供して良いか等を一元管理できるメリットがある。本研究では、このユーザー属性情報連携システムを提案し、実装し、実証実験を含む様々な観点から評価を行った。

このシステムで扱うユーザー属性情報の例として、本論文では、1.1.2 節で述べたとおり、個人の属性情報としてユーザーが保持するデバイスから収集する属性情報が重要であると考えている。このような情報のうち、HFDS を実現するための最も重要な属性は、様々なサービスを提供するための基盤となる位置情報であると考えられるため、位置情報について着目した [11]。

ユーザー属性情報の取得例として、まず歩行者の位置情報を推測するための手法について説明を行う。歩行者の位置を推測するための手法として、複数のセンサーを組み合わせて歩行パターンを検出して、歩行者の位置をインクリメンタルに推測するデッドレコニングと呼ばれる手法を利用した [12]。このデッドレコニングによ

って位置情報を推測するために、センサーモジュール（Pedestrian Dead-reckoning Unit：PDU）を作成した。この PDU は、センサーの生データを抽象化して歩行者デッドレコニングに必要となる情報を Bluetooth 経由で送出する。Service Oriented Architecture [13]の観点から、この PDU が他のサービスと情報を組み合わせて新たなサービスを構築できるように考慮したことと、帯域が限られている Bluetooth 経由で情報を送信し、さらに PDU はバッテリーで駆動するため、無線通信のためにバッテリーを消費したくないという観点から、このようにセンサーの生データではなく、抽象化した情報を送出している。

本提案手法の評価として PDU を利用して実際に歩行者の位置を推測するためのシステムを構築して評価実験を行った、Bluetooth 経由で送信するために、PDU のメッセージは抽象化したコンパクトなメッセージフォーマットを提案したが、この提案したメッセージフォーマット内に歩行者の位置情報を推測するために十分な情報が含まれていることを示した。この結果、個人の属性情報の一つである位置情報について、取り扱うための方法を示した。

1.2.2. IoT デバイス連携システム Colapi

本研究の目的は、前節でも述べたとおり、Human-centric なアプリケーションと Internet of Things の統合を実現するためのアーキテクチャの構築のための方法論を明らかにすることである。前節では個人の属性情報である位置情報の取り扱いについて示したことが、本節ではインターネットに接続されたモノ（IoT デバイス）同士の連携や、IoT デバイスを利用したアプリケーションを実現するために API を統一的に操作するためのプラットフォームを提案した。IoT デバイスはメーカーによって API の仕様が異なっている場合がほとんどであるため、このような API の統一的な操作を通して相互運用性を実用的なレベルで実現するためには、API の標準化を行うことが求められる。しかしながら、API の標準化を行うための労力には制約があり、さらに IoT ならではの難しさが存在している。例えば、IoT デバイスの種類を検討する場合には、一つの IoT デバイスが特定の 1 種類のデバイスの種類のみに限定されないことがあることや、IoT デバイスのメーカーが他社の IoT デバイスと差別化をする目的で、標準化された仕様に収まりきらない、より魅力的な新機能を追加することを強く望むことなどによって、複数の IoT デバイスのメーカーを集めて API の仕様を IoT デバイスの種類毎に標準化することは難しい。

この観点に基づいて、本研究では標準化された API のリクエストをクラウドサー

ビス上に構築したサーバーで受け取り、これを IoT デバイス固有の API 形式に変換して代理リクエストを行うシステム Colapi を提案した。この API 形式の変換のために、機械可読な形式で記述したデバイスプロファイルを提案した。このデバイスプロファイルは、IoT デバイスの種類毎に標準化された API セットを定義するための汎用デバイスプロファイル（generic device profile: GP）と、標準化された API をデバイス固有のプロファイルへと変換するための方法を記述したデバイス固有プロファイル（device specific profile: SP）の 2 種類を提案した。IoT デバイスのメーカーや開発者は、クラウドサービス上に設置した中央サーバーに、自身の作成した IoT デバイスのデバイスプロファイルを登録でき、新たな標準化 API を登録する場合には GP を、新たな IoT デバイスを作成した場合や、すでに本プラットフォームに登録済みの IoT デバイスに対して新たな GP に対応させて API を追加する場合には SP を、それぞれ登録することが可能である。例え IoT デバイスに 1 種類の API しか定義されていないとしても、一つの IoT デバイスには複数の GP を関連づけて登録することが可能となっており、これによって一つの IoT デバイスに対して、GP で定義された複数の API によってアクセスすることが可能となる。

本提案システム Colapi を実装し、パフォーマンス、ユーザビリティ、デバイスプロファイルの記述性、セキュリティ、スケーラビリティについて評価を行った。この評価の結果、IoT デバイス同士やアプリケーションと連携をするために実用的であることが示せた。

1.2.3. 本研究の成果

本研究の成果は、これまでで述べたような TRON プロジェクトの進める究極の目標である HFDS 実現のための要素技術の一つであるといえる、Human-centric アプリケーションと IoT の統合について提案し評価を行ったことである。具体的には、以下の 2 つの観点が本研究の成果である。

1) モノと人間からなるサービスを連携させ総体として扱う為には、ユーザーを含む関連プレーヤーの位置や場所の取り扱いが重要であるため、位置情報や場所情報を取り扱うための仕組みを提供する
2) 多くのサービスを接続する際にはアプリケーションの作りやすさの観点から API の標準化が重要であるが、既存のデバイスの API を再設計することは負担が大きいため、クラウドでリバースプロキシを実現し、既存のデバイスに手

を加えなくとも標準化された API でモノにアクセス可能な仕組みを提供する

1.3. 論文の構成

本論の構成は以下の通りである。まず第 2 章では、関連技術や関連研究を整理し、本研究の位置づけを明らかにする。次に第 3 章では、本研究の基盤となるユビキタス ID アーキテクチャについて説明する。第 4 章では、ユビキタス ID アーキテクチャに基づき、Human-centric アプリケーションと IoT を統合するための提案アーキテクチャ Huot を説明する。第 5 章では、Huot のうちユーザー属性情報連携システムについて説明を行い、さらに扱うユーザー属性情報の例として、ユーザーの位置情報を推定するための手法を説明する。その後、第 6 章では Huot のうち IoT のための API 連携フレームワークを提案する。最後に第 7 章ではこれらの成果について考察を行い、全体をまとめる。

第 2 章

IoT プラットフォーム実現のための課題と問題点

2.1. IoT プラットフォームの定義

IoT プラットフォームとは、IoT 化された様々なモノを管理し、情報取得をしたりコントロールしたりするためのプラットフォームである。IoT プラットフォームは、IoT デバイスをプログラムから制御することが容易であるために API を提供していることが必ず求められる。

住宅環境が IoT 化された未来の住宅となり、そこに住む住人が住宅内のモノを API 経由で自由にコントロールできるようになっていることを想定する。この場合、誰がどのモノをどのように制御できるかという情報を管理する必要があるといえる。IoT デバイスを管理するための情報として、以下のような情報が管理できるようになっているべきであると考えられる。

- IoT デバイス自身の情報
  - IoT デバイスの提供する API とその機能
  - IoT デバイスの IP アドレス
  - IoT デバイスのベンダー
  - IoT デバイスの型番
- IoT デバイスを設置した環境に関する情報
  - IoT デバイスの設置場所

- IoT デバイスの所有者
- IoT デバイスのアクセス権限
  - アクセス権限のリスト
    - ワンサイトアクセス許可、時限付きアクセス許可、個人から個人へのアクセス権限の譲渡など、様々なパターンに対応できるべきである。
- 外部の情報とのインテグレーション容易性
  - IoT デバイスは単体で動作するのではなく、環境の状態を識別する、個人の嗜好に合わせてコントロールするなど、様々な要求が考えられるため、そのような要求に対応できるように、外部の譲歩とのインテグレーションが容易になっている必要がある。

このような観点で IoT プラットフォームについて検討を行い、既存の課題と問題点の抽出を行った。

2.2. 既存の IoT プラットフォームに関する課題と問題点

デバイスプロファイルを利用してデバイスの協調動作を行うための仕組みは学術的な世界でも産業界でも取り組まれている。既存研究は、構文的アプローチ [14] [15] [16] [17]とセマンティックアプローチ [18] [19] [20] [21]に分類できる。

構文的アプローチは、特定のフォーマットに従ってデバイスの機能を表現する。一般的に構文的アプローチは記述の柔軟性に欠けており、デバイスメーカー独自の API を記述することができない場合が多い。OIC [15]は、リソースに制約のあるネットワーク向けのデバイスアーキテクチャであり、CoRE Link Format と呼ばれる形式に則ってデバイスの相互運用性を可能にする方式である。デバイスの API の記述は”/.well-known/core”というパスにアクセスをすると取得できる。クライアントは、デバイスの API 情報をこの記述から取得できるが、API の変換は実現できない。CoRE Link Format はフラットな構造となっており、この API がどの API 標準に従っているかを記述するための領域は用意されていない。提案手法では、この問題を解決するために、GP と SP の 2 種類に分割してデバイスプロファイルを定義している。SP はデバイスがどの標準に従っているかを表現する。さらに OIC は、デバイス側のフレームワークであり、デバイスはデバイスプロファイルの保管や処理、ク

ライアントからのリクエストに対するデバイスプロファイルの応答等をデバイス自身で行う必要がある。本手法では、デバイスプロファイルの管理はクラウド上のコンピューターで行うこととした。このため、デバイスに新たな API を追加する場合には、クラウド上にある Device profile registry に新たな SP を追加または修正するだけで対応可能である。

GSN [16]は、構文的アプローチによってセンサーの種別等のセンサーの情報等のメタデータを表現するためのフレームワークである。XGSN [18]は GSN で定義されたフレームワーク上で、データに対してセマンティクスを付与する仕組みを提供するフレームワークである。しかしこれらの手法は API の相互運用性を実現するための手法ではなく、本研究の目的とは異なる。

Banda ら [17]は、前方互換性を保つための IoT プロトコルとフレームワークを提案した。この手法は、OSI7 階層のアプリケーションレイヤーで独自のプロトコルを定義しており、この独自のプロトロルによって様々なサービスが通信を行う。一般的な Web サービスは HTTP を利用して実現されているが、この提案プロトコルは HTTP ではないため、彼らのプロトコルを Web サービスから利用するためには、彼らのプロトコルから HTTP への変換を行う必要がある。このため様々な Web サービスとマッシュアップするためには、プロトコルの変換が必要となってしまい、本研究の目的からは外れてしまう。

セマンティックな手法は、デバイスとその API について、Resource Description Framework (RDF)などによってセマンティクスを取り入れた記述を用いて表現を行う手法である。この手法は、デバイス間で受け渡しするデータにセマンティクスを付与した手法 [19] [20]や、デバイスの機能や性能や設置場所などの情報をセマンティクスで表現する手法 [17] [21]などが挙げられる。しかしながらどの手法でも、デバイスの相互運用性を実現できていない。

2.3. 位置情報に対する課題と問題点

IoT プラットフォームでは、柔軟なアクセス権限のコントロールや、ユーザーに合わせたサービスの提供が重要である。本節では、特にユーザーの位置情報に着目して、課題と問題点を整理した。

2.3.1. 人間に関する位置情報について

人間の位置情報の取り扱いについて考慮するべき項目として、人間の位置情報の取得方法（測位手法）と、位置情報の管理方法が挙げられる。

人間の位置を取得する方式として、大きく分けて 3 種類がある。

2.3.1.1. インフラストラクチャー型測位手法

1 種類目の手法は、周辺環境側にビーコン等を設置するような測位手法である。本研究では以下、インフラストラクチャー型測位手法と呼ぶ。この方式には、衛星を利用した測位手法 [22] [23] [24]や、電波を使う方式 [25] [26] [27]、超音波を使う方式 [28] [29]、不可視光を使う方式 [30] [31]、可視光を使う方式 [32]、画像を使う方式 [33] [34] [35]が存在する。

衛星を利用する方式は、測位に必要な衛星の準備が整っていれば、測位をするユーザー側では、衛星の発する電波を受信するデバイスさえあれば利用可能であり、特に GPS はすでに広く普及している。測位精度は GPS の場合、およそ 7.8m である [36]。しかし、衛星の電波が届かない屋内や、都市部などの空が開けていない環境では測位ができないという問題がある。

電波を使う方式は、測位のために電波を発するマーカーを設置する手法 [25] [26]と、すでに別の目的で設置されている Wi-Fi 等の電波を発する機器を利用する手法 [27]が存在する。測位精度は手法によって異なるが、UWB では数 10cm 程度の測位精度も実現可能であるが、Wi-Fi では数メートルである [37]。この手法による測位を実現するためには、新たに環境に対してビーコンを設置する作業が必要なため、すべての場所をこの手法だけでシームレスに実現することは難しい。また、Wi-Fi による手法についても、環境に対して新たにアクセスポイント等は設置する必要はないが、どこにアクセスポイントが設置されているか、どこでどの程度の電波強度で

受信可能か等の環境のマップをあらかじめ構築しておき、それをユーザー側の端末で利用して測位を行う必要があるため、環境に合わせた事前準備なしに測位を行うことは難しい。

超音波を使う方式は、超音波を発するマーカーを複数設置することで、数センチメートルオーダーの測位精度を実現できる。一方で、超音波の到達距離は電波よりも短いため、ビーコンの設置数は電波による方式よりも増えてしまう。さらに超音波は直進性が高いため、建物の構造を意識してビーコンを設置する必要があり、電波による方式よりもマーカーの設置の負荷が高い。

不可視光を使う方式は、赤外線の発信器を天井一面に敷き詰めてそれぞれが異なる ID を発する方式 [31]や、個人にユニークな ID を発する機器を持たせて、環境側に設置した受信機で ID を受け取り、位置を端末に返す方式 [30]がある。いずれの方式についても、すべての場所で測位を実現するためには設置コストの観点から向かないが、ピンポイントで位置を案内する等の用途には十分に利用できる。

可視光を使う方式は、LED が利用されている。赤外線による方式と同様に天井に位置に対応した ID を発信する LED が埋め込まれている [32]。利害得失についても、赤外線による方式と同様である。

画像を使う方式は、画像パターンにコードを埋め込んだ方式 [33] [34]と、自然特徴点抽出による方式 [35]が挙げられる。画像パターンにコードを埋め込む方式は、読み取り装置も画像パターンの装置もコストが低く、さらに画像パターンに対して電源を設置する必要もないため、非常に設置コストは低い。しかし、カメラを用いることから、コードの読み取りは画像パターンが端末から撮影できる範囲に近づかなければ測位ができないため、画像パターンの配置を工夫する必要がある。自然特徴点抽出による方式は、1 台のカメラを利用して、あらかじめ作成した環境の特徴点マップと撮影画像から抽出した特徴点とを照らし合わせて位置を推測する。この手法ではあらかじめ環境に機器などを設置する必要はないが、あらかじめ画像を集めて特徴点マップを作成しておく必要がある。また、模様替えなどで屋内の環境が変わることや、照明の条件、外光の条件などによって推測結果が影響を受けやすい。さらに気象条件の変動などの影響を受けることから、屋外での利用には向かない。

2.3.1.2. デッドレコニング型測位手法

2 種類目の方式は、歩行者がセンサーを保持して、そのセンサーの情報に基づいて測位を行うデッドレコニング型測位手法である。この手法では、加速度センサーとジャイロスコープを利用して歩行パターンを計測することによって、位置情報を推測する。加速度センサーとジャイロスコープを合わせて一般的に慣性センサーと呼ぶ。現在では、慣性センサー単体で測位する事は少なく、地磁気センサーや気圧センサーなどを組み合わせて測位をする手法が一般的である [38] [12]。

この手法は、インフラストラクチャー型測位手法と比較すると、あらかじめ環境側に対する機器設置やサーベイ等が不要であり、初めて立ち入るエリアでも測位が可能である。一方で、慣性センサーを利用してステップバイステップで位置を推測していくため、累積誤差が蓄積する。このため現在では、デッドレコニング型測位手法を単体で利用するのではなく、インフラストラクチャー型測位手法と組み合わせることによって、インフラストラクチャー型測位の利用できるエリアとそれ以外のエリアをシームレスにつなぐ目的や、デッドレコニング型測位手法のキャリブレーションのためにインフラストラクチャー型測位手法を利用するという方式が一般的である。

2.3.1.3. ハイブリッド型測位手法

インフラストラクチャー型測位手法とデッドレコニング型測位手法の相互の短所を補うために、両方の手法を組み合わせて測位をする手法が一般的である。例えば、ビーコン型の測位手法とデッドレコニング型測位手法を組み合わせた方式 [39]や、超音波とデッドレコニングを組み合わせた手法、GPS とデッドレコニングを組み合わせた手法 [40] [41]、GPS とデッドレコニングと RFID を組み合わせた手法 [42] などが挙げられる。

現在はデッドレコニング型測位手法を単体で利用されることは少なく、ハイブリッド型測位手法として異なる特徴を持つ複数の方式を組み合わせて測位が実現されている。さらには、これらのセンサー類はすべてスマートフォンに内蔵されるようになってきてるため、スマートフォン単体でハイブリッド型測位手法を実現している研究もある [43] [44]。しかし現在の技術では、屋外では GPS 等によって正確な位置が分かりつつあるが、屋内ではこのように複数の手法を組み合わせても、環境側にビーコン等を適切に設置しなければ、どの部屋にいるか、どの廊下を歩いているか等を正確にトラッキングし続けることはできない。

このため本研究で提案するプラットフォームには、位置だけではなく、位置を抽象化した場所という概念が扱えるべきである。また、上述のように様々な手法によって位置や場所が推測されることから、それぞれの手法の信頼度情報も含めて取り扱えるように考慮するべきであると言える。

第 3 章

ユビキタス ID アーキテクチャ

本章では、提案手法の基礎になっているユビキタス ID アーキテクチャについて解説する。このアーキテクチャは、トロンフォーラム、uID センターにて策定されている。

3.1. ucode

ucode は，識別したいあらゆる対象に割り当てることができる全世界でユニークな識別子である[1]。ucode は「モノ」や「場所」等の現実世界の実体を始め，これら実体を説明するための情報や概念などの非実体にも割り当てることができる。

ucode は以下の規格により、国際標準規格化されている。

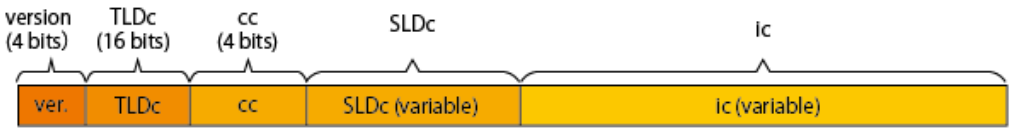
- ITU-T勧告 F.771："Service description and requirements for multimedia information access triggered by tag-based IDentification" (Oct. 2008)
- ITU-T勧告 H. 621："Architecture of a system for multimedia information access triggered by tag-based IDentification" (Oct. 2008)
- ITU-T勧告 H. 642.1："Multimedia information access triggered by tag-based IDentification - IDentification scheme" (June 2012)
- ITU-T勧告 H. 642.2："Multimedia information access triggered by tag-based

[1] http://www.uidcenter.org/ja/learning-about-ucode

IDentification - Registration procedures for IDentifiers" (June 2012)

- ITU-T勧告 H. 642.3："Information technology - Automatic IDentification and data capture technique - IDentifier resolution protocol for multimedia information access triggered by tag-based IDentification" (June 2012)
- IETF RFC6588: "A URN Namespace for ucode" (April 2012).

ID は 128 ビット固定長であり、領域が不足した場合には 128 ビットずつ拡張ができる仕様となっている。図 3.1 は、ucode の構造を表す。



フィールド名	意味
version	バージョン
TLDc: Top Level Domain Code	上位レベルのドメイン識別番号
cc: Class Code	SLDc と ic の境界を示すコード
SLDc: Second Level Domain Code	下位レベルのドメイン識別番号
ic: Identification Code	個体識別番号

図 3.1 ucode の構造（ucode 解説記事[2]より引用）

ucode のユニーク性は、以下のような方法で担保されている。

- ucodeを管理する非営利団体・トロンフォーラムは、その運用・管理方針を規定している。この規定に基づいて、ucodeの割り当てを受ける会員と、唯一性を担保するための契約を締結することで、その唯一性が保証される。
- ucodeの発行対象が消滅したとき、ucodeも破棄される。後から同じucodeを再利用することもなく、破棄されたucodeは欠番となる。従って、ucodeは空

[2] http://www.uidcenter.org/ja/learning-about-ucode

間方向への唯一性だけでなく、時間軸方向への唯一性も保障する。

また、ucode の付与対象は、現実世界のさまざまな「モノ」や「場所」、「コンテンツ」、「概念」などに付与できるため、他の ID 体系のように、特定のモノにしか付与できない、というようなことはないため、IoT デバイスやその周辺環境の状態、接続関係、概念など、様々な情報に ID を付与したり、QR コードとして利用することや IC カード内に保存したり、RFID の発する電波内に埋め込むような、多様な方法によってコンピューター上で扱うことができ、IoT を実現する上で十分な能力を備えていると言える。

また、採用実績も多い。ucode は以下のような様々なサービスで採用されている

- 東京ユビキタス計画
- 銀座ガイドシステム（東京都）
- ユビナビ（津和野町、沖縄県読谷村、秋田県、岩手県、千葉県柏の葉）
- 美術館ユビキタス案内システム（青森県立美術館）
- 世界遺産熊野古道ナビ・プロジェクト（那智勝浦町）
- ふるさと観光ユビキタス「e-地域資源活用事業」 （ふるさと財団）
- ココシル（ユーシーテクノロジ株式会社）
- 住宅備品のトレーサビリティ（一般財団法人ベターリビング）
- インテリジェント基準点（国土地理院）
- 食品トレーサビリティ

特に以下のプロジェクトでは、複数のサービス間での連携に関する実績があり、IoT のように複数のコンピューターシステムが連携して動作するような環境でも採用された実績がある。

- 住宅備品のトレーサビリティ（一般財団法人ベターリビング）
- 食品トレーサビリティ

3.2. ユビキタス ID アーキテクチャ

uID アーキテクチャは、コンテキストアウェアネスを実現するために ucode をベースにして実現されたアーキテクチャである。コンテキストアウェアネスを実現するためには、モノや事象や概念などを一意に識別できる必要があるため、これらに ID として ucode を付与して識別をする。さらに ucode を付与した対象と対象との関係を、さらに ucode で記述する（この情報の表現形式をトリプルと呼ぶ)。図 3.2 は、トリプルの図示である。Subject、Object、Predicate それぞれに対して一つずつ ucode が割り当てられ、これらを predicate で接続していくことで、情報間の関連性を記述していく。このように ucode を付与した対象同士を ucode で関連性を記述していくための仕組みを ucR と呼ぶ。図 3.3 と図 3.4 は、この ucR を実際のデータ記述に応用した例である。なお Subject, Object, Predicate には可読性を保つために URL による記述を行っているが、実際にはこの URL に対応する ucode が定義されており、システムの内部では ucode による関連データベースが構築されていて、これを SQL ライクなクエリ言語の SPARQL 等で検索可能になっている。

ユビキタス ID アーキテクチャでは、このような ucR に基づく情報の表現と、検索を可能としている。

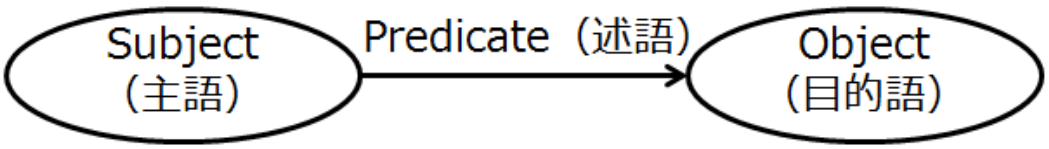


図 3.2 ucR を構成する基本単位 トリプル

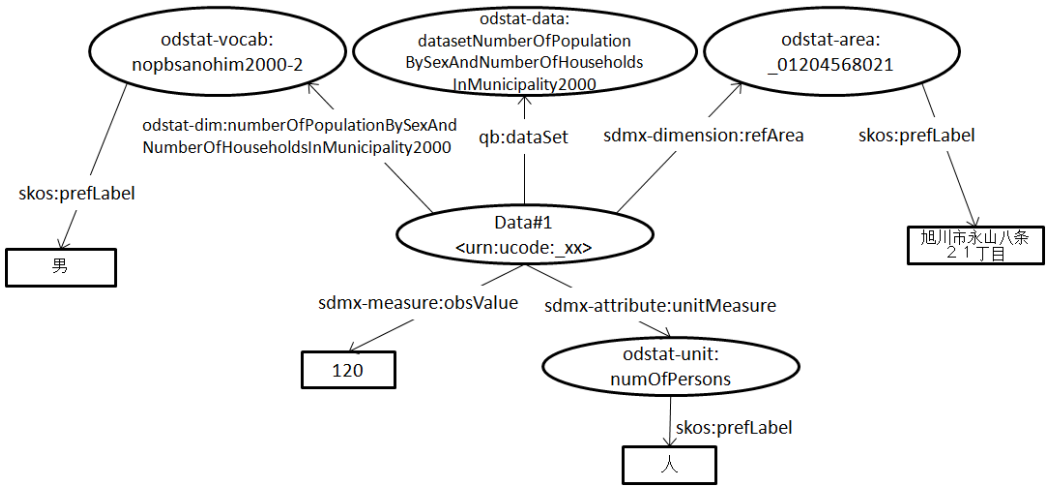


図 3.3 国勢調査データを ucR で記述した例

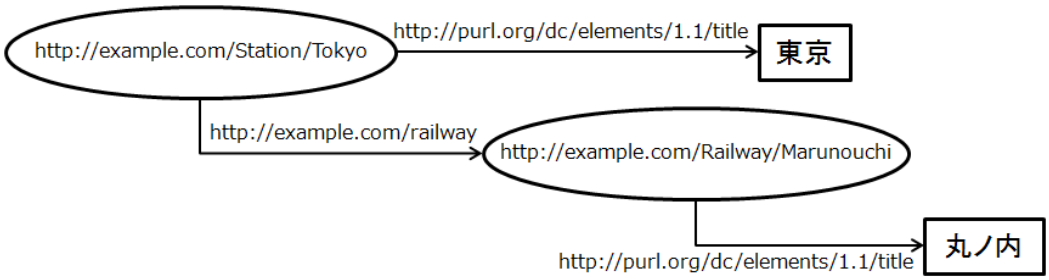


図 3.4 鉄道オープンデータを ucR で記述した例

3.3. u2 アーキテクチャ

先述のユビキタス ID アーキテクチャをさらに拡張したのが u2 アーキテクチャである。この u2 アーキテクチャでは、従来の柔軟性に加え、共通部分を定義しておくことで、複数のサービス間での相互接続性を高めることを目的としている。図 3.5 は、u2 アーキテクチャの全体像である。この図で表現されているように、人や物やサービスの認証を行うのは daresil、モノの流通や所有権限の移転などを管理する monosil、場所情報サービスを kokosil、様々な現実世界でのイベントなどを管理する kotosil というような、サービスで一般的に利用されるであろう共通部分をそれぞれ抽出することで、さらにユビキタス ID アーキテクチャの利便性を向上させることを目的としている。

さらに、データベースのクロスクエリ化にも取り組んでいる。図 3.6 は、データベースの実体がどのような形式のクエリを受け付けるモノであっても、この u2 アーキテクチャが差異を吸収することによってデータベースに統一的なクエリでアクセスできるような手法が提案されている。

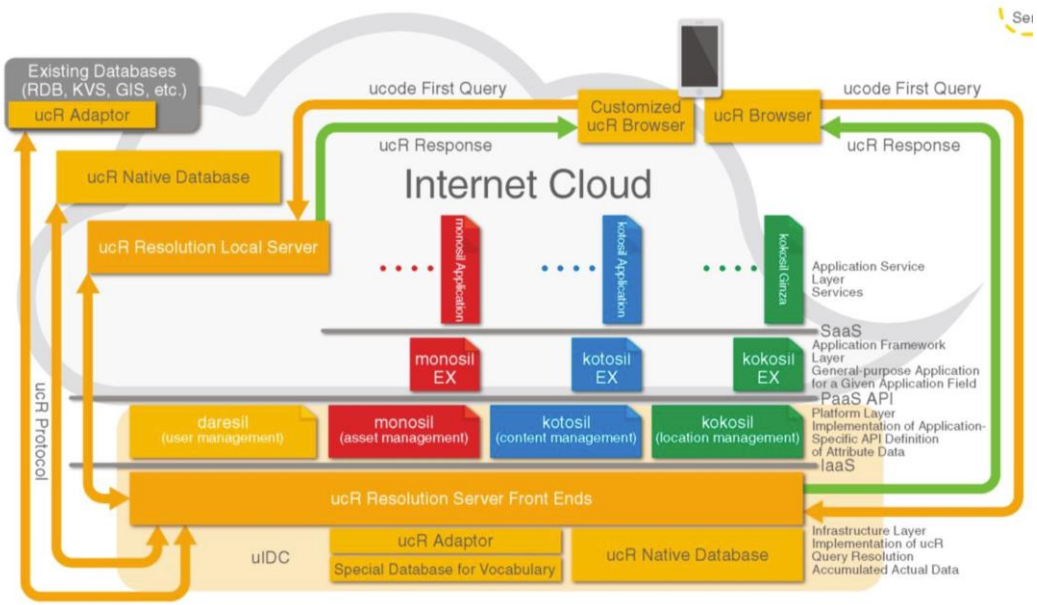


図 3.5 u2 アーキテクチャ全体像

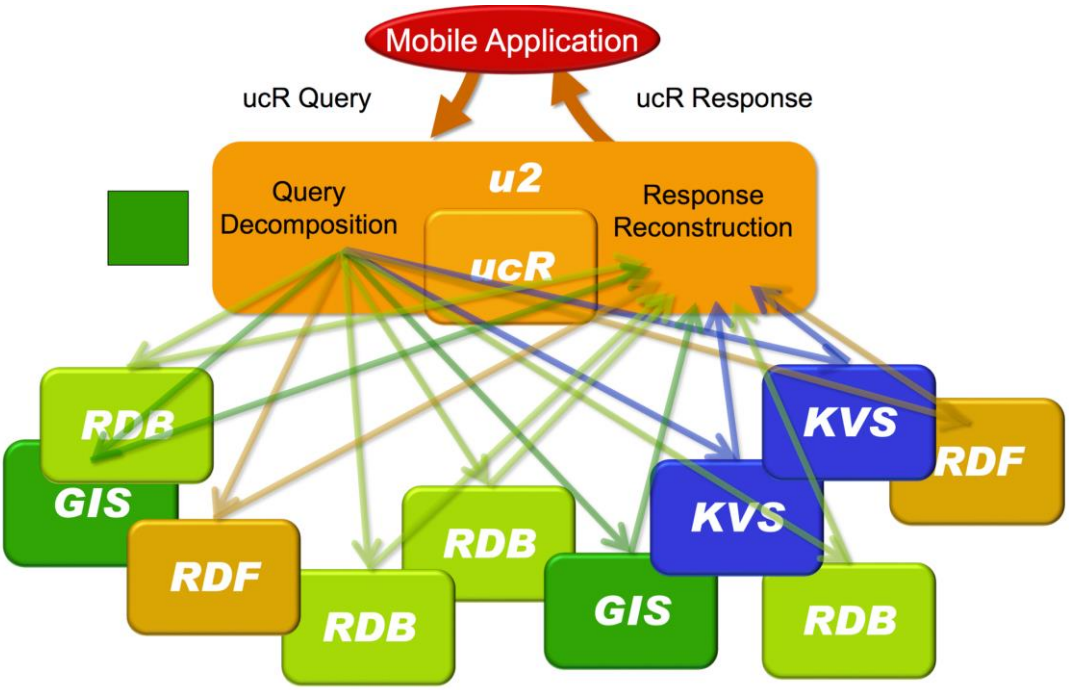


図 3.6 u2 アーキテクチャによる複数データベースサービスの統合
（TRONSHOW2014 の発表資料[3]より引用）

[3] http://www.t-engine.org/wp-content/themes/wp.vicuna/pdf/20131211u2-J3.pdf

3.4. IoT-Aggregator

IoT-Aggregator は、TRON プロジェクトが提案する総体モデルを実現するためのモデルアーキテクチャである。図 3.7 は、総体モデルを利用して実現する IoT を説明した図である。総体モデルでは、様々なモノに組み込まれるようなエッジノードは軽量なシステムとして単純な機能を実現し、クラウド上の計算資源の豊富な環境でアクセスコントロールやセンサーインテグレーション、コンテキストアウェアネス等を実現して、計算結果をエッジノードに返すことで、デバイス単体ではなく、いたるところにあるサービスを総体として組み合わせることで、ユーザーにとって利便性の高いサービスを提供するための仕組みである。

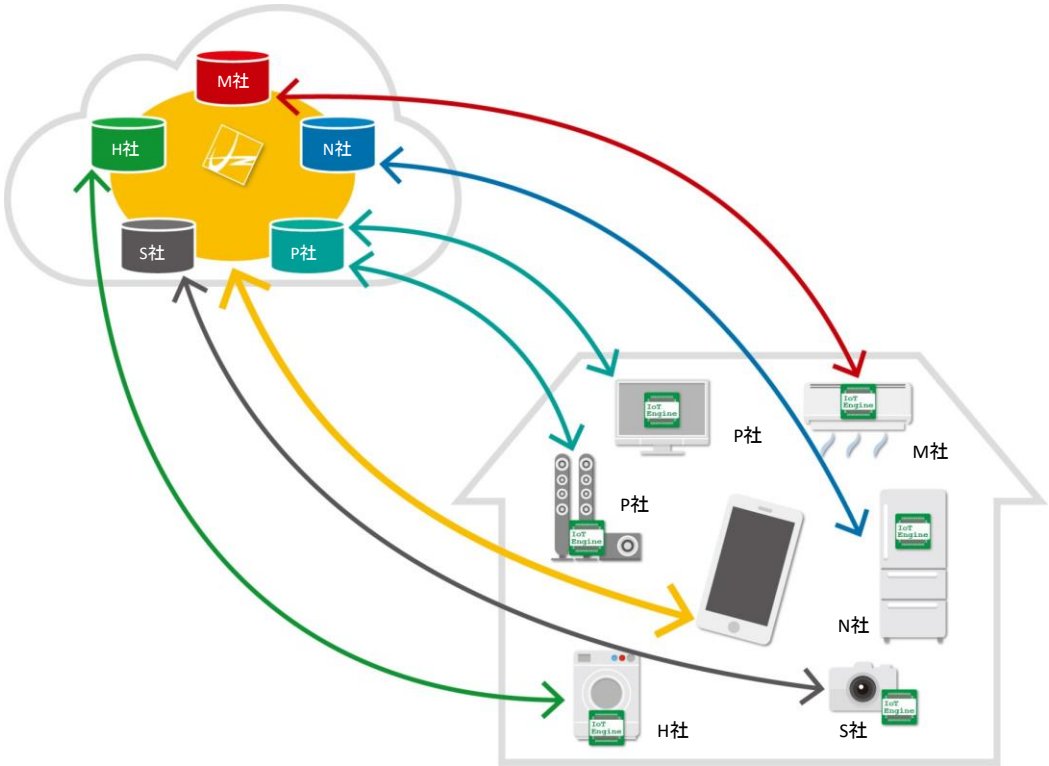


図 3.7 総体モデルに基づく IoT （IoT-Aggregator 発表資料[4]より引用）

[4] http://www.tron.org/ja/2016/04/post-1770/

この総体モデルに基づいて提案されるモデルアーキテクチャが、図 3.8 に示す IoT-Aggregator である。

ucode マネージャは、IoT-Aggregator で利用する ucode を管理する。デバイスマネージャは、IoT-Aggregator に接続するデバイスの情報を登録し、さらに設置場所や所有者などの情報を管理する。さらにデバイスによって異なる API を標準化してアクセスするための仕組みも提供する。デバイスプロファイルリポジトリは、IoT-Aggregator に登録されるデバイスの仕様や API などの情報が記載されたプロファイルを登録するための仕組みを提供する。uID Accounts は、IoT-Aggregator を利用するすべての人とデバイス、サービスに関する認証のための情報を管理する。TRON IoT Dashboard は、デバイスマネージャをエンドユーザーがコントロールするための Web 等によるインタフェースを提供する。

IoT-Aggregator はこのような概念を表すアーキテクチャであり、本研究の成果も取り込んで概念を構築し、さらに本研究ではこの概念に基づいて、特にデバイスマネージャとデバイスプロファイルリポジトリについて実装も行っている。

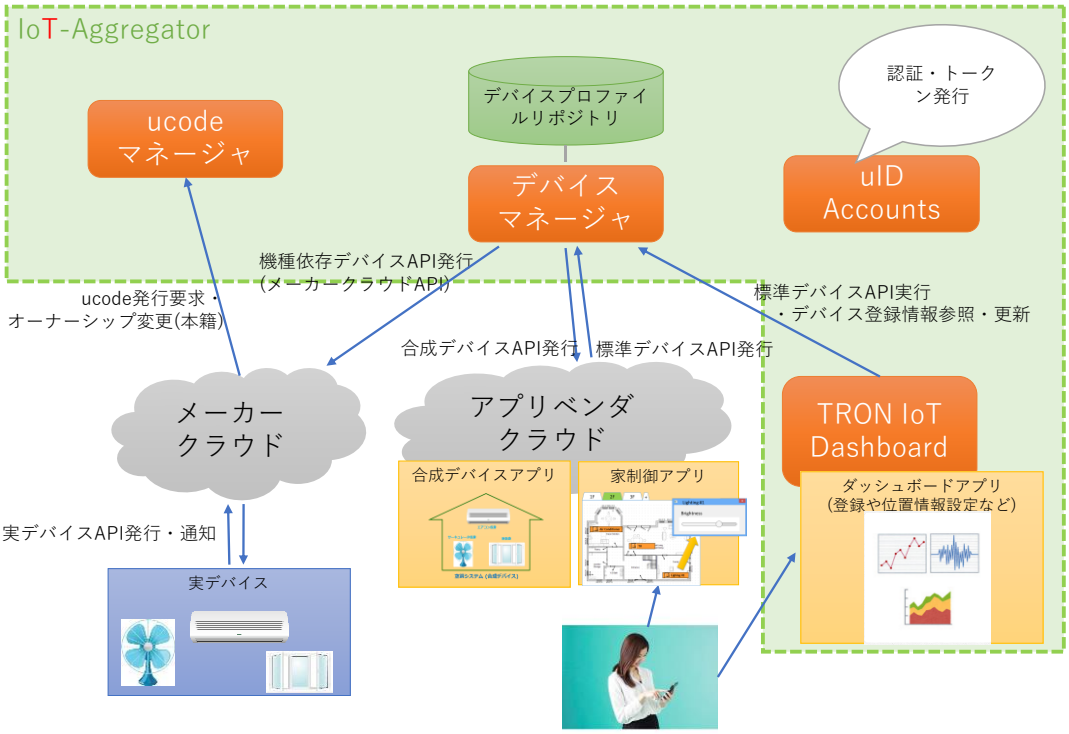


図 3.8 IoT-Aggregator の全体像

この IoT-Aggregator では、総体モデルに基づいているため、例えばアクセスコントロールのような複雑な処理についてはクラウドで実現を行っている。図 3.9 は、この IoT-Aggregator を利用してアクセスコントロールを実現している例である。

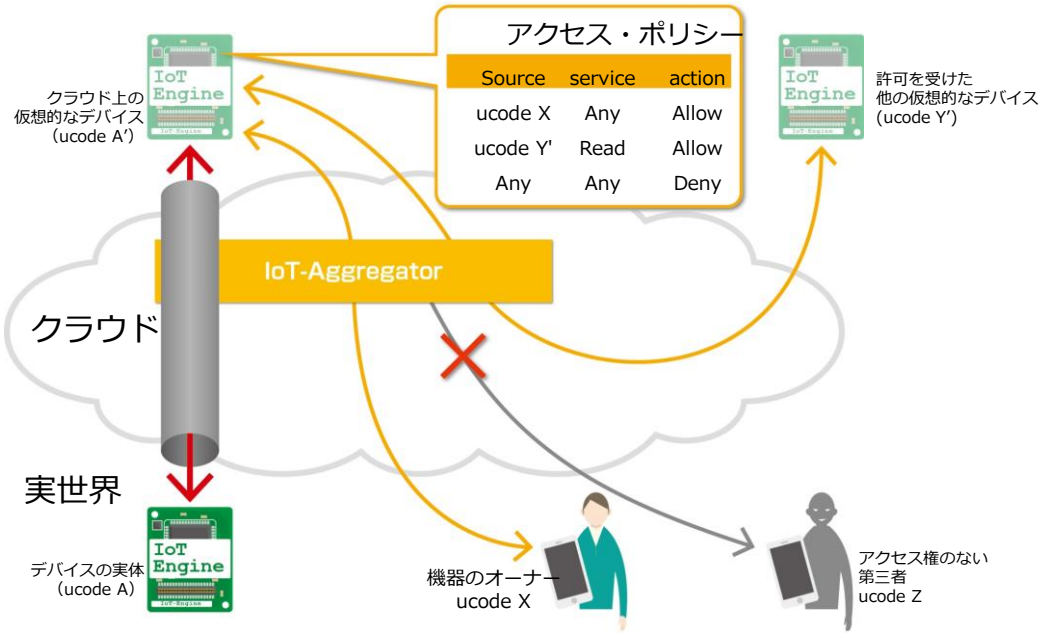


図 3.9 アクセスコントロールはクラウド上で実現する

3.5. OPaaS.io

OPaaS.io（Omotenashi Platform as a Service. Integrated and open）は、IoT-Aggregator のモデルに基づいて提案された個人の属性情報を管理するためのプラットフォームである。本研究では、個人の属性情報をこの OPaaS.io のようなプラットフォームから取得し、その情報と IoT デバイスやサービスとを組み合わせでサービスを実現することを想定している。また、ユーザーの位置情報なども、この OPaaS.io のような仕組みの中で管理されていることを想定している。

本プラットフォームは、図 3.10 に示すように、ユーザー属性情報を蓄積し、エンドユーザーの管理の下でサービスベンダーに適切に渡せるように「仲介」を行うオープンなプラットフォームを実現する。この概念は、顧客の情報をサービスベンダーの管理の下で蓄積・管理していた CRM（Customer Relationship Management）に対して、顧客の管理の下でサービスベンダーに渡す情報をコントロールできることから、VRM（Vendor Relationship Management）と呼ばれている。この VRM の概念を図 3.11 に示す。この VRM を実現する上で重要な技術が、ユーザー属性情報をセキュアに一元管理するサービスである PDS（Personal Data Store）である。PDS ではユーザー属性情報をサービスが取得するための API を提供し、ユーザーが選択したもののみをサービスベンダーに共有することができる。

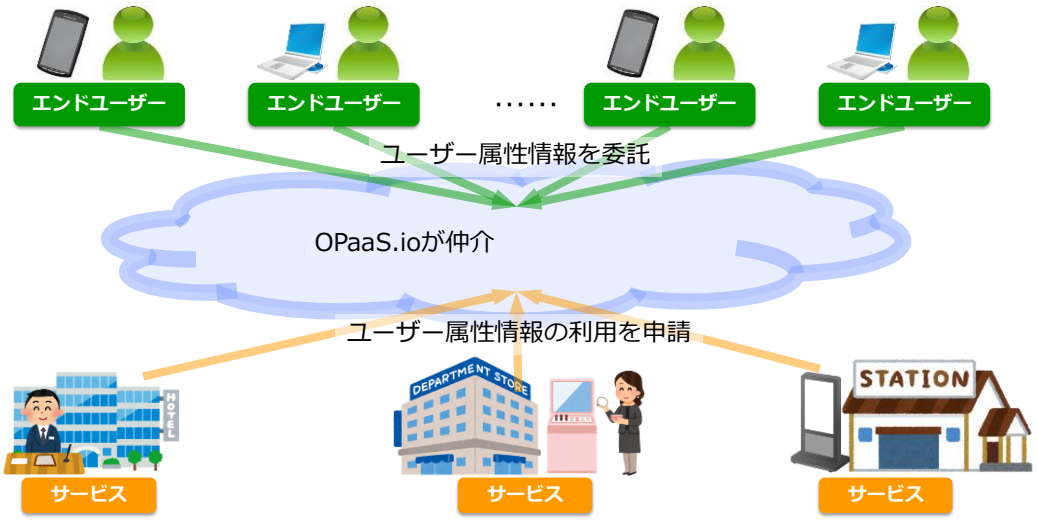


図 3.10 OPaaS.io の位置づけ

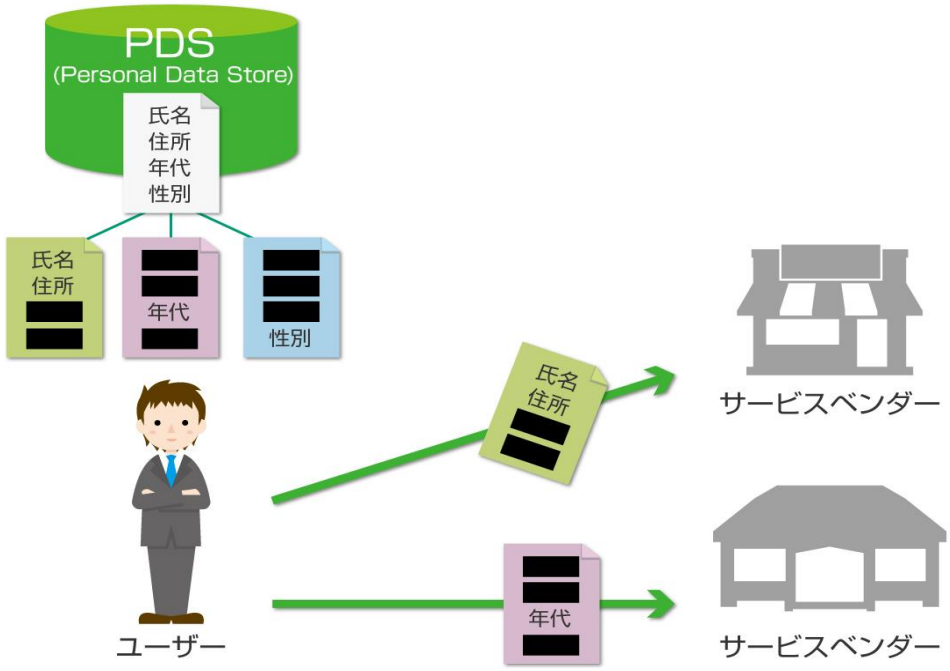


図 3.11 属性情報の提供先（ベンダー）はエンドユーザーがコントロールする

OPaaS.io で扱うユーザー属性情報は、例えば、名前、使用言語、食の禁忌等であるが、どこまで登録するかはユーザーの自由である。OPaaS.io のエンドユーザーは海外からの旅行客や日本国民など、幅広い方を対象としている。対象とするサービスとして、例えば、ホテル、ショップ、レストラン、博物館・美術館、交通機関など、インターネットでなく実世界のサービスを主眼にしている。また、プラットフォームの定める規約に従って登録すれば、誰でも、いつでも、何にでも使えるオープン性を持つ。

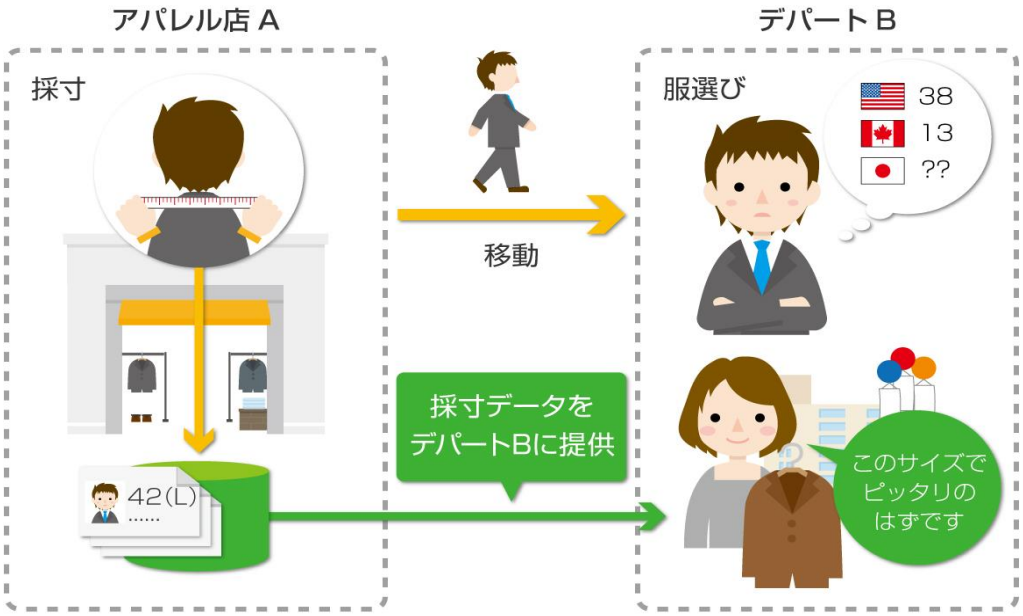


図 3.12 OPaaS.io に属性情報を登録すればサービス間で情報を共有できる

OPaaS.io では、ユーザーに 1 人 1 個の ucode を割り当てて、その ucode に対して個人の属性情報を紐付ける。この ucode をオリジナル ID と呼ぶ。しかし、このオリジナル ID を他のサービスに使い回すと、あるサービスから情報流出した場合に、その情報に含まれる ID をキーにして他のサービスの情報も抽出できてしまうというセキュリティ上の懸念があるため、OPaaS.io ではリンク ID という仕組みを導入している。ユーザーの ID をサービスに提供する場合には、オリジナル ID をそのまま提供するのではなく、オリジナル ID と OPaaS.io 内で紐付けられたサービス毎に異なる ID を発行して、この ID を提供する。この ID のことをリンク ID と呼ぶ。リン

ク ID のイメージを図 3.13 に示す。このような仕組みによって、登録されたユーザーのプライバシーを担保する。

図 3.13 オリジナル ID とリンク ID

OPaaS.io は図 3.14 に示すように構成され、それぞれ以下のような機能を実現する。

- ユーザー属性情報管理モジュール
  - OPaaS.io内や外部サービスのユーザー属性情報を検索・取得
- アイデンティティ管理モジュール
  - ユーザーの認証、アクセス制御、およびAPIの利用状況などの管理を実現
- インフラ管理モジュール
  - OPaaS.ioのインフラ自身の監視など、OPaaS.ioのサービス運用のために必要となる機能を実現
- おもてなしポータル

- OPaaS.ioに情報を登録するために利用するためのポータルを実現
- Webアプリケーションとスマートフォンアプリケーション等を提供

- サイネージレジストリ
  - OPaaS.ioを活用したサイネージに関する情報を保持

また、OPaaS.io は u2 アーキテクチャに基づいたプラットフォームであるため、図中にも uID センターを記載している。

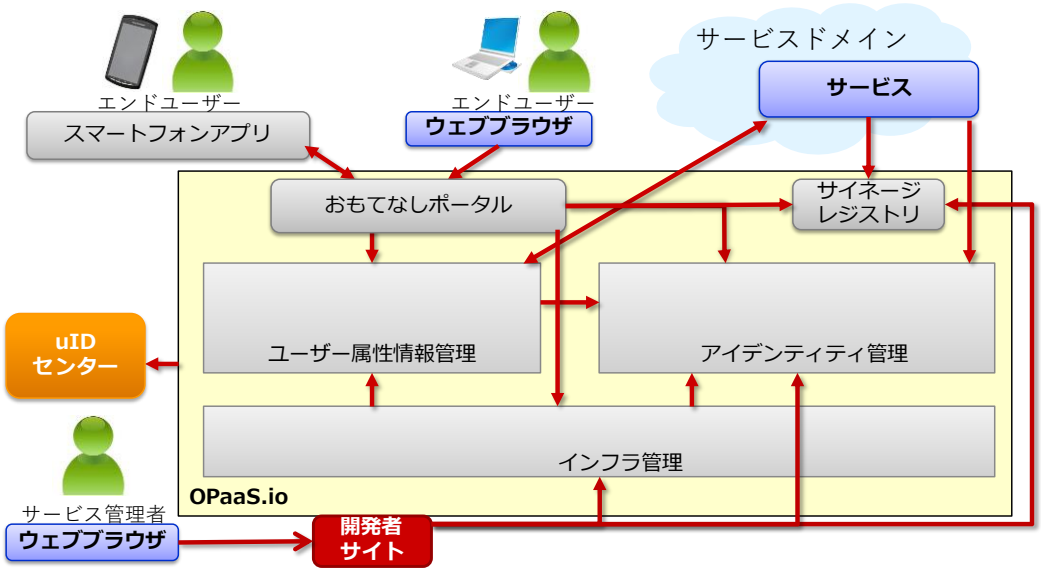


図 3.14 OPaaS.io の全体アーキテクチャ

図 3.15 は、OPaaS.io の詳細なアーキテクチャである。以下、それぞれについて詳細に説明する。

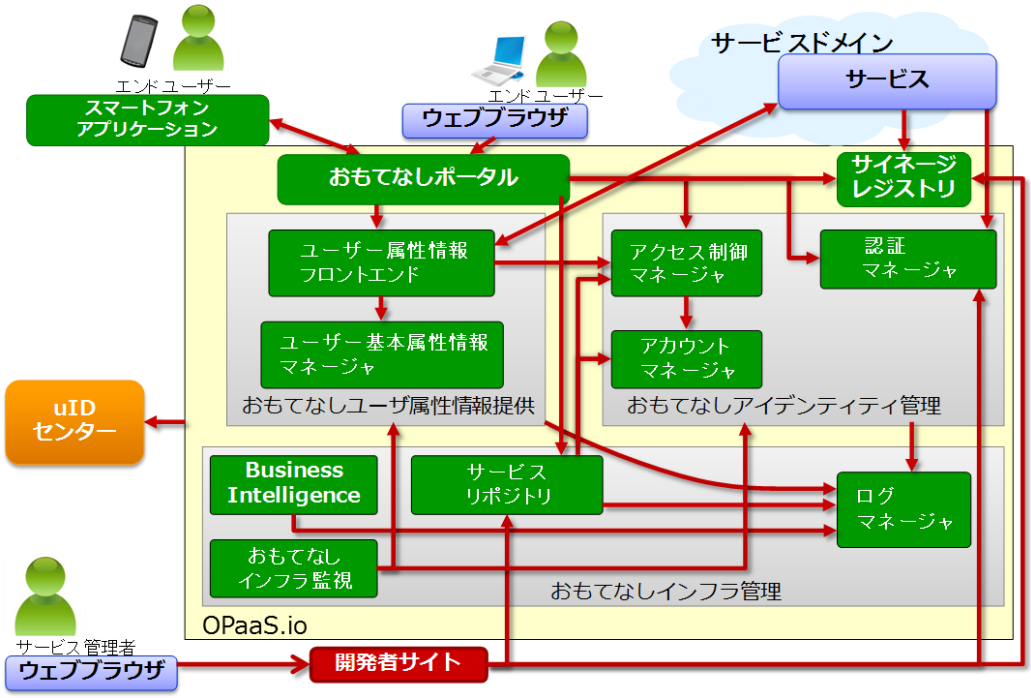


図 3.15 OPaaS.io の詳細アーキテクチャ

3.5.1. ユーザー属性情報管理

ユーザー属性情報管理は、OPaaS.io 内部や外部で管理しているユーザー属性情報をサービスから受け付けたリクエストとユーザーの設定したアクセス権限に基づいて検索・提供する。以下のモジュールにより構成する。

- ユーザー属性情報フロントエンド
  - おもてなしポータルやサービスからユーザー属性情報の取得要求を受け付け、アクセス権限に基づいて、ユーザー属性情報の取得を仲介する
    - サービスへのレスポンス時に、ユーザー属性情報の単位変換、表記変換にも対応する
  - リンク ID をオリジナル ID との対応付けを保持する
- ユーザー基本属性情報マネージャ
  - ユーザー基本属性情報を管理し、サービスの要求に応じて提供する
    - ユーザー基本属性情報とは、OPaaS.io としてあらかじめ定義してい

るユーザー属性情報であり、例えば氏名や性別、母国語などである。

3.5.2. アイデンティティ管理

OPaaS.io にアクセスするユーザーの認証と、サービスの認証、およびアクセス制御を行う。

- アクセス制御マネージャ
  - アクセス権限の管理を行う
  - ユーザー属性情報をサービスに対して提供できる権限があるか確認を行う
- 認証マネージャ
  - ユーザーの認証とサービスの認証を行う
  - 交通系 IC カード（FeliCa）や、OpenID Connect 1.0 に基づくユーザー名・パスワード等の様々な認証方式に対応する
  - 認証要求時には、必要とする信頼度を指定できることとし、結果には認証方式の信頼度も合わせて返却する必要がある
  - ユーザーを識別する ID として内部識別用にはオリジナル ID を利用し、サービスに提供する際にはリンク ID を提供する
- アカウントマネージャ
  - サービス事業者の課金の状況を管理する
    - サービス事業者の登録を行う際や、API アクセス数に基づいて課金をできることを想定する

3.5.3. インフラ管理

OPaaS.io 自身の死活監視や、OPaaS.io の運用上必要となる情報を管理する。

- サービスリポジトリ
  - サービス事業者に関する情報と、サービス事業者が開発したサービスに関する情報を管理する
  - サービス事業者、サービスの登録時には、それぞれに対応する ID を発行する

- ✧ サービス事業者に付与する ID はサービス事業者 ID と呼ぶ
- ✧ サービス登録時に付与する ID はサービス ID と呼ぶ

- ログマネージャ
  - OPaaS.io 内のモジュールに関する全てのログを一元管理し、ログの取得、および検索できる API を提供する
- Business intelligence
  - ログマネージャが管理しているログを活用して、どのユーザー属性情報が頻繁に利用されるか等、OPaaS.io の利用状況や、応答時間等の性能を解析する
- インフラ監視
  - OPaaS.io の死活監視を行い、モジュールに異常が発生した場合には、該当モジュールの再起動や、管理者への通知などを行う

3.5.4. その他

- サイネージレジストリ
  - OPaaS.io に接続するサイネージの情報を登録する
  - サイネージの認証に利用する、証明書やアクセストークン等の情報も管理する
- 開発者サイト
  - OPaaS.io を利用する開発者向けに、API の説明やサンプルコードの提供、サービスから OPaaS.io へ接続するための認証情報の提供など、OPaaS.io と連携したサービスを開発するために必要な機能を提供する

第 4 章　　提案アーキテクチャ　Huot

本章では、Human-centric アプリケーションと IoT を統合するための提案アーキテクチャである Huot の概要について説明した後、実現の方針について説明する。

4.1. 概要

Human-centric アプリケーションと IoT を統合するために、前章で説明したユビキタス ID アーキテクチャに基づく新たなアーキテクチャ Huot を提案した。Huot の全体像を図 4.1 に示す。

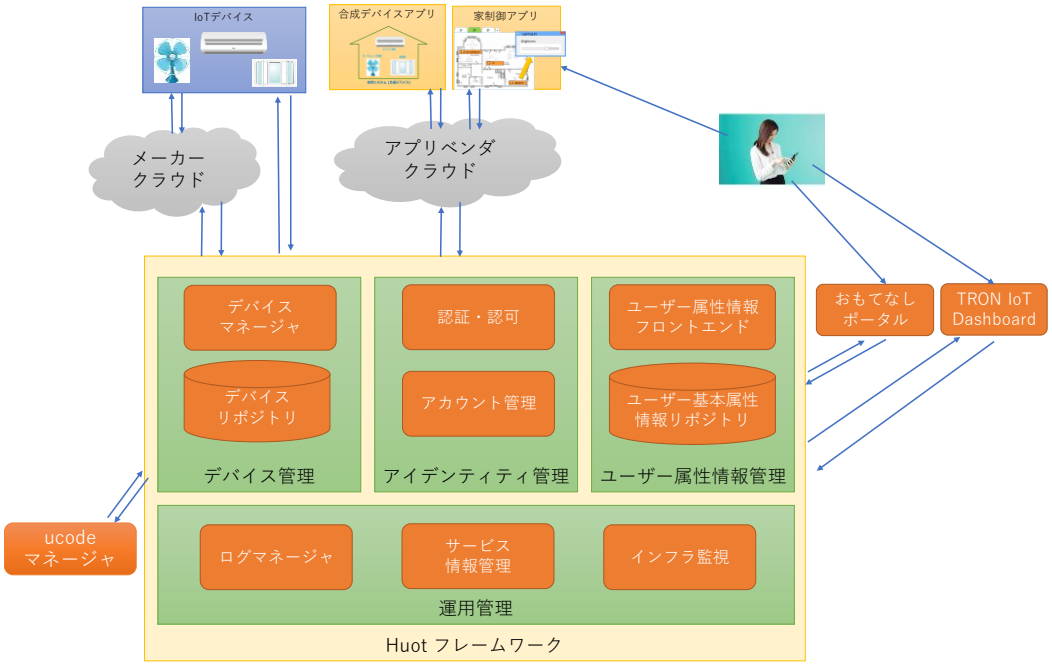


図 4.1 提案アーキテクチャ Huot の全体像

提案アーキテクチャ Huot が関連する外部のシステムは、以下が挙げられる。

- IoT デバイス
  - IoT デバイス自身が Huot の提供する API を利用してアクセスする場合を想定している
- メーカークラウド
  - IoT デバイスを開発したメーカーが提供するクラウドサービスである
  - IoT デバイスの提供している API の仕様や IoT デバイスの開発元の方針などにより、IoT デバイス自身が Huot と接続できない場合を想定している
- アプリベンダクラウド
  - Huot に登録された IoT デバイスを活用したアプリケーションを提供する事業者のクラウドサービスである
  - Huot の API を利用して IoT デバイスを制御するだけではなく、Huot に対して、ソフトウェアで実現された独自の IoT デバイスを登録することも可能である

- ✧ 例えば、複数の IoT デバイスを複合して 1 個の IoT デバイスとして見せることや、部屋をまとめて 1 個の IoT デバイスとして API を提供することなどを想定する

- ucode マネージャ
  - ucode を管理するサーバー
  - Huot に登録する IoT デバイスやユーザー、サービス等にははすべて ucode を付与して管理を行う
- おもてなしポータル
  - Huot に登録したユーザー属性情報を管理するためのポータルサービス
  - ユーザー属性情報の登録・編集や、どのサービスにどの情報を提供するか等をユーザーが設定するためのサービスである
- TRON IoT Dashboard
  - Huot に登録した IoT デバイスの管理を行うためのダッシュボードサービス
  - IoT デバイスの登録・編集や、所有権の設定、簡易なプログラミング等を実現するためのサービスである

提案アーキテクチャ Huot は、大きく分けて以下の 4 種類のモジュールから構成している。

- デバイス管理
  - Huot に登録した IoT デバイスの管理を行う
- ユーザー属性情報管理
  - Huot に登録したユーザー属性情報の管理を行う
- アイデンティティ管理
  - Huot を利用するユーザーアカウントの管理や、Huot に接続するサービスの接続情報を管理する
- 運用管理
  - Huot 自身を安定して運用するために、ログ集約やシステム監視などを実現する

以降では、それぞれのモジュールについて説明を行う。

4.1.1. デバイス管理モジュール

デバイス管理モジュールでは、Huot で扱う IoT デバイスや、メーカークラウド、アプリベンダクラウドの管理を実現するために、以下のような機能を提供する。

- IoT デバイスの種別の管理
  - IoT デバイスの提供する機能等に基づく種別のリストを管理する
  - IoT デバイスの種別でグルーピングをすることで、IoT デバイスの開発者は IoT デバイスの利用時に、個々の IoT デバイスがどのメーカーで作成されたどのモデルのデバイスか意識する必要がなくなる
- IoT デバイスが提供する API の管理
  - IoT デバイスが提供している API のアクセス先やアクセス方法、パラメーターなどを、機械可読な形式で管理する
- メーカークラウドやアプリベンダクラウドが提供する API の管理
  - メーカークラウドが Huot に対して提供する API も、アプリベンダクラウドが Huot に対して提供する API も、いずれも Huot からは IoT デバイスと透過的に扱えるべきであることから、IoT デバイスと同様に管理する

以上のような機能を実現するために、デバイス管理モジュールは以下のようなサブモジュールから構成する。

- デバイスマネージャ
  - Huot に登録された IoT デバイス・メーカークラウド・アプリベンダクラウドを検索・追加・削除・更新するための API を提供する
  - Huot に登録された IoT デバイス・メーカークラウド・アプリベンダクラウドの API を、別の IoT デバイス・メーカークラウド・アプリベンダクラウド等からアクセスするための API を提供する
- デバイスリポジトリ
  - デバイスマネージャを実現するために、IoT デバイスに関する情報を保管する機能を提供する

4.1.2. ユーザー属性情報管理モジュール

ユーザー属性情報管理モジュールでは、Huot で扱うユーザー属性情報の管理を行う。具体的には、以下のような機能を実現する。

- ユーザー属性情報の一覧の管理
  - ユーザー属性情報として Huot が扱う必要のある情報のリストを管理する
  - 将来的に、Huot 内部で管理するユーザー属性情報だけではなく、アプリベンダクラウド等が管理するユーザー属性情報も Huot が仲介してアクセスできることも考慮する
- ユーザー属性情報の保管場所の管理
  - 将来的に Huot の外部で管理するユーザー属性情報も扱えるようにするために、アプリベンダクラウドがどのようなユーザー属性情報を管理・提供可能であるかを管理する
- ユーザー属性情報の表現形式の変換
  - Huot 内部に格納されたユーザー属性情報が、Huot 外部の必要とするユーザー属性情報の表現形式と異なる可能性があることから、ユーザー属性情報の表現形式を簡易的なスクリプト等により変換可能とする
- ユーザー属性情報の管理
  - 氏名やサービスを受ける際の希望言語などを表すユーザー属性情報を追加・更新・削除する

以上のような機能を実現するために、ユーザー属性情報管理モジュールは以下のようなサブモジュールから構成する。

- ユーザー属性情報フロントエンド
  - ユーザー属性情報を取得・更新・削除するための API を提供する
    - ユーザー属性情報の表現形式を Huot 内で変換してから取得をすることも可能とする
    - 将来的にはアプリベンダクラウド等、Huot 外部で管理されているユーザー属性情報も、ユーザー属性情報フロントエンドが仲介して提供する

- ユーザー属性情報の一覧を取得・更新・削除するための API を提供する
  - ユーザー属性情報の保管場所なども合わせて管理する
- ユーザー属性情報の表現形式の変換方法を管理する

- ユーザー基本属性情報リポジトリ
  - ユーザー属性情報フロントエンドから要求を受けて、Huot 内部で保管するユーザー属性情報を管理する

4.1.3. アイデンティティ管理モジュール

IoT デバイスやユーザー等を認証し、アクセス制御を行うために、以下の機能を提供する。

- 認証方式の管理
  - ユーザー名・パスワードによる認証や、アクセストークンによる認証等、様々な認証方式により、アクセス元がどのユーザー、サービスであるかを識別する
  - 将来的に、ユーザーの認証方式として多要素認証や生体認証等に対応できるような設計とする
- アクセス権限の管理
  - 認証した結果判明したアクセス元が、Huot 内で管理されている IoT デバイスやユーザー属性情報に対してアクセス可能かどうかを管理する
- 課金の管理
  - 将来的に、どのサービスが Huot 内のどの機能をどれだけ利用したかに基づいて、利用料を請求可能にするための仕組みを提供する

以上のような機能を実現するために、アイデンティティ管理モジュールは以下のサブモジュールから構成する。

- 認証・認可
  - 認証方式の管理と、アクセス権限の管理（認可）を実現する
- アカウント管理
  - 課金の管理を実現する

4.1.4. 運用管理モジュール

運用管理モジュールでは、Huot を安定的に運用するために以下の機能を提供する。

- ログの集約管理
  - システムに対するアクセスログやモジュールの動作ログなどを一元で管理する
- Huot 内部のモジュール死活監視
  - Huot がサービス提供を継続するために、Huot 内部の各モジュールが正常に動作していることを確認する
- サービス情報管理
  - Huot に接続する外部サービスであるメーカークラウドやアプリベンダクラウド、および Huot を利用するサービスなどの名前や説明、利用するための URL などを管理する
  - TRON IoT Dashboard やおもてなしポータルで、利用者向けに表示するための情報として利用する

以上のような機能を提供するために、運用管理モジュールは以下のモジュールから構成する。

- ログマネージャ
  - ログを一元管理する機能を提供する
    - Fluentd 等を想定する
- インフラ監視
  - Huot の内部のモジュールを死活監視する
    - Zabbix 等を想定する
- サービス情報管理
  - サービスの情報を管理する

4.2. 実現の方針

これまでに説明した機能を実現するために、本研究ではユーザー属性情報を扱うシステムと、IoT デバイスを扱うシステムの、2 種類のシステムを構築する方針とした。

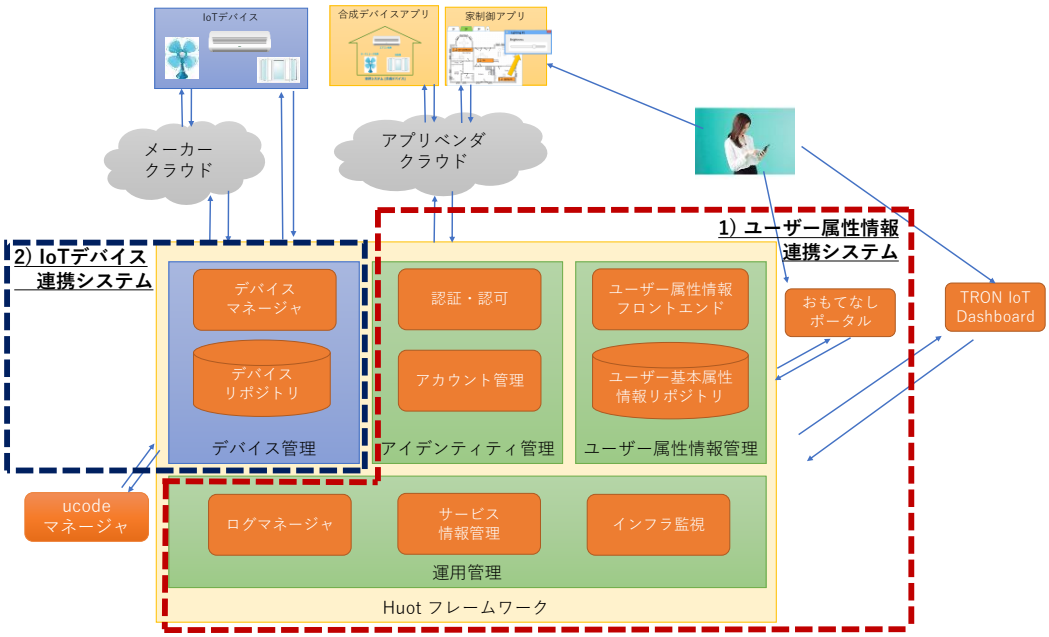


図 4.2 Huot アーキテクチャの実現方針

- ユーザー属性情報連携システム
  - ユーザー属性情報を様々なアプリケーション等で連携するためのプラットフォームとして、Huot のうち OPaaS.io に相当する箇所を構築する
- IoT デバイス連携システム
  - IoT-Aggregator に基づいて、Huot のうち IoT デバイスを連携するためのシステムを構築する
  - このシステムを Colapi と名付ける

第 5 章

ユーザー属性情報連携システム

本章では、提案アーキテクチャの Huot を実現するための構成要素のうち、ユーザー属性情報連携システムに関して説明を行う。また、ユーザー属性情報の例として、位置情報について説明を行う。

5.1. ユーザー属性情報連携システムの概要

ユーザー属性情報連携システムはすでに第 4 章で説明したように、大きく分けて以下の 3 種類のモジュールから構成する。なおそれぞれのモジュールの機能はすでに説明しているため、省略する。

- ユーザー属性情報管理
- アイデンティティ管理
- インフラ管理

ここではさらに、ユーザー属性情報を活用したアプリケーション（サービス）を実現するために、ユーザー属性情報を利用して構築されたサイネージの情報を保管するサイネージレジストリと、ユーザー属性情報連携システムの提供する API 等の仕様や利用方法を説明するための開発者サイトを構築した。

以上のような、ユーザー属性情報連携システムの全体像を図 5.1 に示す。

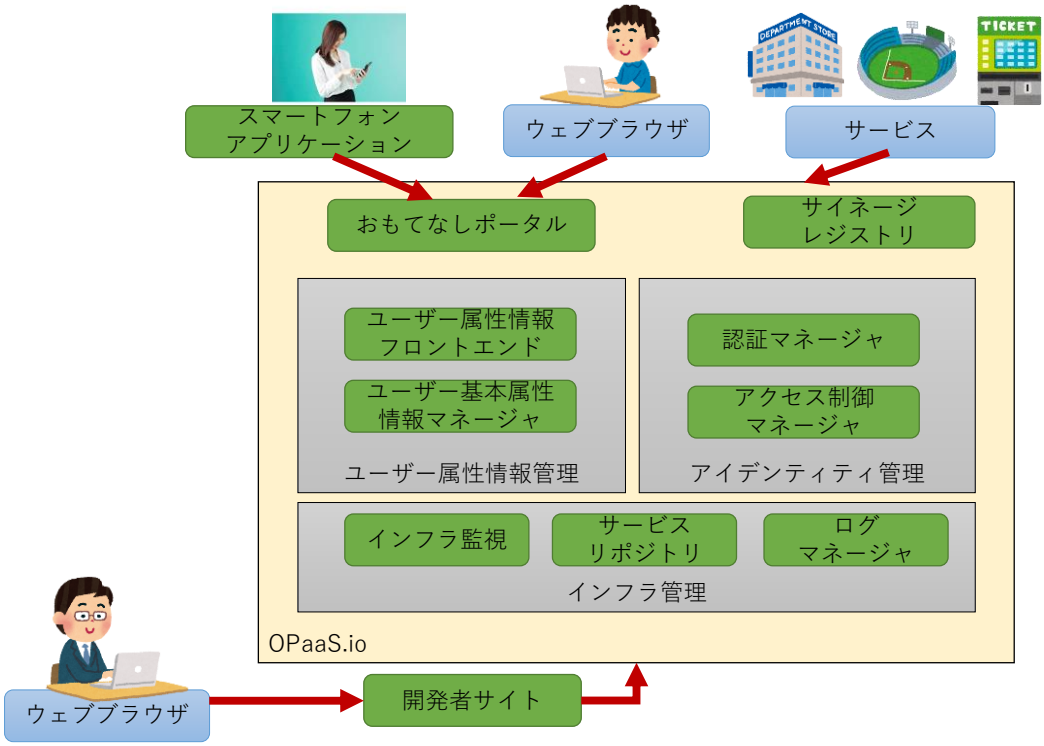


図 5.1 ユーザー属性情報連携システムの全体構成

ユーザー属性情報連携システムの新規性は、サービスからのリクエストに応じて、ユーザー属性情報の表記をスクリプトによって任意の表現形式に変換可能であることである。

5.2. 実現方法

ユーザー属性情報連携システムを実現するために、信頼性やモジュールの交換が容易であるという観点から、オープンソースのパッケージやオープンな規格などを中心に実現した。オープンソースやオープンな規格を採用したモジュールとその内容の一覧を表 5.1 に示す。

表 5.1 ユーザー属性情報連携システムで採用したオープンソース/規格

モジュール名	サブモジュール名	オープンソース/規格	採用したオープンソース/規格の概要
アイデンティティ管理	認証マネージャ・アクセス制御マネージャ	OpenID Connect 1.0	認証・認可のために Web サービスで広く利用されている規格
インフラ管理	ログマネージャ	Fluentd[5]	ログをストリーミングで収集するためのオープンソースソフトウェア
インフラ管理	ログマネージャ	Elasticsearch[6]	RESTful API を備えた検索・分析エンジンであり、ここでは Fluentd と組み合わせて利用した

ユーザー属性情報連携システムを実現するために、図 5.2 のようなソフトウェア構成で実装を行った。API を提供するシステムは安定性の観点から基本的に Java で実装を行い、エンドユーザー向けのインタフェースを提供する箇所は、近年 Web サービスを実現するために利用されている Node.js[7]による実装を行った。提供する API の一覧は、付録として付す。

ユーザー属性情報の表記変換ルールとして、今回の実装では Cloud Foundry[8]等によって実行することを想定し、JavaScript によるコードを記述できるようにした。

また近年注目されている DevOps[9]を考慮して、Java による実装箇所はサーブレットコンテナとして Apache Tomcat 上で実行を行い、Node.js による実装箇所は Docker 上で Docker コンテナとして実行するようにした。これにより、アプリケーションを更新する際にはコンテナを取り替えれば済み、アップデートが容易にでき

5 http://www.fluentd.org/
6 https://www.elastic.co/jp/products/elasticsearch
7 https://nodejs.org/ja/
8 コンテナ上で任意のコードを実行できる PaaS である
https://www.cloudfoundry.org/
9 開発と運用を連携して構築することで、運用の負荷を軽減するための手法
https://ja.wikipedia.org/wiki/DevOps

る。

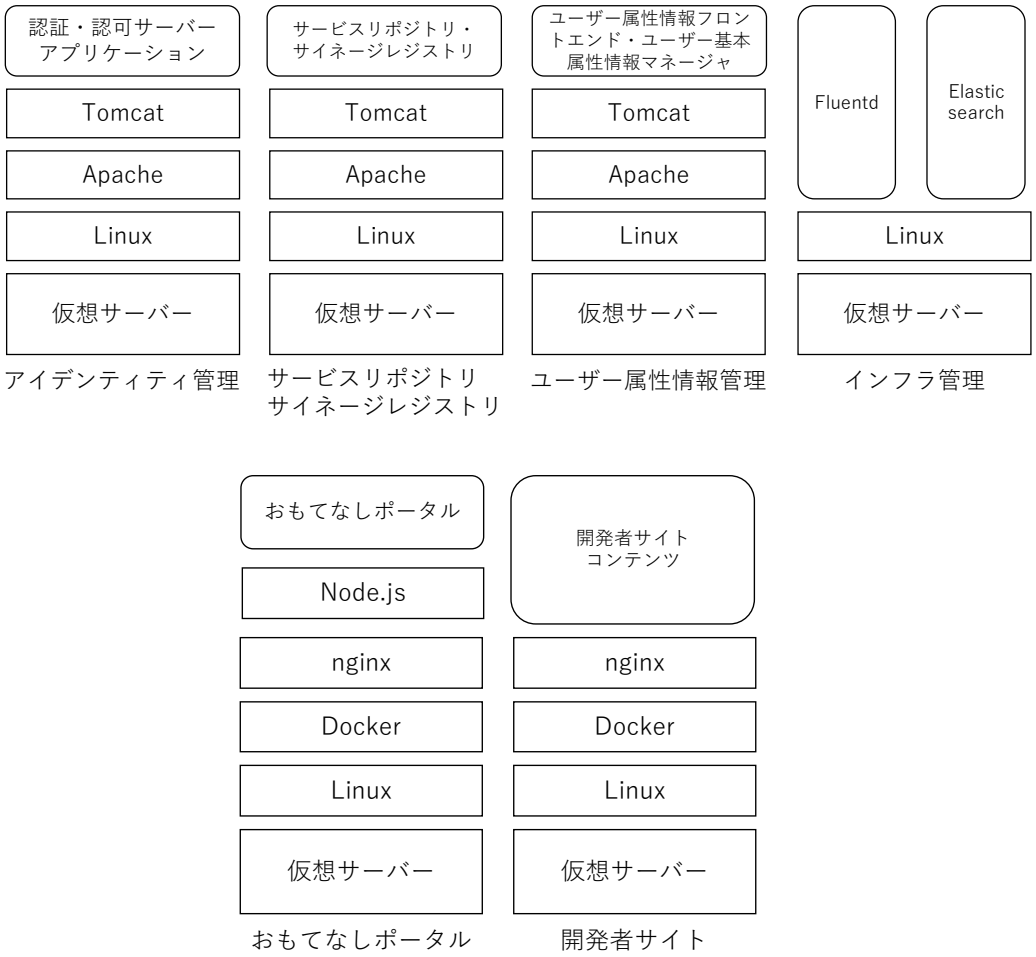


図 5.2 ユーザー属性情報連携システムのソフトウェア構成

また、サーバーはクラウドコンピューティング環境を利用して図 5.3 のように構成した。API へアクセスするすべてのリクエストは DMZ に設置したリバースプロキシを経由することで、セキュリティに配慮した。なお Object storage は、ユーザー属性情報のうち、画像をアップロードする必要があるものを格納した。データベースは基本的に PaaS を活用してデプロイを行った。

それぞれのサーバーは、1 台あたり 2 仮想 CPU、8GB メモリとして構築した。

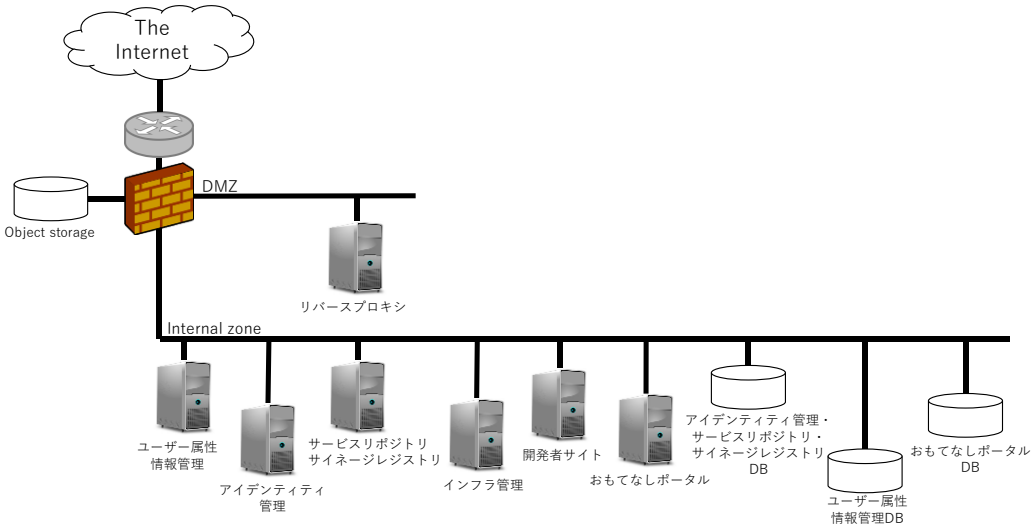


図 5.3 ユーザー属性情報連携システムのサーバー構成

5.3. 評価

5.2 節で説明したシステムを実際に構築し、ユーザー属性情報システムの評価を行った。ユーザー属性情報の項目名は、OpenID Connect 1.0 の scope 名、および claim 名として利用し、scope と claim が 1 対 1 で対応付くように利用した。なお評価のためのユーザー属性情報として、以下のシチュエーションを想定してのユーザー属性情報を採用した。

- サイネージの表示言語切り替え・道案内
  - ユーザーがあらかじめ希望した言語にサイネージの表示言語を切り替える
  - ユーザーの身体属性に応じて道案内のルートを切り替える
- 免税・ホテルチェックイン
  - 交通系 IC カードをかざすだけで、免税処理やホテルのチェックインなど、パスポートの提示が必要な手続きを代行する
- レストランのメニュー表示
  - ユーザーの食の好みやアレルギー等の禁忌に応じて、レストランで案内

するメニューを切り替える

- チケットの購入
  - 美術館やイベントの入場に必要なチケットをオンラインサイトで購入する

表 5.2 採用したユーザー属性情報の一覧

ユーザー属性情報の項目名	データ型	複数	説明
gender	String		性別。とりうる値は、ISO 5218 で規定された区分を利用。
age	String		年代。
native_language	String		母国語。とりうる値は、ISO 639-2 で規定された区分を利用。
priority_language	Array[String]	○	優先言語。とりうる値は、ISO 639-2 で規定された区分を利用。
destination	String		目的地。
user_interface	Array[String]	○	希望するユーザーインタフェース。
accessibilty	Array[String]	○	アクセシビリティに関する情報。
food_and_drink_prohibition	Array[String]	○	食の禁忌。
food_preference	Array[String]	○	食の嗜好。
email	String		メールアドレス。
first_name	String		アルファベット表記の名。
family_name	String		アルファベット表記の姓。
original_name	String		母国語表記の名前。
country	String		国。とりうる値は、ISO 3166-1(Alpha-3)で規定された区分を利用。
zip	String		郵便番号。

ユーザー属性情報の項目名	データ型	複数	説明
state	String		アルファベット表記の都道府県。または、州。
city	String		アルファベット表記の市区町村。
address_line_1	String		アルファベット表記の住所１（番地など）。
address_line_2	String		アルファベット表記の住所２（アクセスなど）。
original_address	String		母国語表記の住所。
year_of_birth	String		生年月日の西暦年。とりうる値は、４桁の数字を文字列で表現したもの。
month_of_birth	String		生年月日の月。とりうる値は、２桁の数字を文字列で表現したもの。
day_of_birth	String		生年月日の日。とりうる値は、２桁の数字を文字列で表現したもの。
telephone	String		電話番号。
passport_country	String		パスポートの発行国。とりうる値は、国際民間航空機関（ＩＣＡＯ）で発行されている“ICAO Doc9303“で規定されている内容に準拠した国を示すコード（3-letter code）である。MRZ からの読み取りできた値のみ提供する。
passport_name	String		パスポート記載の氏名。MRZ からの読み取りできた値のみ提供する。
passport_number	String		パスポート記載の旅券番号。MRZ からの読み取りできた値のみ提供する。
passport_gender	String		パスポート記載の旅券者の性別。MRZ からの読み取りできた値のみ提供する。

ユーザー属性情報の項目名	データ型	複数	説明
passport_birth	String		パスポート記載の旅券者の生年月日。MRZ からの読み取りできた値のみ提供する。
passport_nationality	String		パスポート記載の国籍情報。とりうる値は、国際民間航空機関（ＩＣＡＯ）で発行されている“ICAO Doc9303“で規定されている内容に準拠した国を示すコード（3-letter code）である。MRZ からの読み取りできた値のみ提供する。
issue_date	String		パスポート記載の旅券発行年月日。とりうる値は、"YYYYMMDD"形式で登録。
term_of_validity	String		パスポート記載の旅券有効期限年月日。とりうる値は、"YYYYMMDD"形式で登録。MRZ からの読み取りできた値のみ提供する。
entry_date	String		パスポート記載の上陸年月日。とりうる値は、"YYYYMMDD"形式で登録。
qualification_for_stay	String		パスポート記載の在留資格。
passport_mrz	String		パスポート記載のパスポートの旅券情報（機械読み取り部分）。とりうる値は、国際民間航空機関（ＩＣＡＯ）で発行されている“ICAO Doc9303“で規定されている 44 桁 2 行で構成される旅券情報部分の文字列情報(MRZ)を提供する。ただし MRZ 中の Personal number に英数字の他に"<"が含まれる場合には、"<"が省略される場合がある。

ユーザー属性情報の項目名	データ型	複数	説明
passport_image	String		パスポートの写し画像ファイルのＵＲＬを発行。なお URL はワンタイムのみアクセス可能である。

現在、本システムを利用したサービスが実証実験中[10]である。想定シナリオとして挙げた約 6 種類を含む、様々なシナリオで実証中であり、博士論文執筆段階で約 150 名以上の利用者に実際に利用されている状況である。

処理時間の評価として、ユーザー属性情報を取得するための一連のフローの処理時間を計測した。計測は 10 並列でクライアントからアクセスがあることを想定し、約 30 分間の測定を行った。計測結果を表 5.3 に示す。なお認証のフローとして、FeliCa の IDｍによる認証を行うフロー、ユーザー名・パスワードで認証を行うフロー、認証情報送信時に GET で行う場合と POST で行う場合のフローが存在するため、ステップ番号の末尾にそれぞれ idm、pass、get、post として付与した。最も遅くなる場合でも、認証してからユーザー属性情報を取得するまでに一連のフローに 0.895 秒で完了することから、実用上問題とならないと考えられる。

表 5.3 ユーザー属性情報連携システムの処理時間

ステップ	測定対象 API	概要	平均（秒）
1	GET /oauth2/authorize	認証要求	0.063
2-idm	GET /oauth2/IDmInternalAuthoriEndpoint	IDm 認証 Cookie 発行	0.018
2-pass	GET /oauth2/InternalAuthoriEndpoint	パスワード認証 Cookie 発行	0.018
3-get	GET /auth/login	認証情報送信（GET）	0.164
3-post	POST /auth/login	認証情報送信（POST）	0.259

[10] http://jpn.nec.com/press/201702/20170201_01.html

ステップ	測定対象 API	概要	平均（秒）
4-idm	GET /oauth2/IDmInternalAuthoriEndpoint	IDm 認証処理	0.352
4-pass	GET /oauth2/InternalAuthoriEndpoint	パスワード認証処理	0.354
5	POST /oauth2/token	アクセストークン発行	0.091
6	GET /user_attributes	ユーザー属性情報取得	0.11

ユーザー属性情報の表記変換ルールについて、JavaScript を使って記述をできる仕組みとした。例えば、以下のような変換を可能とした。

ユーザー属性情報の family_name の前に「Dear 」を付与する例：

```
'"return "Dear " + source"
```

ユーザーの母国語が日本語、英語、中国語の場合はそのまま、それ以外の場合には”other”という文字列に変換する例：

```
  '"var lang ="";
  switch (source){
      case "ja":
      case "en":
      case "zh":
         lang = source;
         break;
      default:
         lang = "other";
  }
return lang"
```

以上のように、チューリング完全な言語で自由に変換ルールを記述できることから、サービスの開発者が実行したいと思われる表記の変換は多くが実現できると思われる。また場合によっては外部のサービスとうまく連携することで、時々刻々と変化する為替レートの変換なども実現可能だと言える。

セキュリティについて、一般的な Web サービスで重要となる TLS による通信経路の暗号化や、SQL インジェクション対策等を施していると仮定すると、その他に脆弱性が存在しうる箇所はユーザー属性情報の表記変換ルールである。しかし、この変換ルール自体も Cloud Foundry 等、ユーザー属性情報連携システムとは独立した環境として動作させることが可能なため、セキュリティ上のリスクを軽減できる。

5.4. ユーザー属性情報取得の例

5.4.1. 位置情報に着目した背景

位置情報サービスは世界中の様々な研究者が取り組んでおり、様々なサービスが実現されている [45] [46]。位置情報サービスを実現するためには、高精度でロバストな位置推定手法が必要となる。歩行者の位置情報を推測するためには、歩行者の歩く環境にビーコンやセンサーを設置する必要がある手法と、歩行者が保持するセンサーだけで完結する手法が存在する。本研究では、前者をインフラ利用型測位と呼び、後者をデッドレコニング型測位と呼ぶ。インフラ利用型測位は、ビーコン等が、ビーコンの設置位置に関する情報を含むデータを歩行者の保持する受信機に送ることや、衛星を利用して測位を実現する。一方で、デッドレコニング型測位は、歩行者の身体に装着したセンサーで取得した歩行挙動のセンシング情報を歩行者の保持する携帯型端末等に送り、そこでリアルタイムに歩行挙動の解析を行い、位置を推測する必要がある。センサーを身体に装着することから、センサー自体がコンパクトになっている必要があるため、計算リソースや通信帯域、電源容量に制約がある。さらに、装着時の利便性から、センサーは有線接続ではなく、無線で接続されるべきである。本章では、このデッドレコニングに利用するセンサーについて着目し、デッドレコニング型測位のために必要となる情報をどのようなフォーマットで送るべきかについて検討を行った。

デッドレコニングは加速度センサー、ジャイロスコープ、地磁気センサーと気圧

センサー等を組み合わせて、位置をインクリメンタルに推定する方法として一般的に知られている [47]。本研究では特に断りがない場合は、デッドレコニングという言葉は歩行者のデッドレコニングを指す。図 5.4 は、デッドレコニングのイメージである。初期位置 N=0 から、歩幅L_1でO_1方向へ 1 歩進むと、N=1 の位置に移動する。どうように合計 3 歩歩くと、N=3 の位置へ移動するというように、インクリメンタルに位置を推定する方式がデッドレコニングである。この手法の長所は、特にビーコン等を設置することのように、環境に応じてあらかじめ準備をしなくとも、ユーザーの行動と高分解能な位置情報をすべての場所において取得できることにある。他方で、建造物に利用されている金属の影響で地磁気センサーの値が変動するなど、周辺環境のノイズの影響を受けやすいという問題がある。このような周辺環境のノイズを受けにくくすることが、従来からのデッドレコニングによる手法の課題点であった。

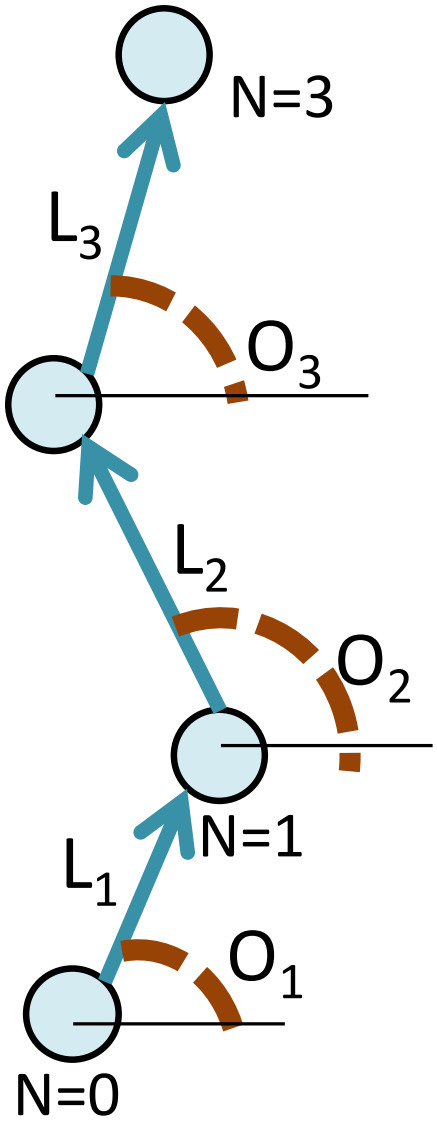


図 5.4 デッドレコニングのイメージ

デッドレコニングの分野では、多くの研究者たちが足にセンサーを取り付けて歩行者位置推定を行っていた [48] [49]。この手法では、加速度センサーとジャイロスコープから取得できた加速度、角速度を積分することによって、移動距離、移動方向を推測することができる。センサーから取得した値を積分し続けることから、センサー値に加わったノイズも積分されてしまうため、誤差が累積してしまうという問題がある。しかし、足が地面に設置する際には足の速度がゼロになることを利用してセンサー値から推測した速度をリセットするという zero velocity update という手法を適用して、高精度に位置を推測することができる。しかし本章では、日常生活で歩行者の位置を推定することを目指しているため、センサーの着脱が容易であることが重要だと考えている。その観点では、このセンサーを足に装着することを考えると、スニーカーやハイヒールなど靴の形状は様々に存在することと、デザイン性等を含めて装着できる方法を検討する必要があることから、実現は難しいと考えた。

一方で、センサーを歩行者の腰に装着して位置を推測する方法が存在する [50]。この手法は、ズボンのベルト等を活用したり、身体にバンドを巻き付けたりする等によってセンサーを装着することができることから、センサーを足に装着する方法に比べてセンサーの装着に対する制約が少ない。位置の推測は、センサーから取得した値を積分するのではなく、加速度センサーから取得した値を使って歩数を計測して歩幅の推測も行い、歩数と歩幅から移動距離を推測し、地磁気センサー等から取得した移動方向の情報と組み合わせてインクリメンタルに行う。このような観点から、本研究では腰にセンサーを装着する手法を採用する。

デッドレコニングのために利用するセンサーは一般的に、3 軸の加速度センサー、3 軸のジャイロスコープ、3 軸の地磁気センサー、気圧センサー、GPS モジュールを含む。本研究では、本研究では、これらのセンサーを含む歩行者デッドレコニングユニット（Pedestrian Dead-reckoning Unit: PDU）を構築した。この PDU は Bluetooth によって無線経由で測定したデータを携帯端末に送信する。この本研究で構築した PDU は、センサーの生データを送信するのではなく、PDU 内で処理を行って抽象化したデータを無線経由で送信することで、生データを送信する場合に比べて PDU から送信するデータ量を小さく抑えることを目的としている。

本研究で着目しているデッドレコニングは、歩数や歩幅、移動方向から歩行者の位置をインクリメンタルに推測する。歩数は PDU で計測した加速度値のピークを数えることによって取得する。歩幅は、あらかじめ登録しておいた基準となる歩幅

を元に、単位時間あたりの加速度の大きさ等から歩幅が平常時よりも長いか短いか推測を行い、歩幅を調整する。歩行者の向いている方角は、ジャイロスコープから測定した角速度と、磁気センサーから測定した地磁気から推測を行う。この方角を計測するためには、PDU の座標系から、歩行者の身体の座標系である進行方向が X 軸、右手方向が Y 軸、地面方向が Z 軸であるような座標系に変換する必要がある。この座標系のイメージを図 5.5 に示す。さらには、この歩行者の身体の座標系が地球全体の座標系と比べてどこにいるのかを把握する必要があるため、地球の座標系である North-East-Down（NED）座標系へと変換する必要がある。このような一連の座標系の変換について、変換の精度が非常に重要である。さらに、歩行中である、立ち止まっている、階段を昇降している等の歩行者の行動を利用する方式も存在する [12]。このように、歩行者の位置を推測するためには、PDU からの生データが有用なのではなく、歩数や歩幅、歩行者の行動、PDU の姿勢などが重要であると言える。

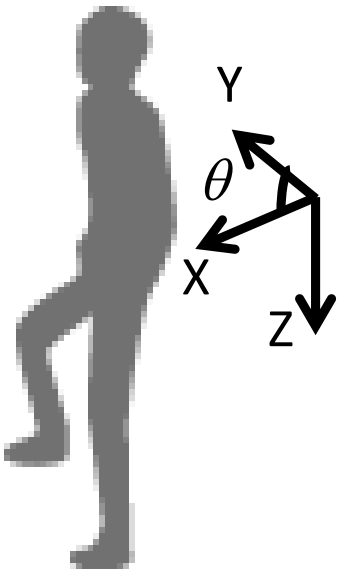


図 5.5 歩行者の座標系の定義

デッドレコニング型測位は、様々な企業や研究者によって、センサーモジュールが提案されてきた [51] [52] [53] [54]。このようなセンサーの一部は、センサーの生データを送るだけではなく、センサー自身の姿勢を計算して送信するようなセン

サーモジュールも存在している。しかしこれらのセンサーモジュールを利用して歩行者の位置情報を推定するためには、携帯端末等に高周期でセンサーデータを送信する必要があるため、有線で接続するか、無線 LAN など、高速ではあるが消費電力が他の無線通信方式に比べて比較的高い無線通信方式で接続する必要がある。このため、センサーモジュールの携帯性の観点から、歩行者の位置情報を計算するためにこのようなセンサーモジュールを利用することは適していない。

本提案手法の貢献は、歩行者デッドレコニングを実現するための PDU について、API デザインと、コンパクトなメッセージフォーマットを定義したことである。本提案手法は、PDU と携帯端末が狭帯域のネットワークで接続されていて、PDU に搭載されている CPU が低スペックであったとしても、歩行者の位置を推測できることを実現する。

また腰に装着したセンサーを利用して推測した、歩いている、階段を上っている、階段を下っている等の歩行者の行動と、地磁気センサーの測定値と地磁気の本来の値、ジャイロスコープのキャリブレーション実施からの経過時間などから推測されるセンサー信頼性を組み合わせて、位置推定結果に対するノイズの影響を削減することを過去の研究で実現している [12]。例えば、気圧センサーから高さを求める場合は、例え高さが同じところにあったとしても、空調の動作している高気密な部屋を出入りすると気圧の変動が起こる場合があり、気圧値から推測される高さに誤差が生じることがある。このため、気圧センサーの値だけを常に利用して高さを推定することは難しい。しかし、階段を上り下りする際などには、腰につけた加速度センサーの値から高さを推定することができないため、気圧センサーの値を利用して高さを推定する必要がある。このため本章では、歩行者の行動に基づいて歩行者の位置を推測する手法を提案した。歩行者の行動は、腰に装着したセンサーを組み合わせて推測する。例えば歩行者の行動が階段を上っていると推測された場合には、歩行者の高度を推測するために気圧センサーの値から求めた高さを利用し、歩行してると推測された場合には、高度は変わっていないと仮定して、気圧センサーの値をキャリブレーションする。

センサーの信頼性は、時間や環境の変化によって変動する。例えば、重力加速度の大きさや地磁気の大きさ、キャリブレーションを行ってからの経過時間などの値を利用することができる。このセンサーの信頼性を算出することで、歩行者の位置情報を推測するための手法を選択したり、キャリブレーションを行ったりすることを過去の研究で示している [12]。このため本研究でも、このセンサーの信頼性を取

り入れて歩行者の位置を推測している。

5.4.2. 位置情報取得のためのシステム構成

5.4.2.1. 全体像

提案した PDU を利用したナビゲーションシステムの装着例を図 5.6 に示す。また、PDU の外観を図 5.7 に示す。PDU はユーザーの腰に装着して利用する。PDU はセンサーの生データから、歩行距離情報（歩数、歩幅に関する情報)、歩行者の行動、PDU の姿勢を計算して、Bluetooth 経由で送信する。Bluetooth の通信プロファイルとして、センサー内部の UART による通信と親和性の高い Serial Port Profile を採用している。本研究では、歩行者の行動として、歩行する、動いている、立ち止まっている、身動きせず静止しているという 4 パターンを取得した。この 4 パターンを取得することで、センサーをキャリブレーションするタイミングを認識することと、歩行者が歩いているのか立ち止まっているのかについて、歩行者の位置情報を利用する携帯端末上のアプリケーションで、例えばユーザーへの案内タイミングを調整する等、様々に応用することを想定している。センサーの生データを取得していれば、階段の昇降など、さらに多様な行動を取得できることは従来研究で明らかにしている [12]が、PDU 上で歩行者の行動を推測しているために計算資源が限られている関係から上記 4 パターンに絞って取得した。

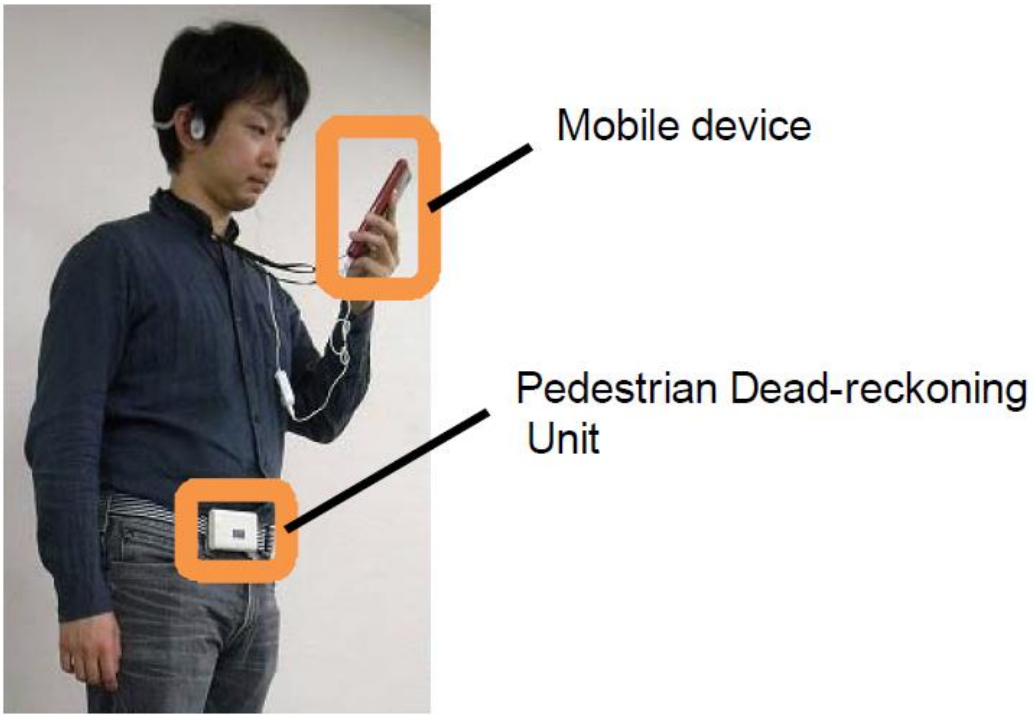
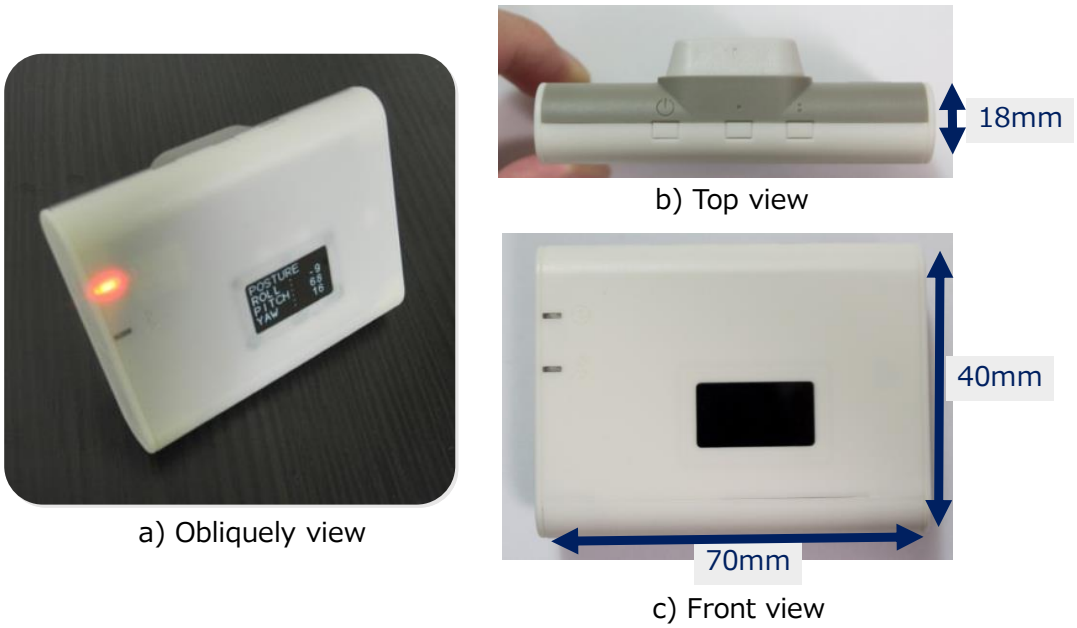


図 5.6 歩行者デッドレコニングユニットの装着例



a) Obliquely view

b) Top view

c) Front view

図 5.7 PDU の外観

PDU を含む歩行者位置情報を利用したシステムの全体像を図 5.8 に示す。この PDU では、先述のように PDU 内部で歩行者のデッドレコニングに必要な情報の計算を行うことによって、PDU のデータを利用して歩行者の位置を推測する携帯端末と PDU 間の通信データ量を削減することに着目している。位置を推測するために必要となるデータは 0 節で列挙したとおり、歩行距離情報、歩行者の行動、PDU の姿勢である。これらの情報を正確に計算するためには高周期でセンサーデータを取得する必要がある。携帯端末上では歩行者が移動したときのみ歩行者の位置情報の更新情報が取得できれば良く、歩行者の位置情報が変動する頻度はセンサーのサンプリング周期に比べるとかなり低い。このため、PDU の内部で高周期なセンサーデータを処理して位置情報や身体の移動に関する情報等を推測すれば、PDU から携帯端末へ送信するデータは高周期で送る必要はなくなる。このため、PDU と携帯端末間のデータ通信料を削減するためには、PDU 内部でセンサーデータを処理して抽象化したデータを送ることが重要である。

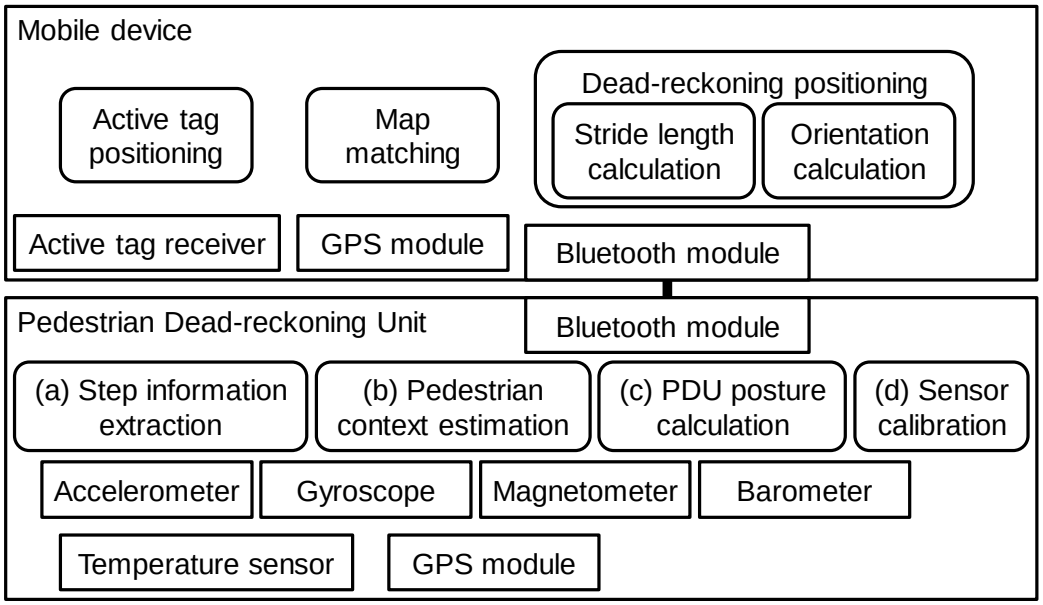


図 5.8 歩行者の位置を推定するためのシステムの全体像

携帯端末では PDU から送信されたデータを利用して歩行者の位置を推測し、歩行者ナビゲーションシステム等の位置情報サービスで利用する。本提案では、一般的に現在主流のスマートフォン等の携帯端末には GPS 受信機が搭載されていることが多いことから、PDU から送信する情報はデッドレコニングに関する情報のみに限定している。デッドレコニングによる測位結果は時間と共に誤差が累積することから、位置情報サービスで実用的な精度を得るためには、GPS やビーコンによる方法等を組み合わせる必要がある。本提案手法では、アクティブタグによる測位手法と、マップマッチングを利用した。詳細は 5.4.4.2 節で説明する。

5.4.2.2. PDU の動作

PDU は周期的にセンサーデータを取得する。サンプリング周期として、加速度センサーとジャイロスコープは高周期でサンプリングすることが精度を高める上で重要であることから今回採用しているセンサーの上限値である 100Hz を採用し、地磁気センサーと気圧センサーは移動に伴う身体運動は 1 秒間に 4,5 回程度が上限であると想定されることから 10Hz を採用した。また GPS 信号は 1 秒間に 1 回取得するように設定した。これらのセンサーデータをサンプリングし、PDU 内では歩行距離情報と、歩行者の行動と、PDU の姿勢を計算する。この計算周期は 10Hz で行い、

同時にセンサーデータのキャリブレーションが可能であればキャリブレーションを行う。この過程で算出された情報は、PDU の設定に応じて、1Hz から 10Hz のいずれかの周期でメッセージを構築し、Bluetooth 経由で携帯端末へ送信する。この処理フローを図 5.9 に示す。

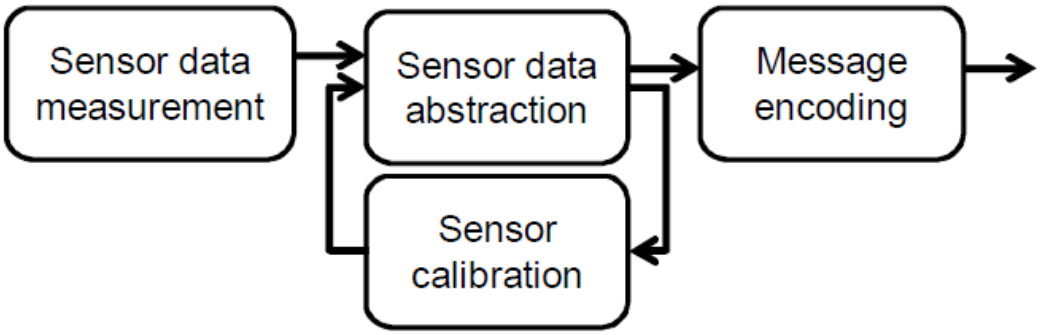


図 5.9 PDU の処理フロー

5.4.2.3. PDU から送信するメッセージのフォーマット

PDU から送信するメッセージには、メッセージを受信したアプリケーションがデッドレコニングに基づいた測位を実現するために有用である表 5.4 に示すような情報が含まれている。メッセージは、狭帯域の無線通信で送信することを想定して、センサーの生データをそのまま送信するよりもコンパクトになるように注意して設計を行っている。

この PDU が送信するメッセージのフォーマットには、GPS 受信機で一般的に採用されている NMEA 0183 フォーマット [55]に親和性のある形式を提案した。このメッセージフォーマットは、データを ASCII 文字列として表現を行い、それぞれのデータはカンマ区切りで 1 メッセージに埋め込まれ、メッセージの種別を表すヘッダと、メッセージの区切りを表す改行によって構成される。このため、この PDU を利用してアプリケーションを構築する開発者は、Bluetooth 経由で送信される ASCII 文字列で表現されたメッセージをプログラム内で処理すれば良いため、容易にデッドレコニングに基づく測位をアプリケーションに取り込むことができる。また、さらにメッセージサイズをコンパクトにするためには、ASCII 文字列で表現されている NMEA 0183 フォーマットではなく、バイナリ表現など、別の方式を採用することで実現することが可能であるが、GPS 等、既存の測位手法で採用されているメッ

セージフォーマットとそろえてプログラマーの学習コストを下げる観点から、このようなフォーマットを採用した。

なお本 PDU では、歩行者の位置が変動するために必要な時間は 0.5 秒以上かかると仮定し、メッセージの送信周期は 2Hz を選択した。

表 5.4 PDU から送信するメッセージに含まれる情報の一覧

Category	Data Name	Used in the Experiment	Total Bytes
Packet information	Message header and tailer	Yes	10
Packet information	Date and time	Yes	24
Packet information	Sampling rate	Yes	3
Raw sensor data	Magnetism	No	30
Raw sensor data	Air pressure	Yes	7
Raw sensor data	Temperature	No	11
Step information	Number of steps	Yes	11
Step information	Duration of a step	No	6
Step information	Acceleration amplitude	Yes	24
Step information	Maximum acceleration	No	24
Step information	Maximum velocity	No	24
Step information	Maximum gradient value of filtered acceleration of a step	No	8
Step information	Amplitude of air pressure	No	8
Pedestrian contexts	Contexts	Yes	2
PDU posture	Posture calculated using angular rate and acceleration	Yes	40
PDU posture	Posture calculated using magnetism and acceleration	Yes	17

5.4.3. PDU の内部処理アルゴリズムの詳細と実装

提案する PDU の特徴的な機能は、図 5.8 中の角丸四角形で表している、センサーデータの抽象化と、センサーのキャリブレーションである。本節では、このセンサーデータの抽象化と、キャリブレーションについてそれぞれ詳細を説明する。

5.4.3.1. センサーデータの抽象化

PDU は、これまでに述べたように、歩行距離情報、歩行者の行動、PDU の姿勢を計算する。

歩行距離情報には、表 5.4 の Step information として記載しているように、歩数に関する情報と、1 歩に要した時間、加速度の振幅、1 歩の最大加速度、加速度の 1 歩間の最大勾配、1 歩の最大速度、気圧の振幅が含まれる。これらの情報は、携帯端末に送信され、携帯端末上で移動距離を計算するための基礎情報として利用されることを想定している。この処理は、図 5.8 のうち、(a)で示している処理である。

歩数と、1 歩に要した時間と、加速度の振幅と、加速度の 1 歩間の最大勾配は、ローパスフィルターを通した加速度を利用している。このローパスフィルターは、4 次のバターワースフィルターとして構成しており、カットオフ周波数は人間の歩行として 1 歩の所要時間が 0.7 秒よりも短くなることはないという仮定から 1.4Hz とした。バターワースフィルターは通過域にリップルがない代わりに、カットオフ周波数以降の阻止域の減衰傾度が緩やかであることを特徴とするフィルターである[56]。このほかの代表的なフィルターには、チェビシェフフィルタや楕円フィルターなどが存在し、いずれもバターワースフィルターよりも急峻な特性を持つが、通過域にリップルが発生するために、通過域の信号の特性に影響が生じてしまい、歩行距離情報として推測するデータに影響を及ぼすため、本提案ではバターワースフィルターを採用した。バターワースフィルターは次数が高くなるとフィルターは急峻な特性を持つために、次数を高めるほどよいが、計算に必要な処理量も合わせて上がってしまうため、本提案では-80dB/decade となる 4 次のフィルターとした。

歩行距離情報のそれぞれについて計算する方法について説明する。歩数は、このバターワースフィルターを通した 3 軸加速度情報を基に、3 軸の合成加速度を算出して、そのピークを数えることで計測した。歩行者の歩幅の推定手法として、下記の数式を利用した。この数式は、既存研究を基にしている [57] [58]。

$$l_{step} = \alpha\sqrt[4]{a_{range}} + \beta \quad \text{式 5.1}$$

ここで、αは歩幅の係数であり、a_{range}は 3 軸合成加速度の最大値と最小値の差分として PDU 内で算出した加速度の振幅であり、βは歩幅のオフセット値である。このαとβは個人によって異なるパラメーターであるため、事前に決められた距離を被験者が歩行することによって調整を行う。

1 歩に要した時間は、この合成加速度のピーク間の時間を利用した。加速度の振幅は、フィルター処理後の合成加速度について、PDU からメッセージを送信する周期間における最大値と最小値の差分を利用した。加速度の最大勾配は、3 軸合成加速度のフィルター処理後の最大値について、最新の歩数と、その一つ前の歩数間の差分を利用した。これ以降の情報は、ローパスフィルターによるデータへの影響を防ぐため、加速度センサーの生データを利用した。最大加速度は、PDU からメッセージを送信する周期間の 3 軸合成加速度の最大値を利用した。最大加速度は、下記の式によって計算した。

$$v_{max} = a_{max} \times \Delta T \quad \text{..... 式 5.2}$$

ここで、ΔTは加速度のサンプリング時間を表しており、提案 PDU では先述の通りサンプリング周期が 100Hz であるため、10ms である。

気圧の振幅は、PDU からメッセージを送信する周期間における気圧センサーで観測した値の最大値と最小値の差分を利用した。

歩行者の行動も図 5.8 の(b)で示されているように PDU 内で計算を行う。推定のフローをに示す。歩行者の行動は、歩数の変化量と 3 軸合成加速度の変化量と、3 軸合成角速度の変化量を利用して推定している。もし 3 軸合成加速度の最大値と最小値の差分R_aと、3 軸合成角速度の最大値と最小値の差分R_gが、それぞれ事前に定義した閾値TH_aとTH_g以内であれば、歩行者の行動は静止状態とする。タイムウィンドウのT_s内における歩数の増加率ΔSがTH_wを超えている場合には、歩行者の行動は歩行状態とした。また、ΔSがゼロの時は、立ち止まり状態とした。それ以外の場合には、歩行中という状態として定義した。上記の閾値は、事前に立ち止まっている場合のセンサーデータを収集し、キャリブレーションのために適切に利用可能な値を検討し、TH_aは$0.05\mathrm{m}/s^2$、TH_gは$0.12\mathrm{rad}/s^2$、T_sは 3 秒、TH_wは 0.5 と経験的に設定した。

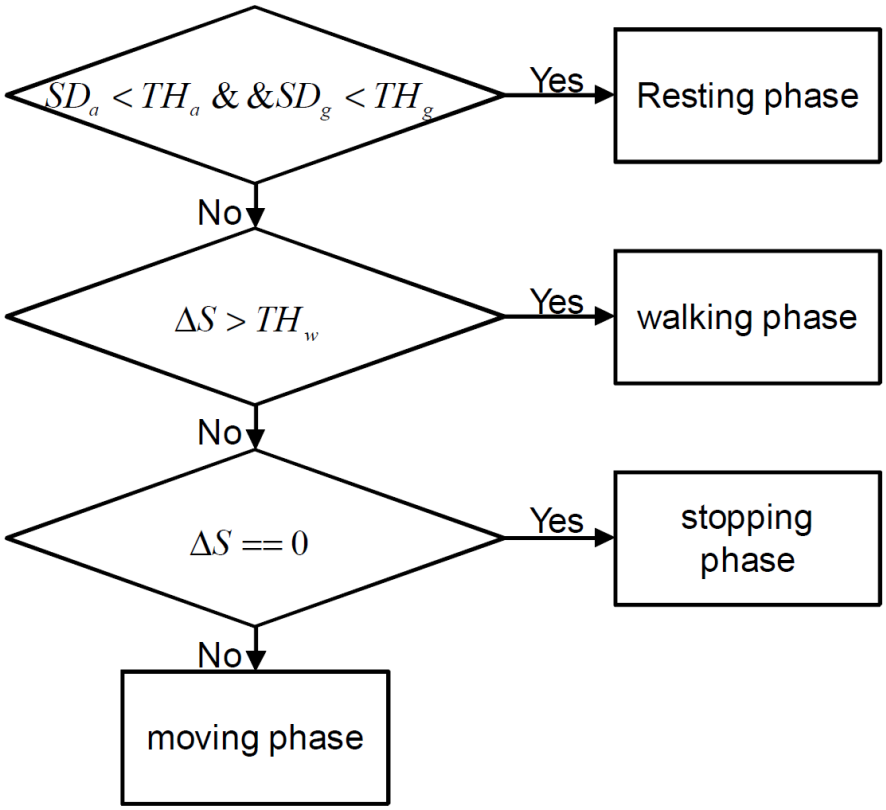


図 5.10 歩行者の行動推定フロー

PDU の姿勢は、2 種類の方法から計算する（図 5.8(c)）。最初の方法は、ジャイロスコープから取得した角速度と加速度センサーから取得した加速度を利用する方法である。もう一つの方法は、磁気センサーから取得した地磁気の値と加速度センサーから取得した加速度を利用する方法である。PDU の姿勢は、携帯端末でどちらを採用するか選択できる仕組みとした。これは、地図やビーコン等、他の測位手法と組み合わせることでどちらの手法を採用するかを携帯端末で選択するために 2 種類両方の姿勢情報を送る方針とした。PDU で採用した姿勢推定手法は、PDU の姿勢を PDU 内でリアルタイムに推定する必要があったため、非常にシンプルな方式で推測している。

角速度を利用する方式は、図 5.11 に示すように次のように実現した。歩行者の行動が静止状態の場合、PDU の姿勢は加速度センサーで取得できる加速度は重力加速度と一致しているため、加速度センサーの加速度から姿勢を推定する。その他の場合には、ジャイロスコープで測定した角速度にサンプリング時間を掛けることで、

PDU の姿勢を推定する。磁気センサーによる姿勢の推定手法は、加速度センサーから歩行者の行動が静止状態で取得した重力加速度と、磁気センサーから取得した 3 軸の磁気値を地磁気であると仮定し、地磁気の向きから PDU の姿勢の推定を行う。

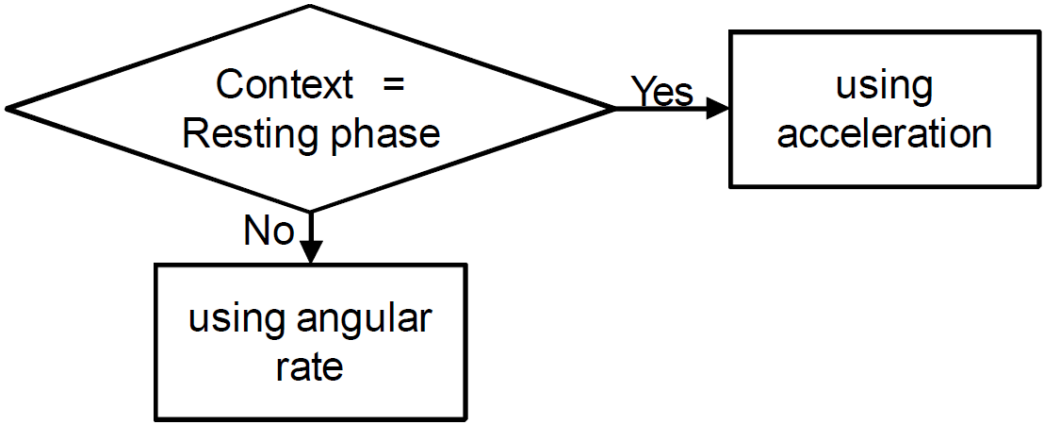


図 5.11 PDU 姿勢推定アルゴリズムの決定フロー

5.4.3.2. センサーのキャリブレーション

PDU 内に搭載されたセンサーは、静的な方法と動的な方法により、温度補償、スケールファクター補償、およびオフセットについてキャリブレーションを行っている。静的なキャリブレーション方法は、あらかじめセンサーの個体差を測定してキャリブレーションを行う方法である。動的なキャリブレーション方法は、最新のセンサーの測定値からキャリブレーションに必要なパラメーターを推測する。

加速度センサーの温度補償とスケールファクター補償は、メーカーによる設定値によって静的に実現した。ジャイロスコープは、カットオフ周波数を低くしたローパスフィルターによって直流成分を抽出し、この値をオフセット値として測定結果から除外するという動的なキャリブレーションを行った。なお歩行者が静止している場合には PDU で計測される角速度も 0rad/sであると仮定できるため、このフィルターは歩行者の行動が静止状態の場合のみ動作させてキャリブレーションを実現した。本手法では、カットオフ周波数が 0.5Hz である 4 次のバターワースフィルターでこのジャイロスコープのキャリブレーションを実現した。

5.4.4. 位置情報取得システムの評価

5.4.4.1. 既存手法と本 PDU とのメッセージサイズの比較

PDU から歩行者デッドレコニングに関するすべてのメッセージを送信する場合のメッセージサイズと、既存のセンサーとの比較を表 5.5 に示す。本 PDU はメッセージサイズの縮小と、既存センサーと比べて最大でおよそ 1/8 程度のメッセージサイズの縮小を実現できた。DRM4000 はデッドレコニングモードが実現されているが、本 PDU では地図情報や他のビーコンなど、様々な情報を組み合わせて歩行者の歩幅を携帯端末で補正できるよう、位置情報を携帯端末に送るのではなく、歩幅等の抽象化されたセンサー情報を送信している。このため、DRM4000 では 20Hz のセンサーの生データと、2Hz のデッドレコニングのデータを送信する場合を仮定してメッセージサイズを計算している。

表 5.5 既存のセンサーユニットと本 PDU のメッセージサイズの比較

Sensor name	Message size [bits]	Message update rate [Hz]	Required transfer rate [bps]
Mti-G [9]	376	100	37600
InertiaCube3 [8]	176	180	31680
DRM4000 [10]	2848*	2	5696
Our PDU	1992	2	3984

* Message size is the sum of dead-reckoning message size and raw sensor message size in 10 Hz.

本 PDU を実現する上で、本 PDU に搭載した Bluetooth によるメッセージ転送速度の最大値は 90kbps であり、本 PDU が必要とするメッセージの転送速度は表 5.5 に示すように 3.984kbps である。この結果から、本 PDU は携帯端末で測位をするために必要な情報を既存のセンサーモジュールよりも小さいデータサイズで送信できるといえる。さらに、メッセージの更新周期は他のセンサーモジュールよりも遅くできるといえる。このようなデータサイズの縮小を実現するためには、PDU から送信するデータを生データではなく抽象化したデータにして、さらにタイミングやサンプリング周期が重要である、歩行距離情報、歩行者の行動と PDU の姿勢については PDU 内部で処理をするようにしたことでこのような機能を実現できたといえ

る。

5.4.4.2. 評価実験と実験結果

本提案で構築した PDU を評価するため、携帯端末で歩行者の位置情報を推定するシステムを構築した。構築したシステムの全体像は図 5.8 に示すようになっており、PDU を利用したデッドレコニングと、アクティブタグによるセルベース測位と、マップマッチングを組み合わせて測位を行った。本システムでは、セルベースによる測位として、アクティブタグの発する ID を受信すると、自身の位置情報を ID に紐付けた位置にするというシンプルな方式を採用した。また、メッセージの送信データサイズは、今期採用した Bluetooth モジュールの通信速度よりも小さい値であった。

評価システムを構築するために、PDU から送信するメッセージは表 5.4 のうち「実験に利用した」列が「Yes」になっているデータのみを利用して測位を行った。このため、実験に利用したシステムで利用するために必要なメッセージサイズは 1 メッセージあたり 138 バイト（1104 ビット）であった。このメッセージを 2Hz で送るため、必要な送信周期は 2208bps であった。

評価実験では、図 5.12 で緑色の点線で記述したルートを、図 5.12 右上の黒いクロスマークで表したところから矢印で表現している向きに左回りに一周、被験者が歩行した。このルートは全長 331 メートルであり、屋外の階段や、屋内のエスカレーターが含まれ、2 フロアで構成されている。またオレンジ色の丸印は、RFID マーカーの設置箇所を表している。水色の長方形で囲んだエリアは、屋内でフロアが変化する箇所は限定されることを利用して、あらかじめエスカレーターの緯度経度を登録しておき、フロアが 2 階から 1 階へ変化したことを PDU 内の気圧センサーの変化量から検出した際に、歩行者の位置をエスカレーターの出口に補正するというようなマップマッチングを行った箇所である。評価システムによる測位結果は図 5.12 の赤い実線で示した通りである。

この結果、この評価システムでは測位誤差が平均 2.24 メートルで、標準偏差が 1.38 メートルで測位を実現できた。この結果から提案 PDU は、従来のようにセンサーの生データをセンサーモジュールから高頻度で送信しなくとも、歩行者デッドレコニングのために抽象化した情報を低周期で送信することで、通信帯域が狭い通信路を通して、従来手法よりもコンパクトなメッセージサイズで歩行者測位システムを実現できるということが分かった。

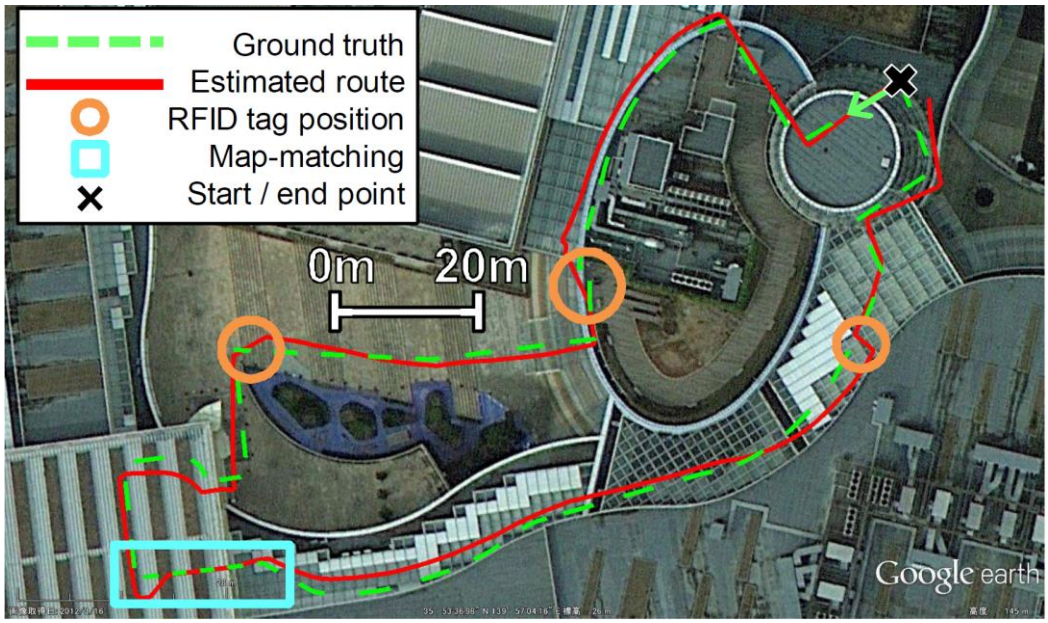


図 5.12 評価実験ルートと提案 PDU を用いた評価システムによる測位結果の抜粋

5.5. まとめ

本章では、提案アーキテクチャの Huot を実現するための構成要素のうち、ユーザー属性情報連携システムについて述べた。いくつかの実用的なシチュエーションを想定してユーザー属性情報を定義し、ユーザー属性情報連携システムを実装し、API の応答時間、表記変換ルール、セキュリティの観点から評価を行い、実用性を示した。

さらにユーザー属性情報を取得する例として、デッドレコニングによる位置情報を取得する手法について説明した。デッドレコニングは位置情報だけではなく、歩行者の行動も同時に取得できることから、本章で紹介したようなユーザー属性情報のサンプルとした。Bluetooth のような狭帯域かつ低消費電力の無線通信方式で接続された携帯端末上で歩行者の位置情報を利用したサービスを実現するための新たな PDU を提案した。提案 PDU の特徴として、携帯端末上でデッドレコニングによる歩行者の測位を行うために必要な情報を PDU 上で抽象化してから携帯端末へ送信することによって、PDU と携帯端末間で通信するメッセージのサイズをコンパクト

にしたことが挙げられる。評価として、従来のセンサーと比較して提案 PDU が採用しているメッセージサイズはコンパクトになっていることを示した。さらに、提案 PDU と携帯端末を組み合わせて、歩行者の位置を推測する評価システムを構築し、実験を行った。その結果、提案 PDU による低周期でコンパクトなデータを利用しても、他の手法と組み合わせることによって歩行者の位置を推定するシステムを実環境でも測位を現実的な精度で実現できることが示せた。

第 6 章

IoT デバイス連携システム Colapi

6.1. 背景

ユビキタスコンピューティングは学術界でも産業界でも最終目標として 1980 年代から注目されている、コンピューターと現実世界を結びつけて我々の生活環境を改善するための概念である [3]。無線通信技術や組み込み用マイクロコントローラー等に代表されるようなユビキタスコンピューティングを実現するための要素技術の発展に伴って、ユビキタスコンピューティングの概念が実現されつつある。このユビキタスコンピューティングを実現するための技術は近年では IoT とも呼ばれるようになっている [59]。IoT を活用したデバイスは、すでに我々の生活に実展開されつつある。インターネット接続機能を持ち、インターネット経由で相互に通信できるような機能を持つ IoT デバイスと呼ばれるデバイスを多くのメーカーが製造しつつある [1]。

未解決の大きな課題として、スマート環境を実現するための IoT デバイスをどのように組み合わせるかが挙げられる。考慮しなければならない一つの要素として、近年の IoT サブシステムは閉鎖的で、各企業で独立したシステムを構築している場合が多い。一般的に、各デバイスメーカーが自社の機器に適した API を独自に定義して、それを API として実装して提供している場合が多い。ユーザビリティを改善するために、デバイスメーカーは自社のデバイスを遠隔操作するための公式アプリケーションをスマートフォンやタブレット向けに提供していることが多い。この結

果、デバイスメーカーは自社のアプリケーションからのみデバイスの操作を行えるようにしていたり、API を公開しないでいたりすることから、各社でクローズした独自の IoT デバイスのネットワークが構成されることとなる。現在の IoT 環境はこのように、IoT デバイスのメーカー毎に独立した IoT 環境となってしまっていると言える。

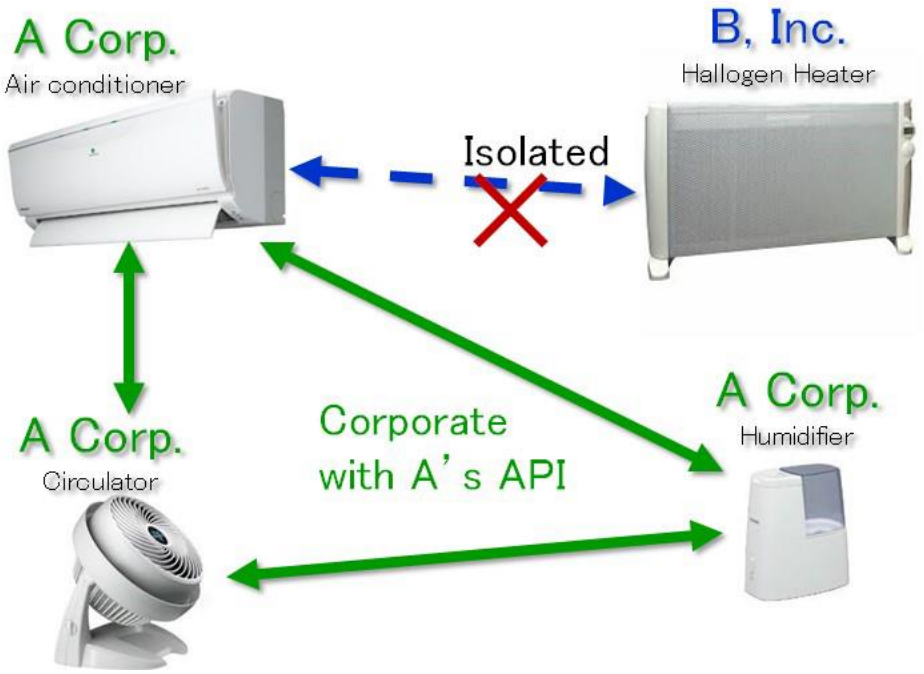


図 6.1 クローズドな IoT サブシステム

デバイスの協調動作を実現する上で考慮する必要のある課題として、IoT デバイスの多様性と協調動作のパターンが限定的でないことが挙げられる。ユビキタスコンピューティングや IoT の概念に基づくと、より良いサービスを実現するためには、協調動作可能な IoT デバイスは限定されるべきではない。例えば室内の気温や湿度などを最適な状態で維持するために、エアコンだけを利用するのではなく、サーキュレーターや加湿器や電動の窓開閉システムなどを組み合わせて動作させることで、室内の空調をより多様な方法によって制御する事が可能となる。さらに、発電所が電力需要などの状況に基づいて各家庭に節電を依頼し、利用可能な電力から生活環境の空調を住人とって最適に制御することなども可能となる。このように、メーカ

ーや機器の種別などの垣根を越えてデバイス間の通信や連携を実現するようなプラットフォームを構築することが本研究の目的である。しかし世の中に存在しているすべての IoT デバイスをカバーできるような世界標準を作ることは簡単なことではない。

このような課題を解決するために、IoT-Aggregator という概念が提案されている [2]。この IoT-Aggregator はアグリゲートコンピューティングモデルに基づき、サービスやデバイスやアプリケーションや場所やユーザーなどの情報を、デバイスやサービスやアプリケーションの作成者に依らず、誰でも自由に簡単に扱うことを目指したした仕組みである。

アグリゲートコンピューティングとは、ネットワークで接続された IoT デバイスやコンピューターが総体として最適化を行い、サービスを行うというコンピューターシステムの概念である。例えばこのアグリゲートコンピューティングの概念に基づくと、複雑なアクセスコントロールやデータ処理は、IoT デバイス上で行わないで、IoT デバイス以外の計算リソースの潤沢な環境で実現する。このようにタスクを適切に分割して適切な箇所で実行することで、デバイスをコンパクトで安価に実現しつつも、ユーザーの要求を十分に実現することが総体として可能となる。このようなグローバルでのサービスやデバイスやアプリケーションや場所やユーザーの協調動作が、オープンデータやオープンガバメントや IoT などのオープンアプローチによって一般的になりつつある。このため、オープンコラボレーションのためのプラットフォームとして IoT-Aggregator が設計された。この IoT-Aggregator は、台湾でのスマートハウスのプロジェクトや、OPaaS.io と呼ばれるユーザー属性情報を共有するためのフレームワーク等、様々なプロジェクトで応用されている。

本研究では、デバイスのコラボレーションのための仕組みにこの IoT-Aggregator の概念を適用した。アグリゲートコンピューティングを実現するため、異なるメーカーの構築した IoT デバイスを相互接続可能にするだけではなく、異なる API を持つデバイスに対して標準化した共通の API でアクセスできる仕組みを構築した。IoT デバイスのメーカーを超えた API 標準化は非常に重要であるが、一方でメーカーは他社とは異なる新しい機能を IoT デバイスに取り込んで行きたいと考えているため、API の標準化の段階で考えられていなかった機能が取り込まれてしまうと、ある時点で API の標準化を行ってもすぐに古くなってしまうことが考えられる。

柔軟性を維持して API の標準化を実現するために、本研究ではデバイスプロファイルと呼ばれる機械可読な形式で IoT デバイスの API を記述し、これを API 形式の

変換に利用する方式を提案した。標準化されたAPIリクエストは本提案手法がプロキシとして動作することによって、デバイス独自のAPI形式に自動的に変換されてリクエストされる。このプロキシのメカニズムによって、デバイスの作成者はデバイス自体に変更を加えなくとも、作成したデバイスのAPI仕様などをデバイスプロファイルに記述をして本フレームワークに登録するだけで作成したデバイスを標準化に対応させることができる。

過去に取り組まれた関連研究の調査から、本研究ではデバイス間やアプリケーション間で相互運用性を実現するための新たなシステムColapiを提案することとした。提案手法のアイディアの有効性を、本研究のようなデバイスプロファイルを取り入れることでデバイスの相互運用性を実現できることを過去の研究を通して明らかにした [60]。この既存研究ではアイディアを検証するためのプロトタイプにとどまっていた。本研究では、より実用的となることを目指してIoT-Aggregatorの概念に沿うように設計し直し、実装を行った。

本研究では、デバイスプロファイルを 2 種類提案した。一つは Generic Device Profile（GP）であり、もう一つはDevice Specific Profile（SP）である。GPはデバイスクラス毎に備えるべき標準的な API のセットを定義する。SP は GP で定義された標準的なAPIをデバイス個別のAPIへ変換する方式を定義している（GPとSPの詳細は、6.3.3 節で定義している）。一つのデバイスに対して SP を複数登録すれば、一つのデバイスを複数のGPに準拠させることも可能である。

提案手法を評価するために、パフォーマンス、ユーザビリティ、デバイスプロファイルの記述性、セキュリティ、スケーラビリティについてそれぞれ評価を行った。その結果、提案手法はプラクティカルであり、デバイスやアプリケーションを相互接続できるための能力があることが示せた。

本章の残りの部分は以下のように構成される。4.3節では、IoT-Aggregatorについて説明を行う。4.4節では本研究での貢献であるデバイス協調フレームワークについて説明を行う。4.5節では、提案手法の評価について整理し、4.6節で本研究のまとめを行う。

6.2. IoT-Aggregator

本章では、TRONプロジェクトにおけるIoT協調動作のための中心的な要素であるIoT-Aggregatorについて説明する。

6.2.1. 設計の目標

TRON プロジェクトは、1980 年代から超機能分散システム（Highly Functional Distributed System: HFDS）の実現を最終目的としている [3]。超機能分散システムとは、ネットワークで相互接続された部品同士が協調動作することによって、サービスが単体で動作するよりもサービスを実環境において実現するためのシステムである。マイクロコンピューターによって制御されるネットワークに接続されたモノのことをインテリジェントオブジェクトと呼ぶ。半導体技術や無線通信技術、電池技術等の進展によって、このインテリジェントオブジェクトが近年現実化されている。本研究ではアグリゲートコンピューティングという言葉を、どのようにモノ同士が通信するかではなく、どのようにモノ同士が協調動作するのかについて着目している。

アグリゲートコンピューティングの基本的なコンセプトは、ネットワークで接続されたすべてのモノを総体的に組み合わせることによって、最適なサービスを実現することである。このネットワークで接続されているすべてのモノとは、すべてのセンサーやアクチュエーターやその他すべてのデバイスやクラウドコンピューティングシステムと、これらのシステムから大量のデータを集めて、処理をして、そしてどのようにモノを制御するかを決定するサービスを指す。図 6.2 は、アグリゲートコンピューティングの利用者からの見え方を示す。現在の通常の住宅では、住人が部屋の温度を変えたい場合には、住人は部屋に設置された暖房装置を個別に制御する必要がある。例えば部屋の温度を 25℃にしたいと考えた場合、部屋に設置されたエアコンの設定温度を 25℃にするだけではなく、窓を閉めたり、サーキュレーターの電源を ON/OFF したりするなど、様々な方法が考えられる。アグリゲートコンピューティングでは、総体として環境を最適化することを目指しているため、仮想的なデバイスである複合デバイスと呼ばれる仕組みを取り入れている。複合デバイスは、複数のインテリジェントオブジェクトからなるグループであり、このグループ単位で要求を受け付けることができる。この方法で、アグリゲートコンピューティングシステムとのインタフェースを実装でき、ユーザーやアプリケーションに対してインタフェースを提供できる。

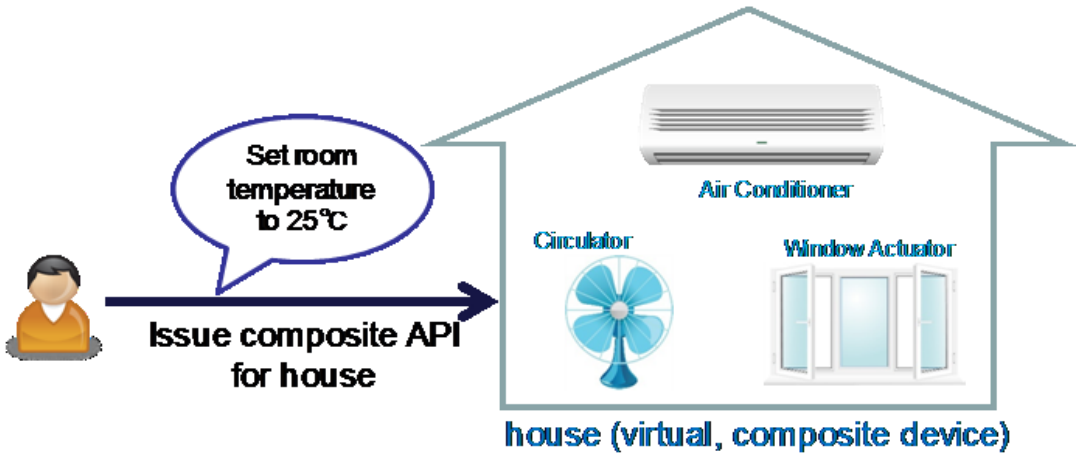


図 6.2 アグリゲートコンピューティングモデルに基づく空調制御の例

このようなアグリゲートコンピューティングのパラダイムを現実の世界で実現するためには、オープン性が重要である。オープン性がなければ、6.1 節で説明したように、アグリゲートコンピューティングは部分最適化のシステムで、クローズな IoT のかけらとなってしまう。IoT-Aggregator に準拠する Colapi を設計する上で、このオープン性を、デバイスのオープン性とアプリケーションのオープン性の 2 つの観点で着目した。

デバイスのオープン性は、異なるメーカーのデバイスが IoT-Aggregator を通して連携することによって実現できる。本研究の後半で、この詳細について述べる。

他方で、アプリケーションのオープン性とは、ある製品やサービスの開発者とは異なるサードパーティーが、その製品やサービスを活用して、新たなアプリケーションやサービスを構築できることを指す。このサードパーティーには、サードパーティーのソフトウェア開発者やサービスの製造者や、個人や、非営利団体などを含む。アプリケーションの開発者は、ユーザーのすべての要求を満たすことができるような機能を、デバイスのメーカーが公式のアプリケーションだけで実現することは非常に難しいことから、このアプリケーションのオープン性は非常に重要である。例えば、様々な障がいのある方へ個別の障がいに合わせてアプリケーションを作り込むことは、公式のアプリケーションで実現することは非常に難しい。目の見えない方へは音声ベースや触覚ベースのユーザーインタフェースが必要になったり、聴覚に障がいのある方へはまた異なるインタフェースが必要になったりする。IoT デバイスをサードパーティーにオープンにしておくことで、このような様々なニーズ

を満たすようなアプリケーションが個別に構築される可能性がある。図 6.3 は、このアプリケーションのオープン性に関する概念図である。ユーザーは自分自身のニーズや嗜好に合わせて、ある目的を達成するためのサービスを自由に選択できる。アプリケーションのオープン性を実現することは、日本の携帯電話がフィーチャーフォンからスマートフォンに移行したようなパラダイムシフトと同様であると言える。スマートフォンの実現する機能は、スマートフォンに新たなアプリケーションをダウンロードしてインストールすることで追加・削除可能である。この IoT-Aggregator で目指すゴールは、このスマートフォンのモデルのように、環境中に設置されたすべてのモノを自由に協調動作させるための IoT のアプリケーションストアを構築することである。

図 6.3 アプリケーションのオープン性

6.2.2. アーキテクチャの概要

ここまでで記述した目的を達成するために、インターネットに接続された様々なデバイスやサービスなどが協調動作するために IoT-Aggregator という概念が存在する。

図 6.4 は IoT-Aggregator の全体構成を示している。IoT デバイスのメーカーが構築した既存のエコシステムも利用できるようにするために、基本的に IoT デバイスは IoT-Aggregator に直接接続されるというよりも、IoT デバイスのメーカーが構築しているクラウド (Manufacturer clouds) に接続されるようにデザインされている。

Manufacturer clouds は IoT-Aggregator の提供する API を利用して IoT-Aggregator に接続されて、他のデバイス等と協調動作を行う。アプリケーションの開発者であれば、アプリケーションのクラウドシステム（Application clouds）を Manufacturer clouds ののように接続することも可能である。このような概念は、図 6.3 に示したようなオープン性を実現するために必要である。

このような IoT-Aggregator を中心としたスタートポロジーに基づき、IoT-Aggregator は Manufacturer clouds と Application clouds などを統合する機能を提供する。例えば、スマートハウス内に設置された IoT デバイスを制御するアプリケーションをユーザーが利用する場合には、アプリケーションは IoT-Aggregator に対してリクエストをすると、IoT-Aggregator がスマートハウス内の IoT デバイスに対してアクセスを行う。このような環境では、IoT デバイスやアプリケーションのメーカーをまたいだマッシュアップが頻繁に起こることが想定される。この際には、IoT デバイスの所有者やアプリケーションのアクセス権限を適切に管理することが重要である。本研究では、IoT-Aggregator を利用してシングルサインオンとアクセスコントロールを行うことによって、Manufacturer clouds や Application clouds は、デバイスに対するアクセスコントロールを自身で実装する必要がなくなり、細かい粒度のアクセスコントロールや利用権限の譲渡などの複雑なアクセスコントロール等も、すべて IoT-Aggregator に任せることが可能となり、個々の IoT デバイスやアプリケーションで実現する必要がなくなる。

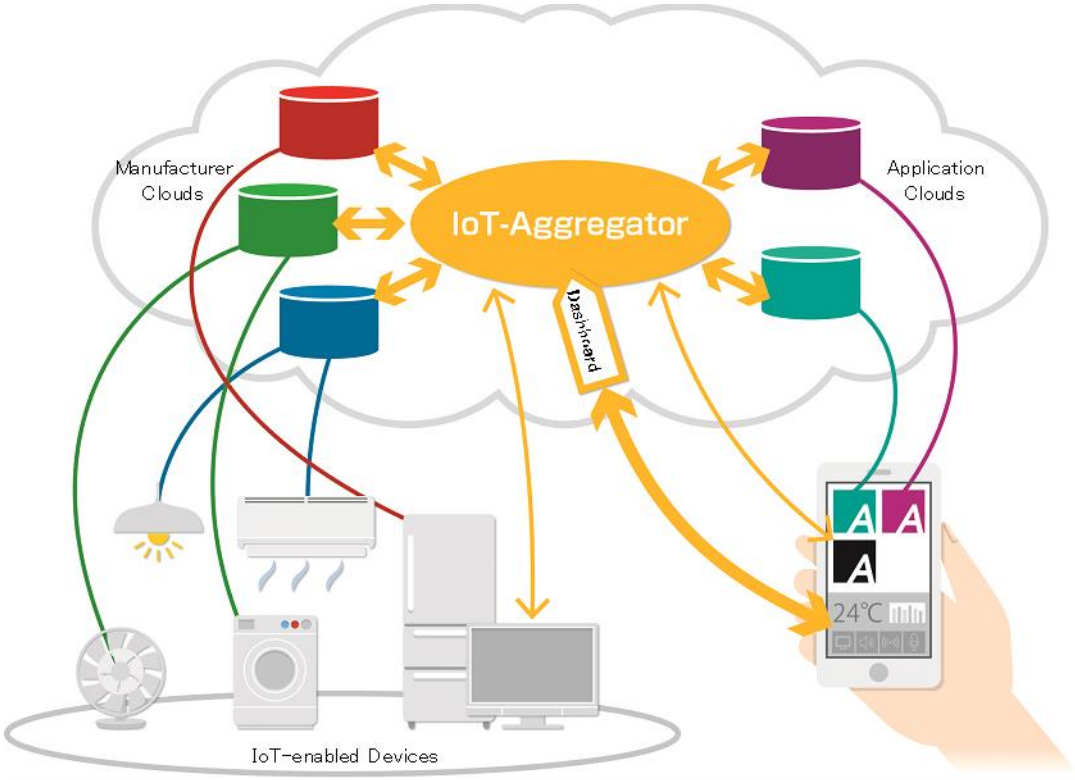


図 6.4 IoT-Aggregator の構成

IoT-Aggregator ダッシュボードは、この IoT-Aggregator に接続した IoT デバイスやアプリケーションなどの情報を一元管理するための標準的なアプリケーションである。IoT-Aggregator ダッシュボードをとおして、デバイスの所有権限の変更やデバイスの追加・削除、アクセス権限の追加、アプリケーションの設定などを行う。このため、アプリケーションランチャーや、アプリケーションストア、デバイス設定を行うためのアプリケーションとも言える。

6.2.3. アクセスコントロール

アグリゲートコンピューティングにおけるアクセスコントロールは、システムに対する様々な関係者がいたり、様々な IoT デバイスやアプリケーションが関与したり、権利があるなど、非常に複雑である。このため、IoT-Aggregator では、IoT デバイスは認証とセキュアな通信のみ対応を行い、複雑なアクセスコントロールの仕組みはクラウド上で提供するモデルとした。このため、IoT-Aggregator を利用すれ

ば IoT デバイスは複雑な機能を実装する必要がなくなり、IoT デバイスに求められる計算リソースも削減できるため、IoT デバイスのコストを削減できる。

IoT-Aggregator で認証と認可の仕組みを実現するために、OpenID Connect 1.0 が採用されている [61]。OpenID Connect 1.0 は、Web サービスの認証の仕組みとして広く使われており、デファクトスタンダード化している手法であるため、Web ベースのインタフェースを提供する IoT-Aggregator に適した手法であると言える。OpenID Connect1.0 に準拠したアプリケーションを Relying party（RP）と呼び、認証情報を管理するシステムを Identity provider（IdP）と呼ぶが、OpenID Connect1.0 は IdP と RP の間のプロトコルを規定している。IoT-Aggregator を実現するためには、IoT-Aggregator を利用したアプリケーション同士やデバイス同士等でお互いが保有する情報を共有する必要性も出てくることが想定されることから、IoT-Aggregator を実現するために OpenID Connect1.0 を独自に拡張して、RP 同士で RP が保有する情報を共有するための仕組みも提案した。なお本研究のスコープからは外れるため、ここでは詳細を省略する。

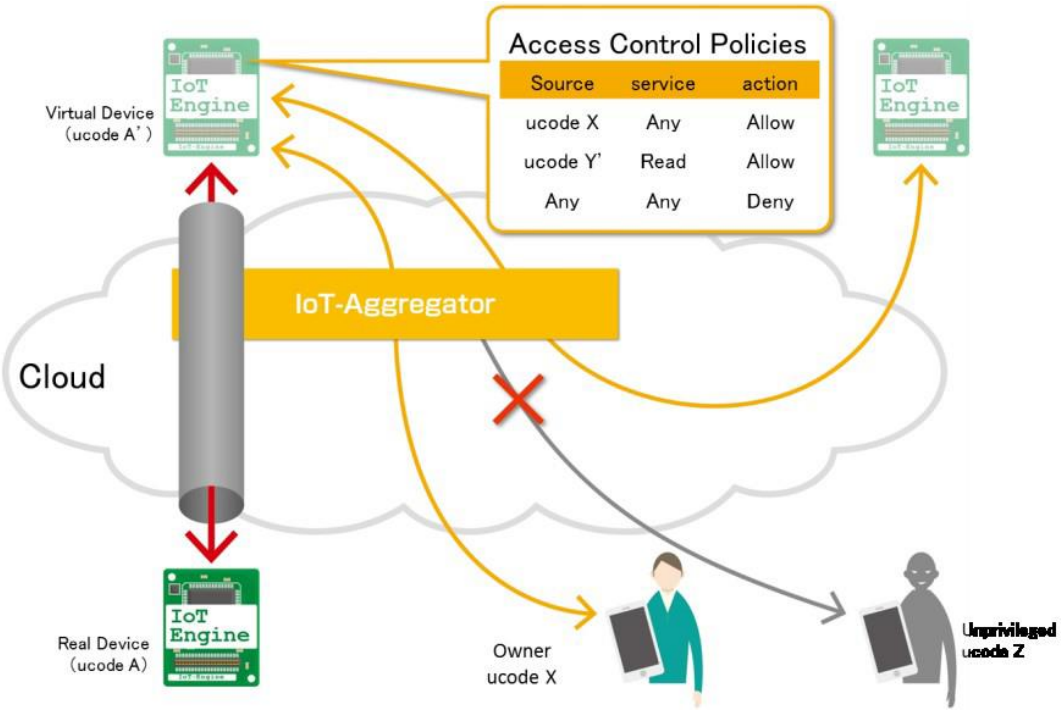


図 6.5 IoT-Aggregator を利用したアクセスコントロール

6.2.4. TRON IoT ダッシュボード

TRON IoT ダッシュボードは、IoT-Aggregator が提供する標準的なアプリケーションである。この TRON IoT ダッシュボードについては、設計の背景と初期設計について述べる。

アプリケーションの設計を行う基礎的な方針として、BTRON で提案された実身仮身モデルを採用する。実身はデータの実体を指し、実身のハイパーリンクを仮身と呼ぶ。仮身をマウスでダブルクリックすると、仮身内に埋め込まれた別のドキュメントを開くことができる（図 6.6）。

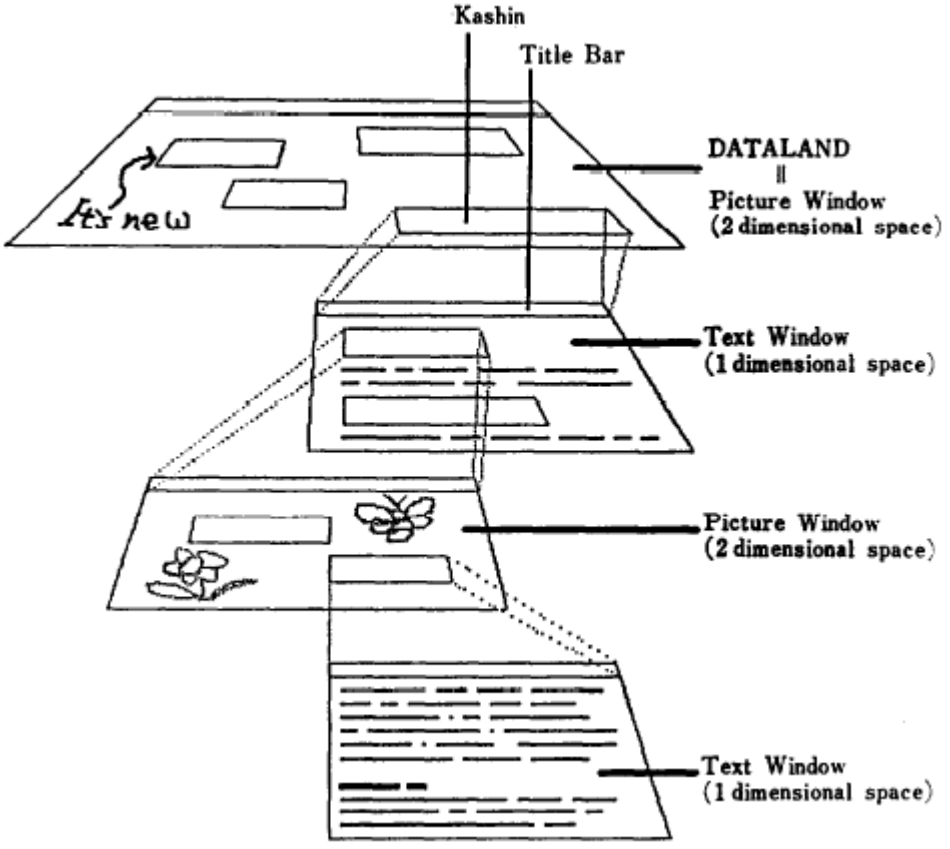


図 6.6 仮身によって実現されるハイパーリンク構造

この考え方に基づき、IoT デバイスの位置を登録のために、および IoT デバイスの情報を確認、変更するために、IoT-Aggregator ではマップ画面上に様々な情報をオーバーレイ表示するように設計した（図 6.7）。IoT デバイスのハイパーリンクをクリックして開かれるウィンドウには、Manufacture cloud が提供する機能へのリンクが一覧表示される。IoT-Aggregator に接続する IoT デバイスやアプリケーションによって異なる機能が、このように単一のインタフェースでユーザーから検索・利用が可能となるのが、この TRON IoT ダッシュボードの特徴である。

さらにこのマップ表示画面に加えて、デバイスやアプリケーションにとって最適になるように、それぞれで異なるビューを追加する方法も提供を検討している。例えば、エンドユーザーが自分自身で使いやすいインタフェースを実装するために Node-RED [62]と呼ばれる GUI プログラミング言語を利用して、自分の好きな IoT デバイスを自分の好きな位置に自由にレイアウトにすることなどを可能とする。

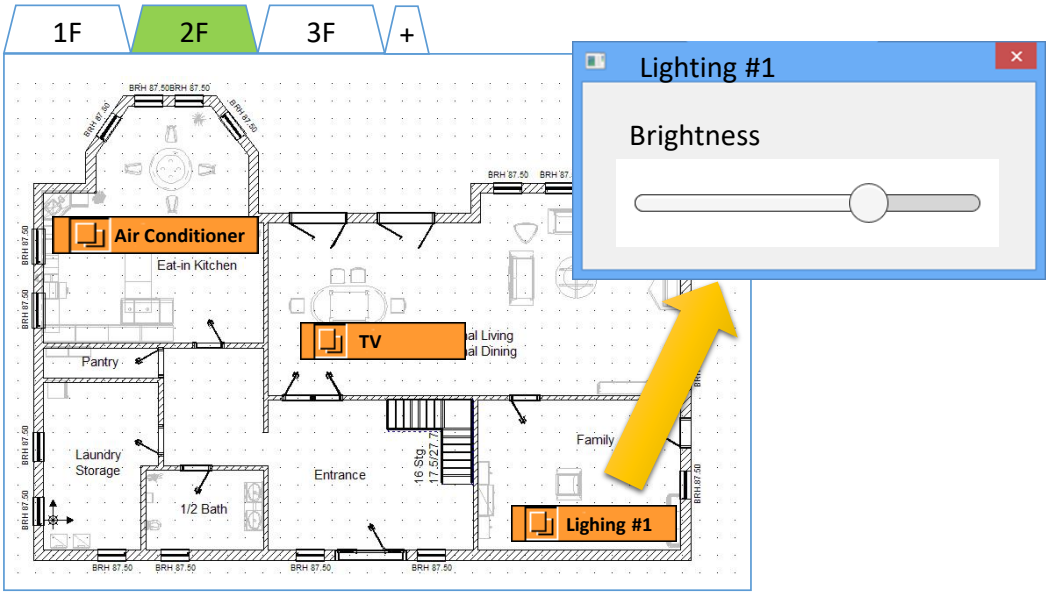


図 6.7 ハイパーリンク化されたデバイス表示用マップ画面

6.2.5. ユースケース

現在、様々なプロジェクトで IoT-Aggregator のコンポーネントを利用している。2015 年に取り組んだ台湾の花蓮におけるスマートハウスプロジェクトでは、デバイスを管理するために、デバイス協調フレームワークのプロトタイプバージョンを採用した。IoT-Aggregator の機構を採用することで、スマートハウス内に設置した IoT デバイスには、デバイスメーカー独自の API を採用したものも存在したが、スマートハウス内の制御用アプリケーションを開発するために必要なコストを削減することができた。

OPaaS.io は、ユーザーの許諾に基づいて複数のサービス間でユーザー属性情報を共有することで、ユーザー個々の嗜好に合わせたサービスのカスタマイズができることを目指した現在進行中のプロジェクトである。例えば、ホテルのコンシェルジュが OPaaS.io から取得した宿泊客の嗜好に基づいてレストランをレコメンドした場合、宿泊客がタクシーで移動しようとすると、タクシードライバーはこの情報を OPaaS.io から受け取れば、宿泊客が自ら行き先をタクシードライバーに告げなくとも目的地を伝えることができる。

6.3. 提案手法

本節では、本研究の主な貢献である IoT-Aggregator に基づくデバイス連携システム Colapi について説明する。

6.3.1. アーキテクチャへの要求

ここまでで議論したように、アグリゲートコンピューティングの考え方に基づいてスマート環境の実用化へ向けた障害を乗り越えることが本フレームワークの目標である。本小節では、それぞれの目標を達成するために必要な要件を定義する。

6.3.1.1. どのようなタイプのデバイスに対しても標準化された API が定義できること

業界団体が策定する API の標準仕様は、基本的に特定の種類のデバイスやサービスを対象としている。例えば DLNA ガイドライン [63]では、主にオーディオ・ビジュアル機器を対象に設計されており、その他の家電製品向けの詳細な仕様は策定されていない。Bluetooth プロファイルや USB のクラス構造 [64] [65]は、共通な仕様に準拠した様々なデバイスを接続するために設計されており、共通の仕様として定義されていない IoT デバイスのメーカーが独自に設計したクラスには対応できない。IoT デバイスのメーカーは、他社に対抗するために、自社の IoT デバイスに付加価値をつけたいと考える場合が多いため、アグリゲートコンピューティングの規格に沿ってメーカー独自の API も扱えるようにする必要がある。

6.3.1.2. エコシステムに準拠することで最小のコストで最大の効果を得ること

IoT デバイスのメーカーの視点から考えると、IoT-Aggregator エコシステムに参加することは費用対効果の観点から妥当である。言い換えると、プラットフォームは IoT デバイスのメーカーが少ない労力で IoT-Aggregator に接続して連携できるようになっていることが必要である。従来は標準仕様にそろえるためには、過去に作成したすべてのデバイスやクラウドを修正する必要があったが、IoT デバイスメーカーが本プラットフォームに接続する場合は、接続に必要なデバイスの記述を少し追加するだけで、既存のメーカー独自のクラウドや IoT デバイスをそのまま無駄にすることなく活用できる。

6.3.2. 設計の概要

ここまでで説明した要求に基づいて、IoT-Aggregator の概念に基づいた IoT デバイス協調フレームワークを提案した。本フレームワークは、IoT デバイスのメーカー独自の API を標準化された API に変換する仕組みを図 6.9 のように提供する。例えば、複数のメーカーのエアコンが部屋に設置されていて、それぞれのエアコンの API が異なる場合でも、提案フレームワークがプロキシとして、標準化された API 形式からそれぞれのエアコンの API に変換することで、アプリケーションの開発者は標準化された API だけを使ってアプリケーションを構築できる。さらに、実用上必須となる IoT デバイス利用時の認証やアクセスコントロールなどもこの提案フレームワークで実現する。この認証やアクセスコントロールなどの複雑な仕組みを IoT デバイス自身で実現するのではなくクラウド上で実現をするのは、アグリゲートコンピューティングの考え方に基づいている。IoT デバイスのデバイス固有の API を標準化された API に変換するために、デバイスプロファイルという API の変換ルールをこの提案フレームワーク内に格納する。このデバイスプロファイルは機械可読な形式で記述されており、デバイス毎に登録されている。

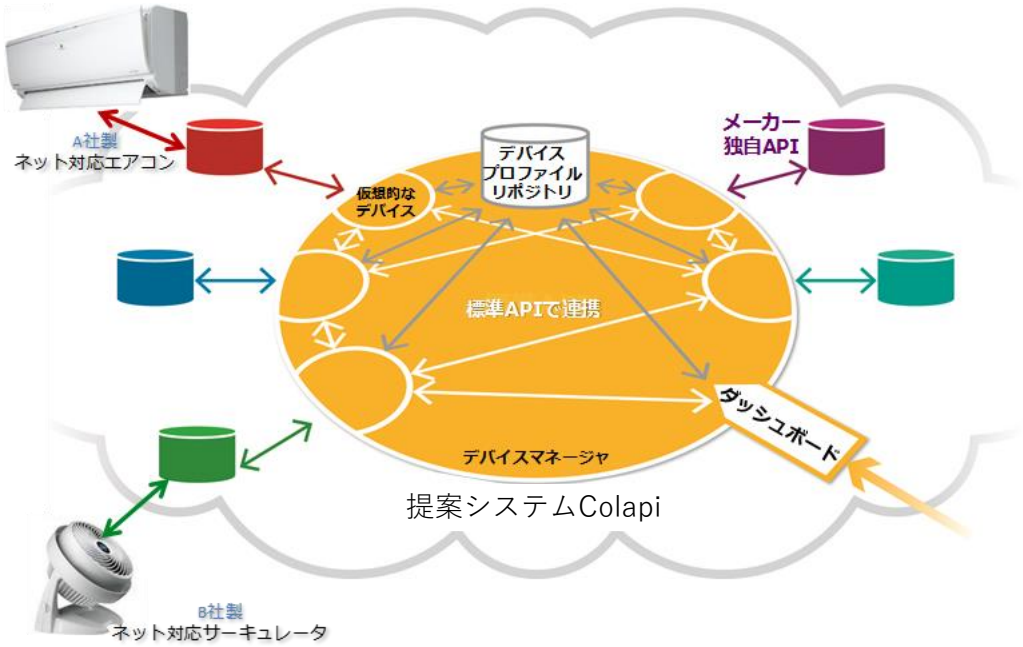


図 6.8 提案システム Colapi と外部システムとの関係

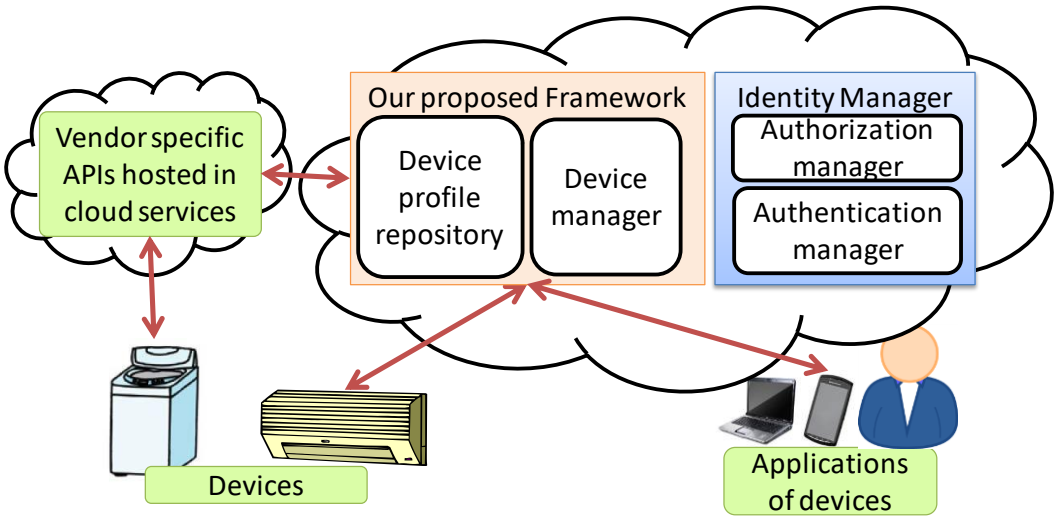


図 6.9 IoT-Aggregator に基づくデバイス連携システム Colapi

図 6.9 に示すように、本提案システム Colapi は、デバイスプロファイルレポジトリとデバイスマネージャという二つの主要なコンポーネントから構成されている。デバイスプロファイルレポジトリは、機械可読な形式のデバイスプロファイルを管理する。デバイスマネージャは標準化された API によるリクエストを受け付けて、メーカーが独自に定義している IoT デバイスやクラウドサービスの API 形式へ変換する。これら二つのコンポーネントに加えて、IoT デバイスに対するアクセス権限をユーザーの指定した特定のサービスやアプリケーションにのみ許可するための、IoT-Aggregator の提供する認証とアクセスコントロールの仕組みを取り入れている。

6.3.3. デバイスプロファイル

デバイスプロファイルの設計にあたり、デバイスメーカーが独自機能を追加するなど、標準化された仕様に従わない場合があることに対して注意を払った。これは主に、メーカーが自社製品に独自の機能をつけることで、他社の製品と差別化をすることに起因している。このような製品の差別化は、製品を販売するためには重要な活動であるが、これは機能に制約を加えるような標準化を行ったとしても、メーカーがそれに従うことは難しいということを表現していると言える。

このような見解に基づいて、提案手法ではデバイスプロファイルを Generic device profile（GP）と Device specific profile（SP）の 2 つに分割した。この 2 つのプロフ

ァイルはいずれも OpenAPI specification 2.0 [66]に、このフレームワークに必要な記述を追加した方式で記述を行う。GP は標準的な API セットを持つデバイスクラスを定義する。例えば、エアコンデバイスクラスは気温を設定するための GET インタフェースと PUT インタフェースを持ち、さらに動作モードの設定やタイマーの設定なども持つ。例えばエアコンに空気清浄機能をつけるためにメーカー独自の API を追加する場合には、エアコンクラスの GP に加えて新たに空気清浄機クラスの GP をこのエアコンに関連づけるか、新たに空気清浄機能付きエアコンのクラスを定義して、この GP をデバイスプロファイルレジストリに登録すれば良い。この手続きによって、新たに登録した API が標準 API として本フレームワーク内で扱われることになる。本フレームワークでは、デバイスは複数のデバイスクラスに関連づけることができるため、この例のように、メーカー独自機能を付加したエアコンに対して、エアコンクラスと、空気清浄クラスの両方を関連づけることができる。このアプローチは、デバイスのメーカーの要望に応じて標準 API を追加することが可能である。GP 内に記述するべきパラメーターは、表 6.1 に示すとおりである。また、照明について記述した GP の主要部分の抜粋を図 6.10 に示す。この例では、/brightness という REST API のエンドポイントを定義しており、この API 経由で照明の明るさを取得、制御することができる。

表 6.1 GP のパラメーターリスト

パラメーター名	概要
id	GP を表す ID を ucode で付与する。
name	GP を表す名前である。半角英数字とハイフン、アンダースコアを組み合わせて記述する。
type	GP の記述対象に応じて、物理的な量や状態を取得するのであれば"Sensor", 物理的な量や状態を制御するものであれば"Actuator"、Sensor や Actuator の設定をするのであれば"Configurator"を選択する 。
title	自然言語で記述した GP の名前である。
informativeText	GP を利用する開発者向けに、利用方法や注意点などを自然言語でプロファイルの説明を記述する。
version	GP のバージョンを指定する。

パラメーター名	概要
api	本研究で拡張した Open API Specification に基づき、API の仕様を記述する。

```
[{ "id": "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX", "name": "Light",
    "type": "Actuator", "title": "Light",
  "informativeText":
    {"description": "Generic Device Profile of Light" },
  "version": 1.0,
  "api": {"swagger": "2.0", "info":
    {"description": "Light API", "version": "1.0.0",
     "title": "Light API" },
    "basePath": "/",
    "schemes": ["http","https"],
    "produces": ["application/json" ],
    "paths": { "/brightness": {
        "put": {
          "summary": "Update brightness of a light",
          "parameters": [
            { "name": "id", "in": "query",
              "description": "ID of a light",
              "required": false, "type": "string" } ],
          "responses": {
            "200": {"description": "Success",
              "schema": { "$ref": "#/definitions/Brightness"}},
            "default": {
              "description": "Unexpected error",
              "schema": {"$ref": "#/definitions/Error" }}},
          "x-api-type": ["Actuator"]},
```

図 6.10 照明 API を記述した GP の主要部分の抜粋

一方 SP は、IoT デバイス固有の API を、GP 内で定義された標準デバイスクラスの API へ変換するためのルールを定義している。IoT デバイスのメーカーは、自社の IoT デバイスのモデル毎に SP を一つ定義する。SP は、GP で定義されているパラメーターに加えて、表 6.2 に示すパラメーターを定義する。API 変換ルールを記述するために、REST API の仕様を機械可読な形式で記述できる Open API Specification を元に、API の変換を実現するための記述を付加するために表 6.3 に示すような拡張を本研究で独自に加えた。もし x-access-type の値が「eval」の場合は、デバイスマネージャは、アプリケーションから受け取ったリクエストをデバイス固有の API 呼び出し形式に変換する。変換には、プロファイルに記述されている x-access-method、x-location、x-request-body、x-response-body 内に、プログラミング言語 Ruby を利用して記述されたスクリプトを利用する。変換する対象となるデータはそれぞれ、HTTP メソッド、URL、リクエストボディ、レスポンスボディである。図 6.11 は、Philips Hue について記述した SP のサンプルの抜粋である。標準 API からデバイス固有の API への変換ルールは、汎用プログラミング言語で定義されているため、シンプルでありながら記述力が高い。

表 6.2 SP 内のみで利用するパラメーター

パラメーター名	概要
modelNumber	デバイスのモデルを一意に識別する ucode。
manufacturer	デバイスのメーカーを一意に識別する ucode。

表 6.3 Open API Specification の本研究における独自拡張

パラメーター名	概要
x-api-type	API の呼び出しが、物理的な運動を引き起こすものは Actuator, 物理的な値を読み取るものは Sensor, IoT デバイスの設定を行うものは Configurator を設定する。
x-location	API の URL を指定する
x-access-type	デバイスが標準的な API を提供している場合は redirect を、それ以外の場合は eval を指定する。
x-access-method	HTTP のメソッドを以下から指定するか、HTTP のメソッド名を返却するスクリプトを記述する。GET, POST, PUT, PATCH, DELETE, OPTION
x-request-body	HTTP の request body について、標準 API からデバイス固有の API への変換スクリプトを記述する。
x-response-body	HTTP の response body について、デバイス固有の API から標準 API への変換スクリプトを記述する

```
"api": { "paths": { "/power": { "put": {
 "parameters": [
   {"name": "id", "in": "query", "required": true,
    "type": "string"},
   {"name": "power", "in": "body", "required": true,
    "schema": {"type": "boolean"}}],
 "x-type": ["Actuator"],
 "x-access-type": "eval",
 "x-access-method": "put",
 "x-location": /api/newdeveloper/lights/{id}/state",
 "x-request-body": {¥"on¥":device_request[¥"power¥"]}",
 "x-response-body":
   "{¥"power¥":#{JSON.parse(device_response)[0]
   ['success'].values[0]}}"
```

図 6.11 Philips Hue の API を記述した SP の主要な部分の抜粋

なおデバイスプロファイルの記述内で記載されている ucode(ITU-T H.642.1)[67]は、IoT デバイスや IoT デバイスのモデルやプロファイルなどを一意にするために利用している。識別子として ucode を採用した理由として、一度利用した識別子が他のシステムも含めて再利用されないことが保証されていることから、提案手法を他のシステムなどと協調動作される際に識別子の表す対象が正しく一意に特定できることが挙げられる。

また図 6.12 には、仮に Colapi の API に準拠した電気錠システムがあった場合に想定される SP の例を記述している。/api/smartlock という API に GET でアクセスをすると電気錠がロックされている（true）かアンロックされている（false）かを取得できる。また、PUT でアクセスをすると、パラメーターによって電気錠をロック/アンロックできるという想定で記述をしている。

```
"api": {  "paths": { "/lock": {
  "get": { "summary": "鍵のLock/Unlock状態取得", "x-type": [ "Actuator" ],
        "x-access-type": "redirect", "x-access-method": "get",
        "x-location": "/api/smartlock",
        "responses": { "200": { "description": "鍵のLock/Unlock状態を取得できた",
              "schema": { "$ref": "#/definitions/Lock" } },
            "default": { "description": "Unexpected error",
              "schema": { "$ref": "#/definitions/Error" } } } },
  "put": { "summary": "鍵のLock/Unlock制御",  "parameters": [
     { "name": "lock", "in": "body", "required": true,
        "schema": { "description": "The status of the lock. Lock: true, Unlock: false",
              "type": "boolean" } } ], "x-type": [ "Actuator" ], "x-access-type": "redirect",
              "x-access-method": "put", "x-location": "/api/smartlock/state",
         "responses": { "200": { "description": "鍵のLock/Unlock状態を制御できた",
            "schema": { "$ref": "#/definitions/Lock" } },
          "default": { "description": "Unexpected error", "schema": { "$ref": "#/definitions/Error" } } }
```

図 6.12 Colapi 準拠電気錠の SP の記述例

6.3.4. デバイスマネージャ API

Colapi 内のデバイスマネージャは、API の変換を行う Proxy として動作し、表 6.4 に示す API を提供する。最も重要な API は、表 6.4 の最後に掲載した **/{:gp_name}/{:id}/*** というパスの API である。パス中の**:gp_name** は、GP で定義されているデバイスクラスの名前を指定する。パス中の**:id** は、特定の IoT デバイスを世界中で一意に特定するために付与されている IoT デバイスの ucode を指定する。再度、空気清浄機能付きエアコンをサンプルに説明する。空気清浄機能付きエアコンは、エアコンクラスと、空気清浄機クラスと、ON/OFF クラスをサポートすると仮定する。本フレームワークを利用するプログラマーは、GP で定義されている以下の API を利用可能である。

- /air_conditioner/{:id}/power
- /air_conditioner/{:id}/mode
- /air_conditioner/{:id}/temperature
- /air_purifier/{:id}/status
- /on_off/{:id}/power

この例から分かるように、GP の名前を接頭辞とする API の命名規則にすることで、一つの IoT デバイスの API に複数の標準 API を同時に対応させることを自然な形で対応させることができる。そしてこのような API の命名規則にしたことによって、変更できない標準化に基づいて機能追加や変更などができないという従来のよ

うな制約は存在せず、IoT デバイスのメーカーは柔軟に標準化された API を組み合わせることが可能となる。

表 6.4 デバイスマネージャの提供する API

API のパス	HTTP メソッド	API の説明
/devices	GET	デバイスマネージャに登録された IoT デバイスの情報を一覧表示する。
/devices	POST	新たな IoT デバイスをデバイスマネージャに登録する。
/devices/{:id}	GET	パラメーターid で指定した IoT デバイスに関する情報のみを取得する。
/devices/{:id}	PUT	パラメーターid で指定した IoT デバイスに関する情報を更新する。
/devices/{:id}	DELETE	パラメーターid で指定した IoT デバイスに関する情報をデバイスマネージャから削除する。
/devices/{:id}/*	*	IoT デバイスの提供するオリジナルの API を呼び出す。
/{:gp_name}/{:id}/*	*	パラメーターid で指定した IoT デバイスの提供する、パラメーターgp_name で指定した GP 内で定義されている API を呼び出す。

6.3.5. デバイスプロファイルリポジトリ API

表 6.5 に、デバイスプロファイルリポジトリの提供する API の一覧を示す。この API を利用することによって、IoT デバイスのメーカーは GP や SP の検索、追加、変更、削除ができ、さらにプログラマーも必要に応じて GP や SP の一覧を検索したり、それぞれの記述を取得したりすることができる。

表 6.5 デバイスプロファイルリポジトリの提供する API

API のパス	HTTP メソッド	説明
/profiles/generic	GET	デバイスプロファイルリポジトリに登録されている GP の一覧を検索する。
/profiles/generic	POST	デバイスプロファイルリポジトリに新たな GP を登録する。
/profiles/generic/{:id}	GET	パラメーターid で指定された GP を取得する。
/profiles/generic/{:id}	PUT	パラメーターid で指定された GP を、リクエストボディ内で指定された内容で更新する。
/profiles/generic/{:id}	DELETE	パラメーターid で指定された GP をデバイスプロファイルリポジトリから削除する。
/profiles/specific	GET	デバイスプロファイルリポジトリに登録されている SP の一覧を検索する。
/profiles/specific	POST	デバイスプロファイルリポジトリに新たな SP を登録する。
/profiles/specific/{:id}	GET	パラメーターid で指定された SP を取得する。
/profiles/specific/{:id}	PUT	パラメーターid で指定された SP を、リクエストボディ内で指定された内容で更新する。
/profiles/specific/{:id}	DELETE	パラメーターid で指定された SP をデバイスプロファイルリポジトリから削除する。

6.4. 評価

Colapi の有効性を評価するため、オーバーヘッドとレイテンシ、ユーザビリティ、プロファイルの記述性、セキュリティ、スケーラビリティの五つの観点から評価を行った。

6.4.1. 評価方法

ケーススタディとして、照明器具の電源を制御するための GP を定義した。定義

した API は表 6.6 に示す。この照明器具の電源を制御する GP に基づいて、2 種類の異なる照明器具に対する SP を定義した。一つは Philips Hue であり、もう一つは独自に定義したシンプルな照明器具である。独自に定義した照明器具の提供する API を表 6.7 に示す。このケーススタディのために定義した GP と SP の一部は、それぞれ図 6.10 と図 6.11 に示す。

表 6.6 ケーススタディとして定義した照明器具の電源を制御する標準 API

API のパス	HTTP メソッド	説明
/power	GET	照明器具の電源の状態を取得する。
/power	PUT	照明器具の電源を ON/OFF する。

表 6.7 ケーススタディとして定義した独自の照明器具 API

API のパス	HTTP メソッド	説明

/api/v1/light	GET	照明器具の電源の状態を取得する。
		レスポンスは以下の形式となる。result には照明の ON 時に true が、OFF 時に false が格納される。
		{
		“results”: result,
		“counts”: 1,
		“total”: 1
		}

/api/v1/light	POST	照明器具の電源を ON にする。GET メソッドと同じレスポンスが返る。
/api/v1/light	DELETE	照明器具の電源を OFF にする。GET メソッドと同じレスポンスが返る。

まず 2 種類の照明を GP で定義した標準照明 API で制御可能となることを確認するために、下記の cURL コマンドを利用した。その結果、いずれの照明も点灯できることが確認できた。

```
$ curl -X PUT ¥
    -H "Content-type: application/json" ¥
    -d '{"power":true,"id":1}' ¥
       "http://$hostname/api/v1/lights/$id/power"
```

さらに本章以降ではこのサンプルを利用して評価を行い、Colapi の有効性について確認を行った。

6.4.2. オーバーヘッドとレイテンシ

Colapi が実現する標準 API を利用することに起因する IoT デバイスへのオーバーヘッドとレイテンシを評価するため、標準 API のラウンドトリップ時間を計測した。ネットワークに起因するレイテンシの影響と照明の API 実行時間を除外するため、直接照明の API を呼び出した場合のラウンドトリップ時間も計測を行った。実行時間の揺らぎの影響などを除外するため、計測はそれぞれ 100 回行った。また、ネットワークによる影響を減らすため、計測用のサーバーソフトウェアとクライアントソフトウェアは、同一仮想マシン上で実行した。

実験に利用した環境は以下の通りである。

- 仮想マシンホスト
  - CPU
    - Intel Core i5-3230M（2 コア、2.6GHz）
  - メモリ
    - 16GB
  - ストレージ
    - SSD　240GB
  - OS
    - Windows 10
  - 仮想環境
    - Oracle VM VirtualBox
  - ネットワーク接続速度
    - 1Gbps（1000Base-T による有線接続）
- 仮想マシンゲスト
  - CPU
    - 2CPU
  - メモリ
    - 4GB
  - OS
    - Ubuntu 14.04
  - ネットワーク
    - ホストの NIC を NAT で利用

計測の結果、照明の API を直接呼び出した場合のラウンドトリップ時間の平均遅延時間は 7 ミリ秒で、標準偏差は 1 ミリ秒であった。Colapi の標準 API を通して照明を制御した場合の平均遅延時間は、15 ミリ秒で、標準偏差は 5 ミリ秒であった。この結果から、Colapi に起因する平均遅延時間は 8 ミリ秒であり、実用上十分無視できる遅延時間であると言える。

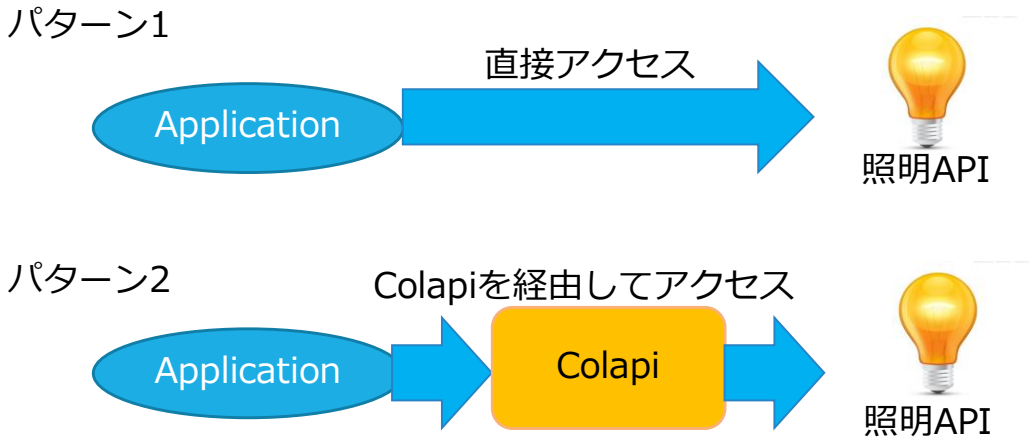


図 6.13 Colapi によるオーバーヘッドの計測パターン

6.4.3. ユーザビリティ

アプリケーション開発者は、様々なメーカーの作成した様々な IoT デバイスに対して Colapi を通してアクセスすることによって、例えそれぞれの IoT デバイスがそれぞれ異なる API を提供していたとしても、GP で定義された標準 API で統一的にアクセスできる。アプリケーション開発者の観点から考えると、それぞれが異なる API を提供する 100 種類の IoT デバイスを利用する場合であったとしても、アプリケーション開発者はそれらをすべて把握する必要はなく、GP で定義された API のみを利用すれば良いため、ユーザビリティの観点からこれは好ましいと言える。

加えて、Colapi は一つの IoT デバイスに対して複数の GP を関連づけることによって、複数の API 群を関連づけて利用できる。このため、アプリケーション開発者は利用したい特定の機能を提供する GP のみ把握すれば良い。例えば、エアコンやテレビや電子レンジなどの IoT デバイスをまとめて電源を切りたい場合には、あらかじめそれぞれの IoT デバイスに「電源 on_off」クラスが紐付けられていれば、ア

プリケーション開発者は「電源 on_off」デバイスクラスにのみ着目すれば良く、個別のエアコンクラスやテレビクラス、電子レンジクラスなどの API を把握する必要はない。

6.4.4. プロファイルの記述性

Colapi では、API 変換ルールをプログラミング言語 Ruby で記述する。Ruby はチューリング完全な言語であるため、ほぼすべての変換ルールを記述できると言える。

しかし、状態を記憶する必要のある API（ステートフルな API）は、記述はできない。今回の実装では、API リクエストを受け付ける度に Ruby で記述した変換ルールを実行するためのセキュアなサンドボックスを新規に作成する。このため、過去に API リクエストを受け付けた際の変数を参照することはできない。過去の状態を参照するようにするためには、サンドボックスの境界を越えて IoT デバイスの状態を共有する方法が考えられるが、これはスケーラビリティの観点とプログラム開発者が状態管理をする作業が生じてプログラムの複雑度が高くなるという観点から現実的ではない。さらに、HTTP の API を設計するために広く利用されているアーキテクチャである RESTful アーキテクチャでも、状態を保存する必要がある処理（ステートフルな処理）は除外するべきであるとされているため、一般的に状態を保存するような処理は対応するべきではないと考えられていると言える。このため、このような状態を記憶する必要のある処理には対応するべきではないと考えている。

6.4.5. セキュリティ

Colapi のセキュリティを担保するためには、TLS による通信経路の暗号化や認証、アクセスコントロール等の一般的な Web サービスのセキュリティ対策として取り入れられている方式を組み合わせることで基本的な部分は対応できる。このため、ここではこのフレームワークに特有のセキュリティホールとなり得る箇所について、議論を行う。

Colapi の設計に基づく脆弱性が考えられる箇所は、SP の中で任意のスクリプトが定義できることが挙げられる。このスクリプトは Ruby で書かれているため、悪意を持ったユーザーが、SP の実行を行うデバイスマネージャのシステムを乗っ取るために、システムファイルを変更するような悪意のあるコードを入力することが考えられる。これを防ぐため、Colapi では安全な関数のみを呼び出せるセキュアな環境で

Ruby スクリプトを評価するように実装した。例えば、Kernel.system や File.open といったシステムレベルに影響を及ぼす関数は実行できない。このように、SP 内に記述されたスクリプトの評価は保護されたセキュアな環境で実行されるように設計されている。

6.4.6. スケーラビリティ

Colapi では、すべての IoT デバイスの API は必ずデバイスマネージャを経由してアクセスするようにしているため、Colapi のユーザーが増えた場合には、デバイスマネージャのスケーラビリティの問題が生じる。デバイスマネージャはシンプルなプロキシとして設計し、状態を保持したり、データを保持したりしないので、スケーラビリティを高めるための並列化も容易である。

デバイスプロファイルリポジトリについては、GP と SP の登録数がシステムのスケーラビリティに関与するため、この点について議論を行う。Colapi では、デバイスプロファイルを保存するためにリレーショナルデータベースを採用している。リレーショナルデータベースでは、適切にインデックスを設定すれば多数のデータがデータベースに登録されても適切な処理時間で登録したデータを得ることができる。しかしデバイスプロファイルリポジトリの API を実現する上では、プロファイルの ID や名前、クラスなどが取得できれば良いため、特にリレーショナルデータベースを採用する必要はない。このため、リレーショナルデータベースでは扱えない規模のプロファイル数を登録する場合には、分散キーバリューストアの採用など、データベースを適切に変更すればスケーラビリティを確保できる。

6.5. まとめ

本章では、スマート環境を実現するためにデバイス連携システム Colapi を提案した。これは IoT-Aggregator の概念を実現するためのシステムであり、IoT デバイスやアプリケーション間の協調動作のために必要な API の相互運用性に関する問題を解決する。この問題を解決するために、標準 API を定義する generic device profile（GP）とデバイス固有の API と GP で定義された標準 API との変換ルールを定義する device specific profile（SP）の 2 種類のデバイスプロファイルを定義した。GP は IoT デバイスのメーカー等が定義でき、一つのデバイスに対して複数の GP を関

連づける事ができる。これによって、たとえ IoT デバイスに実装されている API が 1 種類だけであっても、標準 API と新規機能を表現する新たなクラスに属する標準 API を表現することができる。

さらに、デバイスマネージャとデバイスレポジトリマネージャから構成される Colapi を実装し、パフォーマンス、ユーザビリティ、デバイスプロファイルの記述性、セキュリティとスケーラビリティの観点から評価を行い、アグリゲートコンピューティングを実現する上で求められる IoT デバイスの協調動作のために必要な API の相互運用性に関する問題を解決することができることがわかった。

第 7 章

全体考察と結論

7.1. 全体考察

本研究では、提案した Human-centric アプリケーションと IoT の統合のためのアーキテクチャの有効性を検証するために、実際にシステムを構築して実験を行ってきた。本章では、評価の結果に基づいて本研究の有効性について考察を行う。

7.1.1. ユーザー属性情報連携システム

提案あーｋの Huot を実現するための構成要素のうち、ユーザー属性情報連携システムについて述べた。いくつかの実用的なシチュエーションを想定してユーザー属性情報を定義し、ユーザー属性情報連携システムを実装し、API の応答時間、表記変換ルール、セキュリティの観点から評価を行い、実用性を示した。

7.1.2. IoT デバイス連携システム Colapi

提案アーキテクチャ Huot を実現するための構成要素のうち、IoT デバイス連携に関するシステムを実装している。IoT デバイス連携システム Colapi では、様々な観点から評価を行い、IoT デバイスプラットフォームとしての有効性を検証した。この結果、IoT デバイスプラットフォームとして必要な機能を備え、プロファイルの記述性もあり、レスポンスタイムなどの観点から検討しても実用的であるというこ

とを示せた。

本研究ではアクセスコントロールについて、OpenID connect 1.0 に基づく手法で実現することを提案している。OpenID connect 1.0 は、誰がどのデータにアクセスできるかをコントロールするための手法であるため、この手法を利用することで、誰がデバイスにアクセス可能かについて適切にコントロールできる。

ユーザー属性情報との組み合わせについては、提案アーキテクチャ Huot では、ユーザー属性情報の提供も、IoT デバイスの操作を行う API も、どちらも共通の仕組みで API を提供しているため、アプリケーションを作成する開発者は、IoT デバイスとユーザー属性情報を相互に連携させたサービスを、この Huot の提供する API を利用することで構築できると言える。さらに、Huot の提供する API を活用すれば、ほかの様々な情報や Web サービス等と自由に組み合わせたサービスをアプリケーション開発者が構築することできる。また、Huot の提供する IoT デバイスの連携システムは IoT-Aggregator の仕組みに沿って構築していることから、IoT-Aggregator の提供するクラウドベースの実身／仮身モデルに基づくシステム開発なども可能であり、柔軟性・拡張性が高いプラットフォームになっていると言える。

7.2. まとめ

7.2.1. 関連研究

まず IoT デバイスプラットフォームについて定義を行った。IoT プラットフォームとは、IoT 化された様々なモノを管理し、情報取得をしたりコントロールしたりするためのプラットフォームである。IoT プラットフォームは、IoT デバイスをプログラムから制御することが容易であるために API を提供していることが必ず求められる。

その後、既存の IoT デバイスプラットフォームについて調査を行い、いずれの手法も本研究の目指す仕組みを実現できていないということがわかった。

また、IoT デバイスプラットフォームでは様々な情報と組み合わせができるべきであることから、本研究では人間の位置情報を例として取り上げた。位置情報を推測するための測位手法については、インフラストラクチャー型測位手法、デッドレコニング型測位手法、およびそれらを組み合わせたハイブリッド型測位手法の 3 種類についてそれぞれ特徴を説明した。この結果、提案するプラットフォームでは位置だけではなく、位置を抽象化した場所という概念が扱えるべきである。また、上述のように様々な手法によって位置や場所が推測されることから、それぞれの手法の信頼度情報も含めて取り扱えるように考慮するべきであると言える。

7.2.2. ユビキタス ID アーキテクチャ

本提案フアーキテクチャがベースとしているユビキタス ID アーキテクチャについて紹介を行った。ユビキタス ID アーキテクチャは、ucode、ucR 等から構成されており、さらにその応用として、IoT デバイスの情報を管理するためのモデルアーキテクチャである IoT-Aggregator と個人の属性情報を管理するための OPaaS.io が存在するため、これらの情報を整理した。

7.2.3. Human-centric アプリケーションと IoT の統合アーキテクチャ

本研究の提案アーキテクチャである、Human-centric アプリケーションと IoT の統合アーキテクチャについて説明を行った。また、この実現の方針として、OPaaS.io に相当するユーザー属性情報連携システムと、IoT デバイス連携システム Colapi の

2 種類のシステムから構築する方針であることを説明した。

7.2.4. ユーザー属性情報連携システム

本章では、提案アーキテクチャの Huot を実現するための構成要素のうち、ユーザー属性情報連携システムについて述べた。いくつかの実用的なシチュエーションを想定してユーザー属性情報を定義し、ユーザー属性情報連携システムを実装し、API の応答時間、表記変換ルール、セキュリティの観点から評価を行い、実用性を示した。博士論文執筆段階で、実際に約 6 種類程度のサービスで、150 名以上のユーザーに実証実験として利用していただいている状況である。

さらにユーザー属性情報を取得する例として、デッドレコニングによる位置情報を取得する手法について説明した。デッドレコニングは位置情報だけではなく、歩行者の行動も同時に取得できることから、本章で紹介したようなユーザー属性情報のサンプルとした。Bluetooth のような狭帯域かつ低消費電力の無線通信方式で接続された携帯端末上で歩行者の位置情報を利用したサービスを実現するための新たな PDU を提案した。提案 PDU の特徴として、携帯端末上でデッドレコニングによる歩行者の測位を行うために必要な情報を PDU 上で抽象化してから携帯端末へ送信することによって、PDU と携帯端末間で通信するメッセージのサイズをコンパクトにしたことが挙げられる。評価として、従来のセンサーと比較して提案 PDU が採用しているメッセージサイズはコンパクトになっていることを示した。さらに、提案 PDU と携帯端末を組み合わせて、歩行者の位置を推測する評価システムを構築し、実験を行った。その結果、提案 PDU による低周期でコンパクトなデータを利用しても、他の手法と組み合わせることによって歩行者の位置を推定するシステムを実環境でも測位を現実的な精度で実現できることが示せた。

7.2.5. IoT デバイス連携システム Colapi

第 6 章では、スマート環境を実現するためにデバイス連携システムを提案した。これは IoT-Aggregator の概念を実現するためのシステムであり、IoT デバイスやアプリケーション間の協調動作のために必要な API の相互運用性に関する問題を解決する。この問題を解決するために、標準 API を定義する generic device profile (GP) とデバイス固有の API と GP で定義された標準 API との変換ルールを定義する device specific profile (SP) の 2 種類のデバイスプロファイルを定義した。GP は IoT

デバイスのメーカー等が定義でき、一つのデバイスに対して複数の GP を関連づける事ができる。これによって、たとえ IoT デバイスに実装されている API が 1 種類だけであっても、標準 API と新規機能を表現する新たなクラスに属する標準 API を表現することができる。さらに、デバイスマネージャとデバイスレポジトリマネージャから構成される Colapi を実装し、パフォーマンス、ユーザビリティ、デバイスプロファイルの記述性、セキュリティとスケーラビリティの観点から評価を行い、アグリゲートコンピューティングを実現する上で求められる IoT デバイスの連携動作のために必要な API の相互運用性に関する問題を解決することができることがわかった。

7.3. 結論

本研究では Human-centric アプリケーションと IoT の統合アーキテクチャを提案した。

提案手法は、下記のような成果がある。

- モノと人間からなるサービスを連携させ総体として扱う為には、ユーザーを含む関連プレーヤーの位置や場所の取り扱いが重要であるため、位置情報や場所情報を取り扱うための仕組みを提供した
- 多くのサービスを接続する際にはアプリケーションの作りやすさの観点から API の標準化が重要であるが、既存のデバイスの API を再設計することは負担が大きいため、クラウドでリバースプロキシを実現し、既存のデバイスに手を加えなくとも標準化された API でモノにアクセス可能な仕組みを提供した

さらに、ユーザー属性情報を連携するためのシステムも構築した。博士論文執筆段階で、実際に約 6 種類程度のサービスで、150 名以上のユーザーに利用していただいている状況である。ユーザー属性情報の例として、デッドレコニングによる位置情報についても紹介を行った。

これらの評価としてパフォーマンス、ユーザビリティ、デバイスプロファイルの記述性、セキュリティとスケーラビリティの観点から評価を行い、それぞれの観点から本提案手法が Human-centric アプリケーションと IoT の統合を実現するためのアーキテクチャとして有用であることを示した。

参考文献

[1] J. A. Stankovic, “Research Directions for the Internet of Things,” *IEEE Internet of Things Journal,* 第 巻 1, 第 1, pp. 3-9, February 2014.

[2] K. Sakamura, “本格始動! IoT-Engine & IoT-Aggregator,” 24 October 2016. [オンライン]. Available: http://www.tron.org/ja/2016/10/post-2209/. [アクセス日: 01 December 2016].

[3] K. Sakamura, “The Objectives of the TRON Project,” *TRON Project 1987 Open-Architecture Computer Systems: Proceedings of the Third TRON Project Symposium,* pp. 3-16, 1987.

[4] 横須賀テレコムリサーチパーク YRP ユビキタス・ネットワーキング研究所, [オンライン]. Available: http://www.ubin.jp/about.html. [アクセス日: 01 December 2016].

[5] トロンフォーラム, 5 4 2016. [オンライン]. Available: http://www.tron.org/ja/2016/04/press20160405/. [アクセス日: 1 12 2016].

[6] MyData Alliance, “MyData,” [オンライン]. Available: https://mydatafi.wordpress.com/. [アクセス日: 01 December 2016].

[7] Department for Business, Innovation & Skills and The Rt Hon Edward Davey, “The midata vision of consumer empowerment,” 03 November 2011. [オンライン]. Available: https://www.gov.uk/government/news/the-midata-vision-of-consumer-empowerment. [アクセス日: 01 December 2016].

[8] Berkman Center for Internet and Society at Harvard University, “Project

VRM,”［オンライン］. Available: https://cyber.harvard.edu/projectvrm/Main_Page.［アクセス日: 01 December 2016］.

[9] 総務省 2020 年に向けた社会全体の ICT 化推進に関する懇談会 都市サービス高度化ワーキンググループ, “おもてなしインフラ報告書（案）,”［オンライン］. Available: http://www.soumu.go.jp/main_content/000406837.pdf.［アクセス日: 01 December 2016］.

［10］R. T. Fielding, “Architectural Styles and the Design of Network-based Software Architectures.,” University of California, Irvine, 2000..

［11］別所正博, “ユビキタス空間識別基盤に関する研究,” 東京大学大学院学際情報学府, 2008.

［12］A. Satoshi, W. Yuki, K. Noboru , S. Ken, “A robust pedestrian dead-reckoning positioning based on pedestrian behavior and sensor validity,” *Proceedings of the 2012 IEEE/ION Position, Location and Navigation Symposium,* pp. 328-333, 2012.

［13］The SOA Manifesto Authors, “SOA Manifesto,”［オンライン］. Available: http://www.soa-manifesto.org/.［アクセス日: 01 December 2016］.

［14］UPnP Forum, “Leveraging UPnP+: the Next Generation of Universal Interoperability,” April 2015.［オンライン］. Available: http://upnp.org/resources/whitepapers/UPnP_Plus_Whitepaper_2015.pdf.

［15］Open Connectivity Foundation, Inc., “OCF - Specifications,” 23 December 2015.［オンライン］. Available: https://openconnectivity.org/resources/specifications.

［16］K. Aberer, M. Hauswirth , A. Salehi, “Global Sensor Networks,” *report LSIR-2006-001,* 2006.

［17］G. Banda, C. K. Bommakanti , H. Mohan, “One IoT: an IoT protocol and framework for OEMs to make IoT-enabled devices forward compatible,” *Reliable Intell Environ,* pp. 1-14, 2016.

［18］J.-P. Calbimonte, S. Sarni, J. Eberle , K. Aberer, “XGSN: An Open-source Semantic Sensing Middleware for the Web of Things,” *7th International SSN Workshop,* 19 October 2014.

[19] D. Pfisterer, K. Romer, D. Bimschas, O. Kleine, R. Mietz, C. Truong, H. Hasemann, A. Kröller, M. Pagel, M. Hauswirth, M. Karnstedt, M. Leggieri, A. Passant , R. Richardson, “SPITFIRE: toward a semantic web of things,” *IEEE Commun Mag.,* 第 巻 49, 第 11, pp. 40-48, November 2011.

[20] A. Sheth, C. Henson , S. S. Sahoo, “Semantic Sensor Web,” *IEEE Internet Comput.,* 第 巻 12, 第 4, pp. 78-83, July-Aug. 2008.

[21] P. M. B. M. E. a. C. V. M. Presser, “The SENSEI project: integrating the physical world with the digital world of the network of the future,” *IEEE Communications Magazine,* 第 巻 vol. 47, 第 no. 4, pp. pp. 1-4, April 2009.

[22] National Coordination Office for Space-Based Positioning, Navigation, and Timing, “GPS.GOV,” [オンライン]. Available: http://www.gps.gov/systems/gps/. [アクセス日: 01 12 2016].

[23] Information and Analysis Center for Positioning, Navigation and Timing, Korolyov, Russia, “Information analytical centre of GLONASS and GPS controlling,” [オンライン]. Available: https://www.glonass-iac.ru/en/. [アクセス日: 01 12 2016].

[24] Cabinet Office, Government Of Japan, “準天頂衛星システム (QZSS) 公式サイト - 内閣府,” [オンライン]. Available: http://qzss.go.jp/. [アクセス日: 01 December 2016].

[25] P. Bahl , V. N. Padmanabhan, “Radar: An in-building rf-based user location and tracking system,” *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE,* 2000.

[26] K. Nakamura , K. Sakamura, “Sub-meter accuracy localization system using self-localized portable UWB anchor nodes,” *Consumer Electronics (GCCE), 2012 IEEE 1st Global Conference on,* 2012.

[27] A. LaMarca, Y. Chawathe, S. Consolvo, J. Hightower, I. Smith, J. Scott, T. Sohn, J. Howard, J. Hughes, F. Potter, J. Tabert, P. Powledge, G. Borriello , B. Schilit, “Place Lab: Device Positioning Using Radio Beacons

in the Wild," *Proceedings of the 3rd International Conference on Pervasive Computing (Pervasive 2005),* 第 巻 3468, 2005.

[28] A. Ward, A. Jones , A. Hopper, "A new location technique for the active," *IEEE Personal Communications,* 第 巻 4, 第 5, pp. 42-47, 1997.

[29] N. B. Priyantha, A. Chakraborty , H. Balakrishnan, "The cricket location-support system," *MobiCom '00: Proceedings of the 6th annual international conference on Mobile computing and networking,* pp. 32-43, 2000.

[30] R. Want, A. Hopper, V. Falcao , J. Gibbons, "The active badge location system," *ACM Transactions on Information Systems (TOIS),* 第 巻 10, 第 1, pp. 91-102, 1992.

[31] S. Jung , W. Woo, "UbiTrack: Infrared-based user Tracking System for indoor environment," *In Proceedings of the International Conference on Artificial Reality and Telexistence (ICAT 2004),* 2004.

[32] N. U. Hassan, A. Naeem, M. A. Pasha, T. Jadoon , C. Yuen, "Indoor Positioning Using Visible LED Lights: A Survey," *ACM Computing Surveys,* pp. 1-20, 2015.

[33] D. L. d. I. na, P. R. S. Mendon¸ca , A. Hopper, "Trip: A low-cost vision-based location system for ubiquitous computing," *Personal Ubiquitous Comput.,* 第 巻 6, 第 3, pp. 206-219, 2002.

[34] J. Rekimoto , Y. Ayatsuka, "CyberCode: designing augmented reality environments with visual tags," *In Proceedings of DARE 2000 on Designing,* pp. 1-10 , 2000.

[35] G. Klein , D. Murray, "Parallel tracking and mapping for small AR workspaces," *In ISMAR '07: Proceedings of the 2007 6th IEEE and ACM International-,* pp. 1-10, 2007.

[36] National Coordination Office for Space-Based Positioning, Navigation, and Timing, "GPS Accuracy," [オンライン]. Available: http://www.gps.gov/systems/gps/performance/accuracy/. [アクセス日: 01 December 2016].

[37] S. He , S.-H. G. Chan, "Wi-Fi Fingerprint-Based Indoor Positioning: Recent Advances and Comparisons," *IEEE Communications Surveys & Tut*

orials, 第 巻 18, 第 1, pp. 466-490, 2015.

[38] L. Fang, P. Antsaklis, L. Montestruque, M. McMickell, M. Lemmon, Y. Sun, H. Fang, I. Koutroulis, M. Haenggi, M. Xie , X. Xie, "Design of a wireless assisted pedestrian dead reckoning system - the navmote experience. Instrumentation and Measurement," *IEEE Transactions on Instrumentation and Measurement,* 第 巻 54, 第 6, pp. 2342-2358, 2005.

[39] C. Fischer, K. Muthukrishnan, M. Hazas , H. Gellersen, "Ultrasound-aided pedestrian dead reckoning for indoor navigation," *In Proceedings of the first ACM international workshop on Mobile entity localization and tracking in GPS-less environments,* pp. 31-36, 2008.

[40] G. Retscher , M. Thienelt, "Navio - a navigation service for pedestrians," *Global Navigation Satellite System,* 第 9, 2006.

[41] R. Stirling , K. Fyfe, "Evaluation of a new method of heading estimation for pedestrian dead reckoning using shoe mounted sensors.," *Journal of navigation,* 第 巻 58, pp. 31-45, 2005.

[42] M. Kourogi, N. Sakata, T. Okuma , T. Kurata, "Indoor/Outdoor Pedestrian Navigation with an Embedded GPS/RFID/Self-contained Sensor System," *Proceeding ICAT'06 Proceedings of the 16th international conference on Advances in Artificial Reality and Tele-Existence,* pp. 1310-1321, 2006.

[43] J. Racko, P. Brida, A. Perttula, J. Parviainen , J. Collin, "Pedestrian Dead Reckoning with Particle Filter for handheld smartphone," *2016 International Conference on Indoor Positioning and Indoor Navigation (IPIN),* pp. 1-7, 2016.

[44] W. Kang , Y. Han, "SmartPDR: Smartphone-Based Pedestrian Dead Reckoning for Indoor Localization," *IEEE Sensors Journal,* 第 巻 15, 第 5, pp. 2906-2916, 2015.

[45] M. Bessho, S. Kobayashi, N. Koshizuka , K. Sakamura, "uNavi :Implementation and Deployment of a Place-Based Pedestrian," *Proceedings of COMPSAC 2008,* pp. 1254-1259, 2008.

[46] M. R. Rahimi, J. Ren, C. H. Liu, A. V. Vasilakos , N. Venkatasubramanian, “Mobile Cloud Computing: A Survey, State of Art and Future Directions,” *Mobile Networks and Applications,* 第 巻 19, 第 2, pp. 133-143, April 2014.

[47] R. Harle, “A Survey of Indoor Inertial Positioning Systems for Pedestrians,” *IEEE Communications Surveys & Tutorials,* 第 巻 15, 第 3, pp. 1281-1293, 09 January 2013.

[48] E. Foxlin, “Pedestrian Tracking with Shoe-Mounted Inertial Sensors,” *IEEE Comp. Graphics and Applications,* 第 巻 25, 第 6, pp. 38-46, 2005.

[49] C. Huang, Z. Liao , L. Zhao, “Synergism of INS and PDR in Self-Contained Pedestrian Tracking With a Miniature Sensor Module,” *IEEE Sensors Journal,* 第 巻 10, 第 8, pp. 1349-1359, 2010 May 2010.

[50] M. Kourogi, T. Kurata , T. Ishikawa, “A Method of Pedestrian Dead Reckoning Using Action Recognition,” *2010 IEEE/ION Position Location and Navigation Symp.,* pp. 85-89, May 2010.

[51] S. Wan , E. Foxlin, “Improved Pedestrian Navigation Based on Drift-Reduced MEMS IMU Chip,” *Proc. 2010 Int. Tech. Meeting of The Inst. of Navigation,* pp. 220-229, January 2010.

[52] Xsens Technologies, “Xsens Technologies,” [オンライン]. Available: www.xsens.com. [アクセス日: 3 August 2012].

[53] Honeywell, [オンライン]. Available: http://www.magneticsensors.com/dead-reckoning-module.php. [アクセス日: 3 Aug. 2012].

[54] T. Judd , T. Vu, “Use of a new pedometric dead reckoning module in GPS denied environments,” *2008 IEEE/ION Position, Location and Navigation Symp.,* pp. pp.120-128, May 2008.

[55] National Marine Electronics Associatio, “NMEA 0183 Standard,” November 2008. [オンライン]. Available: http://www.nmea.org/content/nmea_standards/nmea_0183_v_410.asp. [アクセス日: 01 December 2016].

[56] S. Butterworth, “On the Theory of Filter Amplifiers,” *Experimental Wireless and the Radio Engineer,* 第 巻 7, pp. 536-541, 1930.

[57] H. Weinberg, “Using the ADXL202 in Pedometer and Personal Navigati onApplications,” *Analog Devices AN-602 application Note,* 2002.

[58] D. Gusenbauer, C. Isert , J. Krosche, “Self-Contained Indoor Positioning on Off-The-Shelf Mobile Devices,” *Proc. 2010 Int. Conf. Indoor Positi oning and Indoor Navigation,* 2010.

[59] IoT Technical Community, “Towards a Definition of the Internet of Thi ngs (IoT),” IEEE, 2015.

[60] S. Asano, T. Yashiro , K. Sakamura, “A Proxy Framework for API Inter operability in the Internet of Things,” *IEEE 5th Global Conference on Consumer Electronics (GCCE 2016),* 2016.

[61] The OpenID Foundation, “Specifications & developer information,” [オンライン]. Available: http://openid.net/developers/specs/. [アクセス日: 01 December 2016].

[62] Node-RED, “A visual tool for wireing the Internet-of-Things,” [オンライン]. Available: http://nodered.org. [アクセス日: 01 December 2016].

[63] Digital Living Network Alliance, “DLNA guidelines - June 2016,” [オンライン]. Available: http://www.dlna.org/guidelines/. [アクセス日: 01 12 2016].

[64] Bluetooth Special Interest Group, “adopted specifications,” [オンライン]. Available: https://www.bluetooth.com/specifications/adopted-specifications. [アクセス日: 01 December 2016].

[65] USB Implementers Forum, Inc, “USB class codes - June 15,” [オンライン]. [アクセス日: 2016].

[66] The Linux Foundation, “OpenAPI Specification,” [オンライン]. Available: https://github.com/OAI/OpenAPI-Specification/blob/master/versions/2.0.md. [アクセス日: 01 Dec 2016].

[67] International Telecommunication Union,, “H.642.1: Multimedia informati on access triggered by tag-based identification - identification scheme,” 2012.

[68] X. M. W. T. a. X. Z. Z. Sun, “Activity classification and dead reckoning

for pedestrian navigation with wearable sensors," *Measurement Science and Technology,* 第 巻 vol. 20, 第 no. 1, 2009.

[69] J. Nieminen, T. Savolainen, M. Isomaki, B. Patil, Z. Shelby , C. Gomez, "IPv6 over BLUETOOTH(R) Low Energy," October 2015. [オンライン]. Available: https://tools.ietf.org/html/rfc7668. [アクセス日: 01 December 2016].

[70] K. Sakamura, "BTRON における統一的操作モデルの提案," *IPSJ Journal,* 第 巻 26, 第 11, November 1985.

[71] TRON Forum, "TRON Project," [オンライン]. Available: http://www.tron.org/tron-project/. [アクセス日: 01 December 2016].

[72] 東京都都市整備局, 国土交通省, "平成２４年度「東京ユビキタス計画・銀座」," 7 February 2013. [オンライン]. Available: http://www.toshiseibi.metro.tokyo.jp/topics/h24/topi040.html. [アクセス日: 01 December 2016].

[73] P. Goyal, V. Ribeiro, H. Saran , A. Kumar, "Strap-down Pedestrian Dead-Reckoning system," *2011 Int. Conf. Indoor Positioning and Indoor Navigation (IPIN),* pp. 1-7, 21-23 Sept. 2011.

付録１　ユーザー属性情報連携システムの API の一覧

表 7.1 ユーザー属性情報連携システムの API の一覧

API 種別	API 名称	API	実装の有無	内部処理用	外部公開用
認証・アクセス制御	認証要求受付	GET /oauth2/authorize	○	-	○
	FeliCa 認証	GET or POST /auth/login	○	-	○
	トークン要求受付	GET /oauth2/token	○	○	-
	アカウント二重登録チェック	GET /users	○	○	-
	アカウント新規作成	POST /users	○	○	-
	クレームＪＷＴ要求	POST /outh2/clamjwt	○	○	-
	トークン確認	GET or POST /oauth2/verify	○	-	○
	アクセス権限情報検索	GET /users/{uid}/authorities	○	○	-
	アクセス権限変更	PUT /users/{uid}/authorities	○	○	-
	アクセス権限情報取得	GET /users/{uid}/authorities/{user_attribute}	○	○	-
	ユーザー情報検索	GET /users	○	○	-
	ユーザー情報更新	PUT /users/{uid}	○	○	-
	FeliCa 情報取得	GET /users/{uid}/felica	○	○	-
	FeliCa 登録	PUT /users/{uid}/felica	○	○	-

API 種別	API 名称	API	実装の有無	内部処理用	外部公開用
	パスワード変更	PUT /users/{uid}/password	○	○	-
	アクセス権限設定依頼	POST /permissions/request	○		○
ルール管理	ルール一覧取得	GET /rules	○	○	-
	ルール削除	DELETE /rules/{rules_id}	-	○	-
	ルール取得	GET /rules/{rules_id}	○	○	-
	ルール更新	PUT /rules/{rules_id}	-	○	-
	ルール新規登録	POST /rules	-	○	-
サービスドメイン管理	サービスドメイン一覧取得	GET /service_domains	○	○	-
	サービスドメイン削除	DELETE /service_domains/{service_domains_id}	-	○	-
	サービスドメイン取得	GET /service_domains/{service_domains_id}	-	○	-
	サービスドメイン更新	PUT /service_domains/{service_domains_id}	-	○	-
	サービスドメイン登録	POST /service_domains	-	○	-
サービス管理	サービス一覧取得	GET /services	○	○	-
	サービス削除	DELETE /services/ {services_id}	-	○	-
	サービス取得	GET /services/ {services_id}	-	○	-
	サービス更新	PUT /services/ {services_id}	-	○	-
	サービス登録	POST /services	-	○	-

API 種別	API 名称	API	実装の有無	内部処理用	外部公開用
	サービス検索	GET /services/search	-	○	-
サイネージ管理	サイネージドメイン一覧取得	GET /signage_domains	-	○	-
	サイネージドメイン削除	DELETE /signage_domains/{signage_domains_id}	-	○	-
	サイネージドメイン取得	GET /signage_domains/{signage_domains_id}	-	○	-
	サイネージドメイン更新	PUT /signage_domains/{signage_domains_id}	-	○	-
	サイネージドメイン登録	POST /signage_domains	-	○	-
	サイネージ一覧取得	GET /signages	○	○	-
	サイネージ削除	DELETE /signages/{signages_id}	-	○	-
	サイネージ取得	GET /signages/{signages_id}	-	○	-
	サイネージ更新	PUT /signages/{signages_id}	-	○	-
	サイネージアクセストークン取得	GET /signages/{signages_id}/token	-	○	-
	サイネージアクセストークン発行	POST /signages/{signages_id}/token	-	○	-
	サイネージ登録	POST /signages	-	○	-
ユーザ属性情報	ユーザ属性情報取得	GET /user_attributes	○	-	○
	ユーザ属性情報取得（内部	GET /user_attributes/{original_id}	○	○	-

API 種別	API 名称	API	実 装 の 有無	内部 処理用	外部 公開用
	処理用)				
	ユーザ属性情報登録	PUT /user_attributes/{original_id}	○	○	-
	ユーザ属性情報（画像）登録	PUT /user_attributes/{original_id}/image	○	○	-
	ユーザ属性情報（画像）削除	DELETE /user_attributes/{original_id}/image	○	○	-
	ユーザ属性情報提供履歴取得	GET /user_attributes/offer_history	○	○	-
	ユーザ属性情報変更履歴取得	GET /user_attributes/update_history	○	○	-
	オリジナルＩＤ発行	POST /original_id	○	○	-
	リンクＩＤ取得	GET /link_id	-	○	-
	オリジナルＩＤ取得	GET /original_id	-	○	-