TTC標準
Standard

JF-W3C-wot-architecture-20200409
Web of Things (WoT) アーキテクチャ

Web of Things (WoT) Architecture
(W3C Recommendation 9 April 2020)

第 1 版

2021 年 11 月 11 日制定

一般社団法人
情報通信技術委員会

THE TELECOMMUNICATION TECHNOLOGY COMMITTEE

〈参考〉

1. 国際勧告等の関連

本標準は、W3C の Web of Things (WoT) Architecture W3C Recommendation 9 April 2020 (https://www.w3.org/TR/wot-architecture/) の和訳である。このドキュメントの正式版は W3C のサイト上にある英語版であり、このドキュメントには翻訳に起因する誤りがありえる。

2. 上記国際勧告等に対する追加項目等

2. 1 オプション選択項目

特になし

2. 2 ナショナルマター項目

特になし

2. 3 原標準に対する変更項目

できるだけ原文に忠実に翻訳を行ったが、読者の理解を助けるために補足が必要な箇所については、「訳者注釈」として追記した。

2. 4 その他

翻訳は、以下メンバーによる。

翻訳原本作成:上綱秀治（国立国会図書館）

査読担当:WoT 日本コミュニティ有志(以下)

芦村和幸(慶應義塾大学)、松田哲史（三菱電機株式会社)、水嶌友昭（株式会社インターネット総合研究所)、浅井智也 (一般財団法人 WebDINO Japan)、東村邦彦(株式会社日立製作所)、吉澤直美 (W3C/慶應)、太田浩史 (ヤフー株式会社)、塩濵大平 (Media Do International, Inc.)、川口透 (パナソニック株式会社)

2. 5 現勧告との章立て構成比較表

上記国際勧告との章立て構成の相違はない。

3. 改版の履歴

版数	制定日	改版内容
第 1.0 版	2021 年 11 月 11 日	制定

4. 工業所有権

本標準に関わる「工業所有権等の実施の権利に係る確認書」の提出状況は、TTC ホームページで御覧になれます。

5. 標準策定部門

IoT エリアネットワーク専門委員会（本標準の策定）

W3C Web of Things Working Group（W3C Web of Things Architecture の策定）

【注意】 この文書は、W3C の Web of Things (WoT) Architecture W3C Recommendation 9 April 2020 の和訳である。

この文書の正式版は W3C のサイト上にある英語版であり、この文書には翻訳に起因する誤りがありえる。誤訳、誤植などのご指摘は、翻訳チームの GitHub Issue (https://github.com/wot-jp-community/wotarchitecture/issues) までお願いしたい。

First Update: 2020 年 05月25日 | Last Update: 2021 年 5月6日

✧ ✧ ✧

Web of Things (WoT) アーキテクチャ

W3C 勧告 2020 年 4月9日

(https://www.w3.org/)

本バージョン:
https://www.w3.org/TR/2020/REC-wot-architecture-20200409/

最新公開バージョン:
https://www.w3.org/TR/wot-architecture/

最新編集者草案:
https://w3c.github.io/wot-architecture/

実装報告書:
https://w3c.github.io/wot-thing-description/testing/report.html

旧バージョン:
https://www.w3.org/TR/2020/PR-wot-architecture-20200130/

編集者:
Matthias Kovatsch (Huawei (https://www.huawei.com/))
Ryuichi Matsukura (Fujitsu Ltd. (https://www.fujitsu.com/))
Michael Lagally (Oracle Corp. (https://www.oracle.com/))
Toru Kawaguchi (Panasonic Corp. (https://www.panasonic.com/))
Kunihiko Toumura (Hitachi, Ltd. (https://www.hitachi.com/))
Kazuo Kajimoto (Former Editor, when at Panasonic)

参加可能:
GitHub w3c/wot-architecture (https://github.com/w3c/wot-architecture/)
File a bug (https://github.com/w3c/wot-architecture/issues/)
Commit history (https://github.com/w3c/wot-architecture/commits/master)
Pull requests (https://github.com/w3c/wot-architecture/pulls/)

貢献者:
In the GitHub repository (https://github.com/w3c/wot-architecture/graphs/contributors)

公開以後に報告されたエラーや問題がないか正誤表 (https://w3c.github.io/wot-architecture/errata.html) を確認のこと。

翻訳版 (http://www.w3.org/2003/03/Translations/byTechnology?technology=wot-architecture) も参照のこと。

概要

W3C の Web of Things (WoT) は、IoT プラットフォームとアプリケーション領域にまたがる相互運用性を可能にすることを目的としている。全体として、WoT の目標は、既存の IoT 標準とソリューションを維持し補完することである。一般的に、W3C WoT アーキテクチャは、何を実装するのかを規定するのではなく、何が存在するのかを記述することを目指している。

この WoT アーキテクチャの仕様では、W3C Web of Things の抽象アーキテクチャを記述している。この抽象アーキテクチャは、複数のアプリケーション領域のユースケースから導かれた一連の要件に基づいており、ユースケースおよび要件の両方とも、この文書で示されている。他の文書で詳細な仕様が示されているモジュール構成要素についても確認を行っている。この文書では、これらの構成要素がどのように関連付けられ、連携するかを記述している。WoT 抽象アーキテクチャは、様々な具体的な展開シナリオにマッピングできる基本的な概念フレームワークを定義したものであり、そのいくつかの例を示している。しかし、この仕様で記述している抽象アーキテクチャ自体は、具体的なメカニズムを定義したり、具体的な実装を規定したりするものではない。

この文書のステータス

この節は、この文書の公開時のステータスについて記述している。他の文書がこの文書に取って代わることがありえる。現行の W3C の刊行物およびこの技術報告の最新の改訂版のリストは、https://www.w3.org/TR/ の W3C 技術報告インデックス (https://www.w3.org/TR/) にある。

この文書は、抽象的なアーキテクチャの設計について記述している。しかし、関連する WoT Thing Description の仕様に基づいて一連の具体的な実装を記述している実装報告書 (https://w3c.github.io/wotthing-description/testing/report.html) がある。これらは、W3C Web of Things のアーキテクチャに準拠した実装である。

この文書は、W3C 会員、ソフトウェア開発者、他の W3C グループ、および他の利害関係者によりレビューされ、ディレクターにより W3C 勧告として承認されたものである。安定した文書であり、参考資料として用いること、他の文書で引用することができる。勧告の作成における W3C の役割は、その仕様への関心を引いて、広く普及させていくことにある。これにより、ウェブの機能および相互運用性の向上につながる。

この文書は、Web of Things ワーキンググループ (https://www.w3.org/WoT/WG/) によって勧告として公開された。

この仕様の議論には GitHub の Issue をお勧めする。または、メーリングリストにコメントを送信することもできる。コメントは public-wot-wg@w3.org (アーカイブ (https://lists.w3.org/Archives/Public/public-wot-wg/)
) に送信いただきたい。

この文書は、W3C 特許方針の下で活動しているグループによって作成された。W3C は、このグループの成果物に関連するあらゆる特許の開示の公開リストを維持し、このページには特許の開示に関する指示も含まれている。不可欠な請求権 (Essential Claim(s)) を含んでいると思われる特許に関して実際に知っている人は、W3C 特許方針の 6 項 (https://www.w3.org/Consortium/Patent-Policy/#sec-Disclosure) に従って情報を開示しなければならない。

この文書は、2019 年 3月1日の W3C プロセスドキュメント (https://www.w3.org/2019/Process-20190301/) に準拠している。

目次

§1. はじめに

Web of Things (WoT) の目標は、Internet of Things (IoT) の相互運用性と使いやすさを改善することである。多くの利害関係者が関わった長年にわたるコラボレーションを通じて、これらの課題の対処に役立ついくつかの構成要素が特定されている。

この仕様は、W3C WoT の標準化の範囲に焦点を当てており、その構成要素と、その関係を定義する抽象アーキテクチャとに分類できる。構成要素は、別の仕様で詳細に定義され説明されている。しかし、抽象アーキテクチャとその用語や概念フレームワークの定義に加え、この仕様は、WoT 構成要素に対する入門としての機能も果たし、それらの連携について説明している。

- Web of Things (WoT) Thing Description [WOT-THING-DESCRIPTION] は、Thing のメタデータとネットワーク向けインターフェースを記述するための機械可読データ形式を規定的に提供する。これは、相互作用のアフォーダンスなどの、この文書で紹介している基本概念に基づいている。

- Web of Things (WoT) バインディングテンプレート [WOT-BINDING-TEMPLATES] は、特定のプロトコルと IoT エコシステムに対する、Thing のネットワーク向けインターフェースを定義する方法 (プロトコルバインディングと呼ぶ) に関する参考情報のガイドライン を提供している。また、この文書では、多くの既存の IoT エコシステムと標準の例も提供している。

- Web of Things (WoT) スクリプティング API [WOT-SCRIPTING-API] は、オプションであり、これを用いると、ウェブブラウザ API のような一般的な JavaScript API で Thing のアプリケーションロジックを実装できるようになる。これによって、IoT アプリケーションの開発が簡素化され、ベンダーやデバイスにまたがる移植が可能となる。

- Web of Things (WoT) のセキュリティとプライバシーに関するガイドライン [WOT-SECURITY] は、分野横断的な構成要素を表す。この参考情報である文書は、Thing の安全な実装と設定に関するガイドラインを提供し、W3C WoT を実装するシステムで検討すべき課題について論じている。しかし、セキュリティとプ

ライバシーは、特定の実装に関する一揃いの具体的なメカニズムのコンテキストでのみ完全に評価されうるものであり、WoT の抽象アーキテクチャでは完全には規定していないことを強調しておくべきである。これは特に、WoT アーキテクチャが既存のシステムに記述的に用いられる場合に当てはまる。なぜならば、

W3C WoT はそのようなシステムの動作を制約することはできず、それを記述することしかできないからである。この文書では、§ 10. セキュリティとプライバシーへの配慮の項で、プライバシーとセキュリティに関するリスクとその軽減策について概観する。

この仕様は、WoT システムの展開に関する非規定的なアーキテクチャの側面と条件もカバーしている。この仕様は特定の具体的な実装を規定的に定義するものではないが、このガイドラインは、展開シナリオの例に照らして記述されている。

この仕様は、W3C WoT 仕様を包括する機能を有しており、用語や W3C Web of Things の基本となる抽象アーキテクチャなどの基礎を定義している。要約すると、この仕様の目的は次を提供することである。

- § 4. ユースケースでは、W3C WoT アーキテクチャへとつながったユースケース、
- § 5. 要件では、WoT 実装の要件、
- § 6. WoT アーキテクチャでは、抽象アーキテクチャの定義、
- § 7. WoT 構成要素では、WoT 構成要素とその相互作用の概要、
- § 8. 抽象的な Servient のアーキテクチャでは、抽象アーキテクチャを存在しうる具体的な実装にマッピングする方法に関する参考情報のガイドライン、
- § 9. WoT のデプロイメント例では、存在しうるデプロイメントシナリオに関する参考情報の例、

および§ 10. セキュリティとプライバシーへの配慮では、W3C WoT アーキテクチャに基づくシステムを実装する際に注意すべきセキュリティとプライバシーに関する留意点の高いレベルでの議論。

追加の要件、ユースケース、概念的な機能、および新しい構成要素に関しては、この文書の将来の改訂で取り組む。

§2. 適合性

非規定的と記している項と同じく、この仕様のすべての作成ガイドライン、図、例、注は、参考情報である。この仕様のその他の部分はすべて規定的である。

この文書の「することができる/してもよい (MAY)」、「しなければならない (MUST)」、「すべきである/ する必要がある (SHOULD)」というキーワードは、ここで示しているように、すべて大文字で表示されている場合にのみ、BCP 14 [RFC2119] [RFC8174] で記述されているように解釈されるべきである。

§3. 用語

この章は参考情報である。

この仕様では、ここで定義しているとおりに次の用語を用いる。WoT の接頭辞は、モノのウェブの概念のために特別に（再）定義されている用語の曖昧さを回避するために用いる。

アクション (Action)
: 状態を操作したり（例えば、照明のオン/オフを切り替える）、Thing におけるプロセスを始動させる（例えば、時間の経過とともに照明を暗くする）といった、Thing の機能の呼び出しを可能にする相互作用のアフォーダンス。

バインディングテンプレート (Binding Templates)

様々なIoTプラットフォームとの通信のための再利用可能な青写真の集合。この青写真は、WoT Thing Descriptionと、必要なプロトコルスタックや専用通信ドライバーに関する実装注記によって、相互作用のアフォーダンスをプラットフォーム固有のメッセージにマッピングするための情報を提供する。

利用される Thing（Consumed Thing）ローカルなアプリケーションが用いるリモートの Thing を表すソフトウェア抽象化。この抽象化は、ネイティブな WoT ランタイムが作成したり、WoT スクリプティング API がオブジェクトとしてインスタンス化する可能性がある。

Thing を利用する（Consuming a Thing）
TD ドキュメントを解析して処理し、それから、ローカルなランタイム環境にあるアプリケーションに対するインターフェースとして、利用される Thing のソフトウェア抽象化を作成すること。

Consumer
WoT Thing Description（JSON ベースの表現形式を含む）を処理し、Thing と相互作用する（つまり、Thing を利用する）ことができるエンティティー。

データスキーマ（Data Schema）
データスキーマは、情報モデルおよび関連するペイロード構造と、相互作用中に Thing と Consumer の間で受け渡される対応するデータ項目を記述する。

デジタルツイン（Digital Twin）
デジタルツインは、クラウドやエッジノード上に存在するデバイスやデバイス群の仮想表現である。オンライン上に継続的に存在していない可能性のある実在するデバイスを表したり、実際のデバイスに展開する前に新しいアプリケーションやサービスをシミュレーションしたりするために使用できる。

領域固有の語彙（Domain-specific Vocabulary）
WoT Thing Description で使用できるリンクトデータの語彙だが、W3C WoT では定義していない。

エッジデバイス（Edge Device）
企業やサービス提供者の基幹ネットワークへのエントリポイントを提供するデバイス。例には、ゲートウェイ、ルーター、スイッチ、マルチプレクサ、およびその他の様々な接続デバイスが含まれる。

イベント（Event）
イベントの情報源を記述している相互作用のアフォーダンスで、イベントデータを利用者に非同期でプッシュする（例えば、オーバーヒートの警報）。

公開された Thing（Exposed Thing）
リモートの利用者がネットワーク経由でアクセスできる、ローカルで提供されている Thing を表すソフトウェア抽象化。この抽象化は、ネイティブな WoT ランタイムが作成したり、WoT スクリプティング API がオブジェクトとしてインスタンス化する可能性がある。

Thing を公開する（Exposing a Thing）
Thing の状態を管理し、動作の実装と連動するために、ローカルなランタイム環境にある公開対象の Thing のソフトウェア抽象化を作成すること。ハイパーメディア制御（Hypermedia Control）
ハイパーメディアにおけるプロトコルバインディングのシリアライゼーション、つまり、ナビゲーション用のウェブリンク［RFC8288］や、他の操作を実行するためのウェブフォーム。フォームは、利用者が項目に記入して送信するための、Thing が提供するリクエストテンプレートと考えることができる。

相互作用のアフォーダンス（Interaction Affordance）
可能な選択肢を利用者に提示し説明する Thing のメタデータで、これにより、利用者がどのように Thing と相互作用できるかを提案する。潜在的なアフォーダンスには多くの種類があるが、W3C WoT では、プロパティー、アクション、イベントという 3 種類の相互作用のアフォーダンスを定義している。四つ目の相互作

用のアフォーダンスはナビゲーションで、これは、ウェブでは既にリンク付けという方法で利用できる。

相互作用モデル（Interaction Model）

アプリケーションの意図（application intent）から具体的なプロトコル操作へのマッピングを形式化し、限定する中間の抽象化。W3C WoT では、定義済みの相互作用のアフォーダンスの集合が相互作用モデルを構成する。

Intermediary
Thing を代理、拡張、または構成する、Consumer と Thing の間のエンティティーで、元の Thing ではなく仲介の WoT インターフェースを指し示す WoT Thing Description を再公開できる。REST の階層化システムの制約により、利用者には、Intermediary は Thing と見分けがつかないかもしれない。

IoT プラットフォーム（IoT Platform）
OCF、oneM2M、Mozilla プロジェクトの Thing などの特定の IoT エコシステムで、アプリケーション向けの API、データモデル、プロトコルまたはプロトコル設定に関して独自の仕様を備えている。

メタデータ（Metadata）
エンティティーの抽象的な特性に関する記述を提供するデータ。例えば、Thing Description は Thing のメタデータである。

個人識別可能情報（PII）（Personally Identifiable Information（PII））
ある情報と関係性がある自然人を特定するために使用できる情報、または自然人に直接または間接的にリンクされている、またはその可能性がある情報。［ISO-IEC-29100］と同じ定義を用いる。

プライバシー（Privacy）
私生活や個人的な出来事への侵害（その個人に関するデータを不当または違法に収集し使用した結果、侵害が生じた場合）がないこと。［ISO-IEC-2382］と同じ定義を用いる。個人を特定できる情報、セキュリティ、および［ISO-IEC-29100］のその他の関連定義も参照のこと。

Private Security Data
Private Security Data は、Thing のセキュリティ設定の構成要素であり、秘密に保たれ、他のデバイスやユーザと共有されない。ひとつの例は、PKI システムの秘密鍵である。理想的には、そのようなデータは、アプリケーションがアクセスできない別のメモリに保存され、それを利用するアプリケーションにも秘密情報を漏らさない、署名などの抽象的な操作を介してしか用いられない。

プロパティー（Property）
Thing の状態を公開する相互作用のアフォーダンス。この状態は、後で取得（読み取り）し、オプションで更新（書き込み）できる。Thing は、変更後の新しい状態をプッシュすることにより、プロパティーを監視可能にすることも選択できる。

プロトコルバインディング（Protocol Binding）
相互作用のアフォーダンスから特定のプロトコルの具体的なメッセージへのマッピング。これにより、利用者に相互作用のアフォーダンスを作動させる方法を通知する。W3C WoT は、プロトコルバインディングをハイパーメディア制御としてシリアライズする。

Public Security Metadata
Public Security Metadata は、Thing にアクセスするために必要なセキュリティメのメカニズムとアクセス権を記述した、Thing のセキュリティ設定の構成要素である。これには秘密情報や具体的なデータ（公開キーを含む）は含まれておらず、単独で Thing へのアクセスを提供することはない。代わりに、ユーザがどのように認証を得なければならないかを含め、承認されたユーザがアクセスを取得する方法が記述されている。

セキュリティ（Security）
情報の機密性、完全性、および可用性の保持。真正性、責任追跡性、否認防止、信頼性などのプロパティーも関係する場合がある。この定義は、［ISO-IEC-27000］の情報セキュリティの定義の焼き直しで、これには、言及されている個々のより具体的なプロパティーに関する追加定義も含まれている。その他の関連する定義については、この文書を参照いただきたい。さらに、このプロパティーは、通常動作時とシステムが攻撃にさらされている時の両方の場合において保持されることが望ましいことに注意のこと。

セキュリティ構成情報（Security Configuration）
Public Security Metadata、Private Security Data、および Thing のセキュリティメカニズムを運用上構成するために必要なその他の構成情報（公開キーなど）の組み合わせ。

Servient
: WoT 構成要素を実装するソフトウェアスタック。Servient は、Thing を提供し公開することができ、かつ（または）Thing を利用する Consumer の役割を務めることができる。Servient は、様々な IoT プラットフォームと相互作用できるように、複数のプロトコルバインディングをサポートできる。

サブプロトコル（Subprotocol）
: 相互作用をうまく行うために知っていなければならない転送プロトコルに対する拡張メカニズム。例は、HTTP に対するロングポーリング（long polling）である。

TD
: WoT Thing Description の略。

TD 語彙（TD Vocabulary）
: WoT バインディングテンプレートの通信メタデータを含む、WoT Thing Description において Thing のメタデータにタグ付けを行うための W3C WoT によるリンクトデータの統制語彙。

Thing または Web Thing
: WoT Thing Description でメタデータとインターフェースが記述されている物理エンティティーまたは仮想エンティティーの抽象化。それに対し、仮想エンティティーは一つ以上の Thing の合成物である。

Thing Directory
: （[CoRE-RD] のように）TD を登録して検索する（例えば、SPARQL クエリや CoRE RD ルックアップインターフェース [CoRE-RD] を用いて）ためのウェブインターフェースを提供する，TD のディレクトリサービス。

転送プロトコル（Transfer Protocol）
: オプションやサブプロトコルメカニズムに対してアプリケーション固有の要件や制約のない、基礎となる標準的なアプリケーション層のプロトコル。例は、HTTP、CoAP、または MQTT である。

ヴァーチャル Thing
: 別のシステム構成要素に置かれている Thing を表す、Thing のインスタンス。

WoT インターフェース（WoT Interface）
: WoT Thing Description で記述されている、hting のネットワーク向けインターフェース。

WoT ランタイム（WoT Runtime）
: アプリケーションの実行環境を維持し、Thing を公開および（または）利用し、WoT Thing Description を処理し、セキュリティ設定を維持し、プロトコルバインディング実装と連動できるランタイムシステム。

  WoT ランタイムは、特注の API を持つか、オプションの WoT スクリプト API を用いることができる。

WoT スクリプティング API（WoT Scripting API）
: WoT ランタイムで実行される動作やアプリケーションの実装を容易にするために、Servient が提供するアプリケーション向けプログラミングインターフェース。ウェブブラウザ API に相当する。WoT スクリプティング API は、W3C WoT のオプションの構成要素である。

WoT Servient
: Servient の同義語。

WoT Thing Description または Thing Description
: Thing を記述した構造化データ。WoT Thing Description は、一般的なメタデータ、領域固有のメタデータ、相互作用のアフォーダンス（サポートされるプロトコルバインディングを含む）、および関係する Thing へのリンクで構成される。WoT Thing Description は、W3C WoT の中心となる構成要素である。

§4. ユースケース

この章は参考情報である。

この項では、W3C WoT の対象である、§ 7. WoT 構成要素で論じている抽象アーキテクチャを導き出すために用いたアプリケーション領域とユースケースを示す。

Web of Things のアーキテクチャは、ユースケースとアプリケーション領域に制限を設けていない。抽象アーキテクチャが満たさなければならない一般的なパターンを収集するために、様々なアプリケーション領域について検討が行われた。

以下の項は網羅的ではない。むしろ、それらは例示としての機能を果たすものであり、接続されたモノからさらに恩恵を得たり、新しいシナリオが可能となったりする可能性がある。

§4.1 アプリケーション領域

§4.1.1 消費者

消費者空間には、接続によって恩恵を得られる複数の資産がある。部屋の利用状況に基づいて照明とエアコンをオフにできる。気象条件と人の存在に基づいてブラインドを自動的に閉じることができる。エネルギーやその他の資源の利用は、使用パターンと予測に基づいて最適化できる。

この項の消費者のユースケースには、スマートホームのユースケースが含まれる。

図 1 は、スマートホームの例である。このケースでは、ゲートウェイは、対応する KNX、ECHONET、ZigBee、DECT ULE、Wi-SUN などのローカルな通信プロトコルにより、センサー、カメラ、家電などのエッジデバイスに接続されている。一つの家に複数のゲートウェイが存在可能で、各ゲートウェイは複数のローカルなプロトコルをサポートできる。

ゲートウェイはインターネット経由でクラウドに接続できるが、クラウドに直接接続できる機器もある。クラウドで実行されているサービスは、エッジデバイスからデータ収集を行い、そのデータを分析し、その後でエッジデバイスやその他の UX デバイスを介してユーザに価値を提供する。

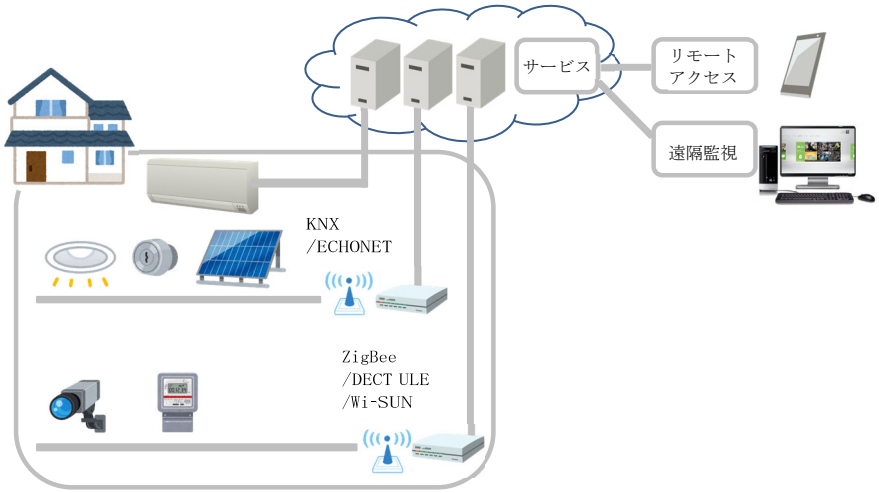

図 1 スマートホーム

スマートホームは、リモートのアクセスと制御、音声制御、ホームオートメーションなどのメリットを消費者に提供する。スマートホームにより、デバイスの製造者がリモートでデバイスを監視しメンテナンスを行うことも可能となる。スマートホームは、エネルギー管理やセキュリティ監視などの付加価値サービスを実現できる。

§4.1.2 産業

この項の産業に関するユースケースは、様々な産業部門に適用できる。

アプリケーションシナリオには重複する性質があるため、異なる業種に似たようなユースケースが存在する。

§ 4.1.2.1 例: スマートファクトリー

図 2 は、スマートファクトリーの例である。このケースでは、現場レベル、セル、エ場ラインの制御装置が、PROFINET、Modbus、OPC UA TSN、EtherCAT、CAN などの産業用通信プロトコルに基づいて、様々なエ場設備をオートメーション化している。産業用のエッジデバイスは、様々な制御装置からデータを選択収集して、例えば、ダッシュボードを用いた遠隔監視などのクラウドのバックエンドサービスでデータを利用できるようにしたり、予防保全のためにデータの分析を行う。

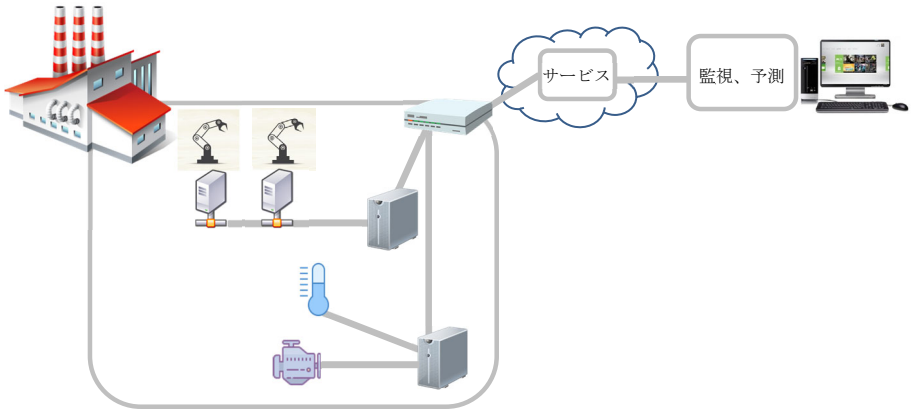

図 2 スマートファクトリー

スマートファクトリーでは、接続された製造設備と製品の高度な監視が必要である。機械の故障を予測し、異常を早期に発見して、コストのかかるダウンタイムとメンテナンスの労力を防ぐことで恩恵を受けられる。

さらに、有毒ガス、過度な騒音や熱の存在に関して、接続された製造設備と生産設備の環境を監視することにより、労働者の安全性が向上し、事件や事故のリスクが減少する。

生産設備のリアルタイム監視と KPI 計算は、生産性の問題を検出し、サプライチェーンを最適化するのに役立つ。

§ 4.1.3 輸送と物流

車両、燃料費、メンテナンスの必要性、割り当ての監視は、業務用車両を最大限活用できるように最適化するのに役立つ。

輸送品の一貫した品質と状態を確保するために、積み荷が配送中であることを追跡できる。これは、倉庫から冷蔵トラック、配達までのコールドチェーンの完全性を言明するのに特に役立つ。

倉庫とヤードでの一元的な在庫の監視と管理により、在庫切れや過剰在庫の状況を防ぐことができる。

§ 4.1.4 公益事業

住宅や商工業（C&I）のメーターの自動読み取りと請求により、資源の消費と潜在的なボトルネックに関する継続的な洞察が得られる。

分散型再生可能エネルギー生成装置の状態と出力を監視することにより、分散型エネルギー資源の最適化が可能になる。

配電設備の監視と遠隔制御は、配電プロセスの自動化に役立つ。

発電と配電のインフラストラクチャの継続的な監視により、公益事業の現場担当者の安全性が向上している。

§4.1.5 オイルとガス

タンクや貯蔵庫における貯蔵量を監視および制御することに加えて、海洋プラットフォームの監視や、パイプラインの漏れを検知および予測することは、環境のために加えて，従業員にとっての労働安全性改善に役立つ。

様々な貯蔵タンクと配送パイプ/トラックを通じた分散型在庫の自動計算により、計画の改善と資源の最適化が可能となる。

§4.1.6 保険

連結構造物、業務用車両などの高い価値を持つ資産の予防的資産監視により、事件の予測と早期発見を通して、深刻な損害と高い損失のリスクを軽減する。

利用状況の追跡とカスタマイズされた保険契約を用いて、利用ベースの保険を提供できる。

予測的な気象監視を行い、業務用車両を屋根付き車庫にルート変更させると、ひょう害、樹木の被害による損失を抑えることができる。

§4.1.7 土木工事と建設

産業上の安全性を監視することにより、セキュリティ上の危険性のリスクが軽減される。建設現場の資産を監視することで、被害や損失を防止できる。

§4.1.8 農業

土壌の状態監視と、散水、施肥の最適な計画の作成、農産物の状態監視により、農産物の品質と生産量が最適化される。

§4.1.9 医療

臨床試験データのデータ収集と分析は、新しい領域に関する洞察を得るのに役立つ。

遠隔患者モニタリングは、高齢者や入院後の患者の重篤な状況を見逃すリスクを軽減する。

§4.1.10 環境モニタリング

環境モニタリングは通常、測定データを共通のゲートウェイ、エッジデバイス、クラウドサービスに送信する多くの分散型センサーに依存している。クリティカルな環境状態を検出するために、大気汚染や水質汚染、そして、微粉塵、オゾン、揮発性有機化合物、放射能、温度、湿度などのその他の環境リスク要因を監視することにより、回復不能な健康や環境の被害を防ぐことができる。

§4.1.11 スマートシティ

橋梁、ダム、堤防、運河の資材の状態、劣化、振動の監視により、保守修理作業を発見し、重大な被害を防止することができる。幹線道路の監視と適切な標識の提供により、最適な交通の流れが確保される。

スマートパーキングにより、駐車スペースの利用と可用性の最適化と追跡が行われ、課金/予約が自動化される。

存在検知、天気予報などに基づく街灯のスマート制御により、コストが削減される。

ゴミ容器の監視により、廃棄物管理やゴミ収集ルートを最適化できる。

§4.1.12 スマートビルディング

ビル全体のエネルギー使用量の監視は、資源消費の最適化と無駄の削減に役立つ。

冷暖房空調設備（HVAC）、エレベーターなどのビル内の設備を監視し、早期に問題を解決することで、居住者の満足度が向上する。

§4.1.13 コネクテッドカー

稼働状況の監視、サービスニーズの予測により、メンテナンスの必要性とコストが最適化される。危機的な道路・交通状況に関する早期警告システムの通知により、ドライバーの安全性が強化される。

> 翻訳者のメモ
>
> 「稼働状況の監視、サービスニーズの予測により、メンテナンスの必要性とコストが最適化される。」という部分は、「適切なコストで適切なメンテナンスがなされる」という趣旨である。』

§4.1.13.1 コネクテッドカーの例

図3は、コネクテッドカーの例である。このケースでは、ゲートウェイはCANを介して車の部品に接続され、独自のインターフェースを介してカーナビゲーションシステムに接続される。クラウドで実行されているサービスは、交通のパターンを判断するために、車の部品からプッシュ配信されるデータを収集し、複数の車からのデータを分析する。このケースでは、ゲートウェイはクラウドサービスを利用して交通データを取得し、カーナビゲーションシステムを通じてドライバーに表示することもできる。

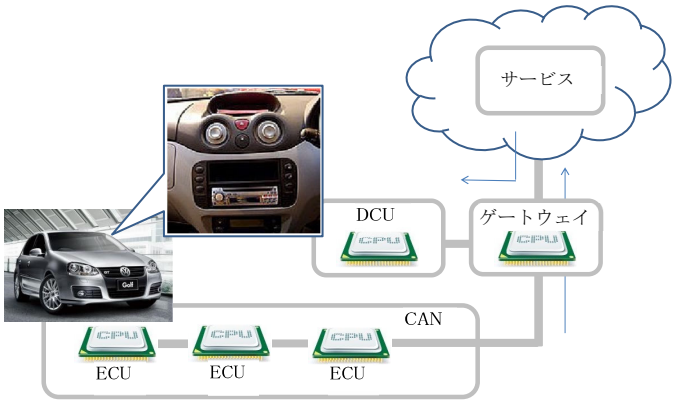

図3 コネクテッドカー

§4.2 共通パターン

この項では、デバイス/Thingが、コントローラー、他のデバイス、エージェント、およびサーバーとどのように相互作用するかを示す共通的なユースケースのパターンを紹介する。この項では、トランスポートプロトコルの発動要素としてクライアントの役割（client role）という用語を用い、トランスポートプロトコルの受動要素としてサーバーの役割（server role）という用語を用いる。これは、システム構成要素に特定の役割を定めることを意味するものではない。一つのデバイスは、クライアントとサーバーの役割を同時に持つことができる。

翻訳者のメモ

英語原本中では「client role」が要素によって強調されている一方で，「server role」は強調されていないが，「server role」も同様に要素により強調するべきと考えられる．

このニ重の役割の例の一つはセンサーであり、クラウドサービスに自身を登録するとともに、センサーの測定値を定期的にクラウドに送信する。応答メッセージでは、クラウドは、センサーのメッセージ送信速度を調整したり、特定のセンサー属性を選択したりすることができ、これらは今後送信されることになるメッセージを対象としたものである。センサーは自身をクラウドで登録して接続を開始するため、これは「クライアント」の役割である。しかし、応答メッセージで送信されるリクエストにも反応するため、「サーバー」の役割も果たす。

以下の項では、複雑さが増しつつある役割、タスク、ユースケースのパターンについて説明する。これらは網羅的なものではなく、この仕様の後半の項で定義している WoT アーキテクチャと構成要素を動機付けるために提
示されるものである。

§4.2.1 デバイスコントローラー

図 4 で示しているように、最初のユースケースは、ユーザが操作するリモートコントローラーで制御されるローカルなデバイスである。リモートコントローラーは、ローカルなホームネットワークを介して電子機器に直接アクセスできる。このケースでは、リモートコントローラーはブラウザやネイティブなアプリケーションで実装できる。

このパターンでは、電子機器などの少なくとも一つのデバイスには、他のデバイスからのリクエストを受け入れて応答できるサーバーの役割があり、時として機械的なアクションを開始する。リモートコントローラーのような他のデバイスには、センサー値の読み取りやデバイスの電源投入などの、リクエストに応じてメッセージを送信できるクライアントの役割がある。さらに、デバイスの現在の状態やイベントの通知を送信するために、デバイスは、サーバーの役割を持つ別のデバイスに対してメッセージを送信できるクライアントの役割を持つことができる。

図 4 デバイス制御

§4.2.2 Thing と Thing

図 5 は、直接的な Thing と Thing（Thing-to-Thing）の相互作用の例を示している。シナリオは次のとおりである。センサーが部屋の状況変化、例えば、温度が基準値を超えていることを検出し、電子機器に対して「オンにする」などの制御メッセージを発信する。センサーユニットは、他のデバイスにトリガーメッセージを発信できる。

このケースでは、サーバーの役割を持つニつのデバイスが接続されている場合、少なくとも一方のデバイスには、他方に対して作動させたり通知するためにメッセージを発信するクライアントの役割もなければならない。

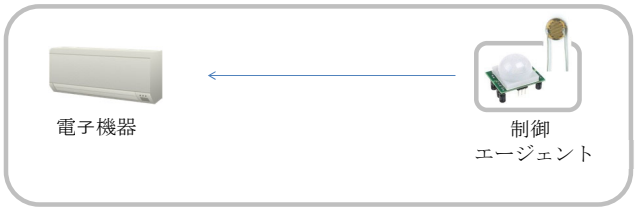

図 5 制御エージェント

§4.2.3 リモートアクセス

図 6 で示しているように、このユースケースには、モバイルリモートコントローラー（例えば、スマートフォン）が含まれる。リモートコントローラーは、セルラーネットワークや、Wi-Fi や Bluetooth などのプロトコルを用いたホームネットワークなどの様々なネットワーク接続とプロトコルを切り替えることができる。コントローラーがホームネットワークにある場合、それは信頼できるデバイスであり、セキュリティやアクセス制御を追加する必要はない。コントローラが信頼できるネットワークの外にある場合は、アクセス制御やセキュリティのメカニズムを追加適用して、信頼関係を確保しなければならない。このシナリオでは、様々なネットワークアクセスポイントやセルラー基地局間の切り替えにより、ネットワークの接続性が変わりえることに注意すること。

このパターンでは、図 4 の関連するシナリオと同様に、リモートコントローラーと電子機器は、クライアントとサーバーの役割を持っている。

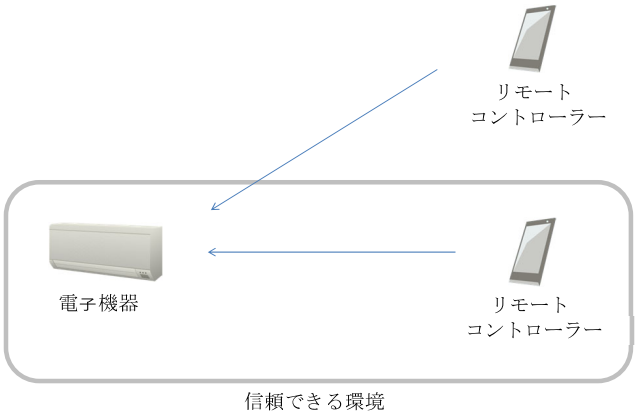

図 6 複数のネットワークインターフェース

§4.2.4 スマートホームゲートウェイ

図 7 は、スマートホームゲートウェイを用いたユースケースを示している。このゲートウェイは、ホームネットワークとインターネットの間に位置づけられる。これは、屋内の電子機器を管理し、前述のユースケースのようにスマートフォンからなど、インターネット経由でリモートコントローラーからの命令を受信できるようにする。これは、デバイスの仮想表現でもある。スマートホームゲートウェイは通常、プロキシとファイアウォールの機能を提供する。

翻訳者のメモ

英語原本中、「It is also is a virtual representation of a device.」とあるが、一文中に「is」が二つあり，

二つ目の「is」（=「is a virtual representation」の「is」）は不要。

このパターンでは、ホームゲートウェイは、クライアントとサーバーの両方の役割を持っている。リモートコントローラーが電子機器を作動させた時に、クライアントの役割の電子機器とサーバーの役割のリモートコントローラーとの接続を可能とする。電子機器がリモートコントローラーにメッセージを送信する際は、ゲートウェイは、電子機器に対するサーバーの役割を果たし、リモートコントローラーに対するクライアントの役割を果たす。

翻訳者のメモ

英語原本中、「the gateway act as server roles ... and it act as client roles」とあるが、「gateway」が単数形であることから、「act」は、いずれも「acts」が正しいと考えられる。 また、「act server roles」および「act client roles」が正しいと考えられる。

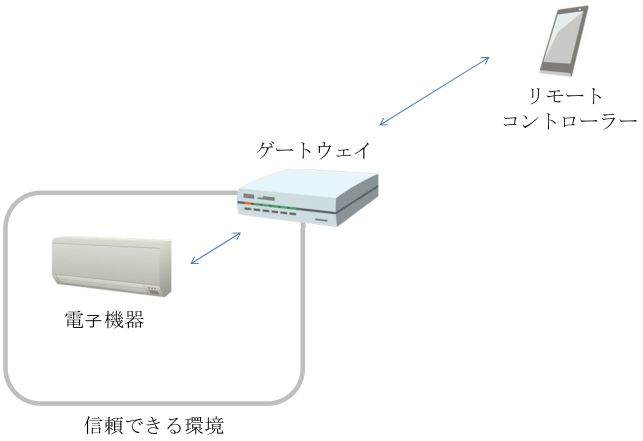

図 7 スマートホームゲートウェイ

§4.2.5 エッジデバイス

エッジデバイスまたはエッジゲートウェイは、スマートホームゲートウェイに類似している。この用語は、エッジゲートウェイによって実行される追加のタスクを示すために用いる。図 8 のホームゲートウェイは主に公開されているネットワークと信頼できるネットワークの間を橋渡しするだけだが、エッジデバイスにはローカルな計算性能があり、通常は、異なるプロトコル間の橋渡しを行う。エッジデバイスは通常、産業界のソリューションで用いられ、その場合には、接続されたデバイスとセンサーが提供するデータの前処理、フィルタリング、集約を行うことができる。

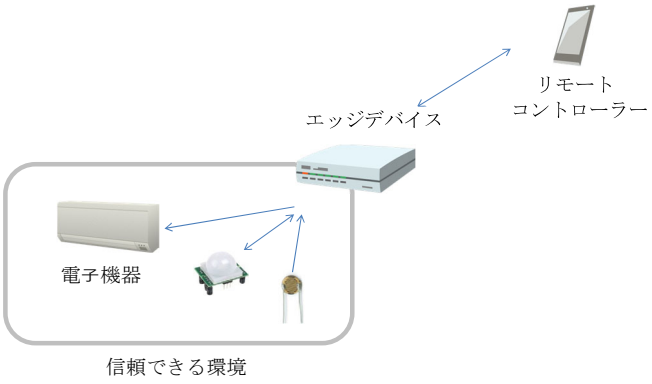

図 8 エッジデバイス

§4.2.6 デジタルツイン

デジタルツインは仮想表現である。つまり、クラウドサーバーやエッジデバイスに存在するデバイスまたはデバイス群のモデルである。これは、継続的にオンライン上に存在するとは限らないデバイスを表すため、または、新しいアプリケーションやサービスを実際のデバイスに展開する前にシミュレーションを実行するために使用される。

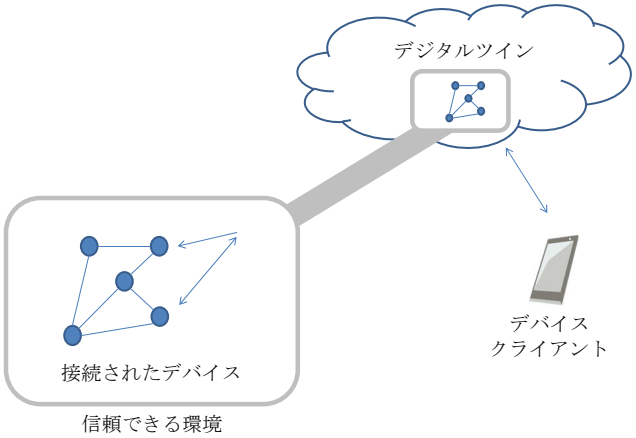

図 9 デジタルツイン

デジタルツインは単一のデバイスをモデル化するこもでき、結合されたデバイスの仮想表現に複数のデバイスを集約することもできる。

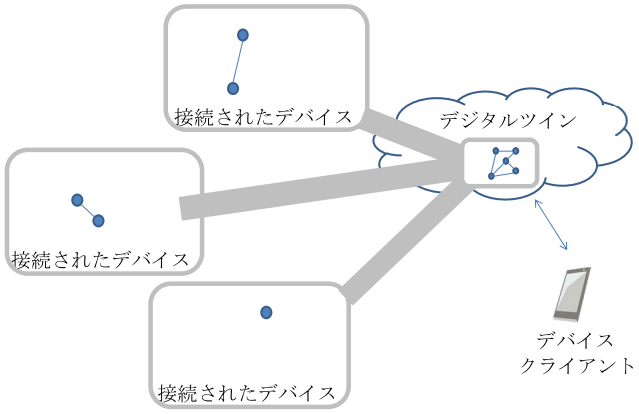

図 10 複数のデバイスのデジタルツイン

デジタルツインは、デバイスが既にクラウドに接続されているか、それともクラウドに接続されているゲートウェイに接続されているかに応じて、様々な方法で実現できる。

§ 4.2.6.1 クラウド対応デバイス

図 11 は、電子機器がクラウドに直接接続されている例を示す。クラウドは機器をミラーリングし、デジタルツインとして機能し、リモートコントローラー（例えば、スマートフォン）から命令を受信する。デジタルツインはグローバルに到達可能であるため、承認されたコントローラーはどこにでも設置できる。

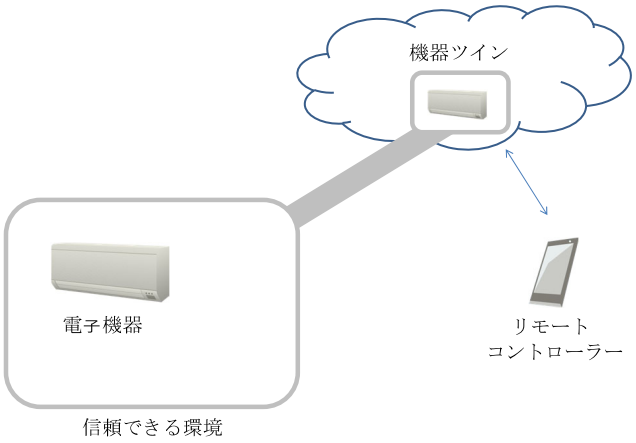

図 11 クラウド対応デバイスの機器ツイン

§4.2.6.2 旧式デバイス

図 12 は、旧式の電子機器をクラウドに直接接続できない例を示す。ここでは、接続を中継するためにゲートウェイが必要である。ゲートウェイは次のような機能を果たす。

- 物理的と論理的の両方の観点での様々な旧式の通信プロトコルのインテグレータインターネットに対する
- ファイアウォール
- 実際の画像や音声を置き換え、データをローカルで記録するプライバシーフィルタネットワーク接続が中
- 断された場合のローカルなエージェント

火災警報や同様のイベントが発生したときにローカルで実行される緊急サービス

クラウドは、接続されたすべての機器とゲートウェイをミラーリングし、それらをゲートウェイと連携してクラウド内で管理するデジタルツインとして機能する。さらに、クラウドはどこにでも設置できるリモートコントローラー（例えば、スマートフォン）からの命令を受信することができる。

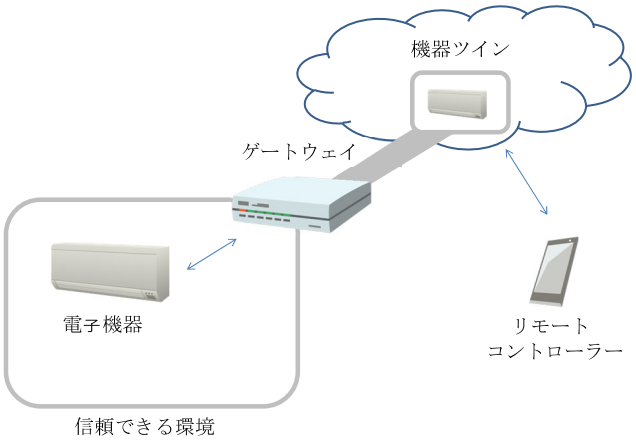

図 12 旧式デバイスのデジタルツイン

§4.2.7 マルチクラウド

典型的な IoT のデプロイメントは、複数（数千）のデバイスで構成される。標準的なメカニズムがなければ、特定のクラウドのためのファームウェア更新の管理には、多大な労力が必要であり、IoT の広範な採用の妨げとなる。

デバイスおよびデバイス型を記述するための標準的なメカニズムの主な利点は、デバイスに対してソフトウェアやファームウェアのレベルでカスタマイズを行う必要なく、つまり、クラウド固有のコードをデバイスにインストールする必要なく、デバイスを異なるクラウド環境にデプロイできることである。これは、このソリューションは、複数の IoT クラウド環境のデバイスに対して新規に接続し（on-boarding）使用することを可能とするような形でデバイスの記述ができるくらいに柔軟なものであることを意味する。これにより、既存のデバイスをクラウドからクラウドへ移行することが可能になるとともに、既存のデプロイメントにおける新しいデバイスの利用が簡単になるため、Web of Things デバイスの採用が促進される。

§4.2.8 領域横断型連携

図 13 は、領域横断型連携の例を示している。この場合、各システムは、スマートファクトリーとスマートシティ、スマートシティとスマートホームのように、他の領域に含まれる他のシステムと関わり合いを持っている。［IEC-FOTF］で示す通り、この種のシステムは「共生」（symbiotic）エコシステムと呼ばれる。その中には、直接的連携と間接的連携の二つのモデルがある。直接的連携モデルでは、システムはピアツーピア（peer-to-peer）方式で互いに情報を直接交換する。間接的連携では、システムは何らかの連携プラットフォームを介して情報を交換する。この連携を維持・継続するために、各システムは自身の能力やインタフェースに関するメタデータを提供し、他のシステムに適応させる。

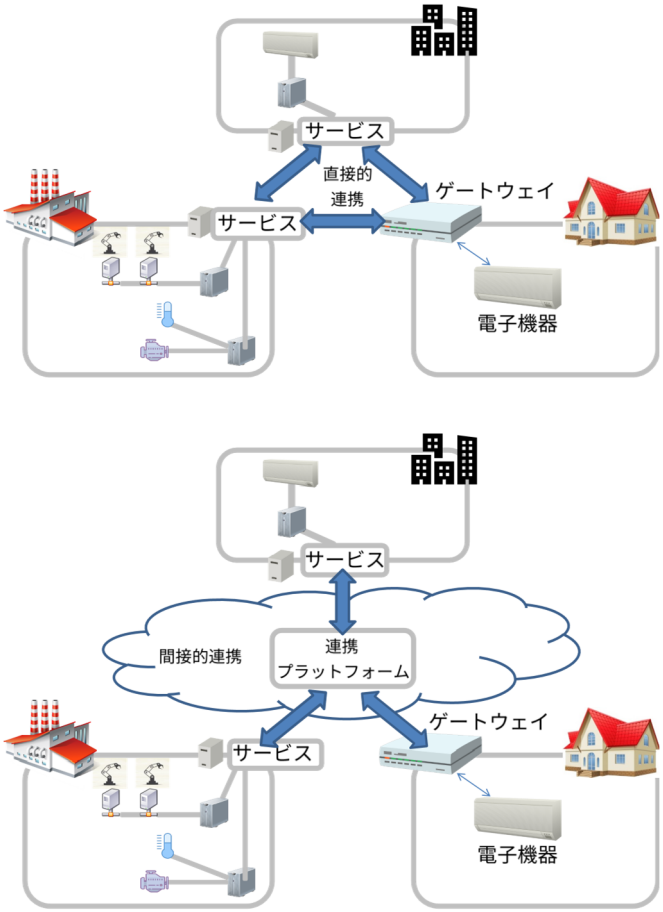

図 13 領域横断型連携

§4.3 要約

前項では、様々なアーキテクチャのパターンについて説明した。これらのパターンでは、旧式デバイス、コントローラー、ゲートウェイ、クラウドサーバーを含むデバイスなどの一部の機能のエンティティーは、建物の内外やデータセンターなどの物理的な場所に置かれる。図 14 は、これらのエンティティーの組み合わせと通信経路を示した概要である。

トランスポートプロトコル層では、各エンティティーが通信に適した役割を任意に選択する。例えば、デバイスが不特定多数のアプリケーションにサービスを提供する場合、そのデバイスはサーバーとして機能する。一方で、デバイスのネットワーク接続が制限されていたり断続的であったりする場合は、デバイスはクライアントとして機能し、ネットワークが利用できるときにアプリケーションにメッセージを活発に送信することもできる。これに関係なく、アプリケーション層では、アプリケーションは、デバイスが相互作用を行うための抽象的なインターフェースを提供していることを理解し、その抽象的なインターフェースを用いてデバイスと相互作用を行うことができる。

翻訳者のメモ

英語原本中で「On the other hand, if a device has limited or intermittent network connectivity, they
may act as a client...」とあるが、「they may act as...」の「they」は、「device」に対する代名詞であることから「it」が正しいと考えられる。その他、いくつかの誤字について、W3C 側に指摘ずみ。

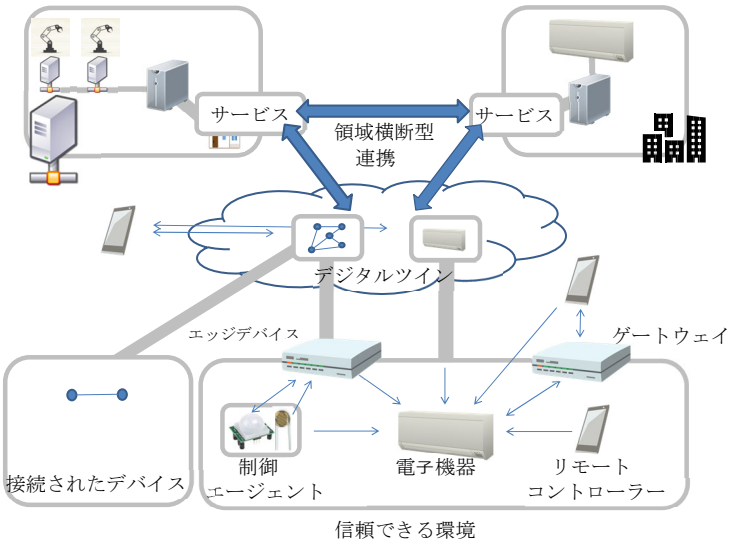

図 14 ユースケースの概要

§5. 要件

この章は規定である。

§5.1 機能要件

この節では、Web of Things (WoT) の抽象アーキテクチャに必要な特性を定義する。

§5.1.1 一般的な原則

- WoT アーキテクチャは、ウェブ技術を用いて異なるエコシステムの相互運用を可能にすべきである。
- WoT アーキテクチャは、RESTful API を用いたウェブアーキテクチャに基づくべきである。
- WoT アーキテクチャでは、ウェブで一般的に用いられている複数のペイロード形式を使用できるべきである。
- WoT アーキテクチャでは、様々なデバイスアーキテクチャが可能でなければならず、システム構成要素として特定のクライアントやサーバーの実装を強制してはならない。

柔軟性

WoT 実装には、様々な物理デバイスの構成がある。WoT の抽象アーキテクチャは、そのすべてのバリエーションに対してマッピングができ、それらをカバーできるべきである。

- 互換性

  多くの業界には、すでに多くの既存の IoT ソリューションと進行中の IoT 標準化活動がある。WoT は、これらの既存および開発中の IoT ソリューションとウェブ技術との間の橋渡しを WoT の概念に基づいて提供すべきである。WoT は、既存の IoT ソリューションおよび現在の標準の上位互換であるべきである。

- スケーラビリティ

  WoT は、数千から数百万のデバイスを組み込む IoT ソリューションに対応できなければならない。これらのデバイスは、製造者が異なっていても同じ機能を提供する可能性がある。

- 相互運用性

  WoT は、デバイスとクラウドの製造者にまたがる相互運用性を提供しなければならない。WoT 対応デバイスをデバイスと異なる製造者のクラウドサービスに追加設定なしで接続できなければならない。

§5.1.2 Thing の機能

- WoT アーキテクチャでは、Thing が次のような機能を持てるようにすべきである。
  - Thing のステータス情報を読む。
  - 動作を発生させる Thing のステータス情報を更新する。
  - Thing のステータス情報の変更通知の登録、受信、登録解除を行う。
  - 入出力パラメータにより、特定の動作または計算を発生させる機能を呼び出す。

単なる状態遷移の報告だけでなく、一般的なイベントの通知の登録、受信、登録解除を行う。

§5.1.3 検索と発見

- WoT アーキテクチャでは、クライアントは、Thing自体にアクセスする前に、Thing の属性、機能、およびアクセスポイントを知ることができるべきでである。
- WoT アーキテクチャでは、クライアントはその属性と機能によって Thing を検索できるべきである。
- WoT アーキテクチャでは、機能の名前に関係なく、統一された語彙に基づいて必要な機能を提供する Thing のセマンティックな検索ができるべきである。

§5.1.4 記述方法

- WoT アーキテクチャは、Thing とその機能の記述を可能にする一般的な記述方法をサポートすべきである。
- このような記述は、人が読めるだけでなく、機械処理が可能であるべきである。
- このような記述により、その構造と記述内容のセマンティックアノテーションが可能となるべきである。

このような記述は、ウェブで一般的に用いられている複数の形式を用いてやり取りできるべきである。

§5.1.5 属性の記述

- WoT アーキテクチャでは、次のような Thing の属性を記述できるべきである。
  - 名前説
  - 明
  - 仕様のバージョン、形式、記述内容自体
  - 他の関係する Thing やメタデータ情報へのリンク
- このような記述は国際化をサポートすべきである。

§5.1.6 機能の記述

- WoT アーキテクチャでは、§ 5.1.2Thing の機能で示している Thing の機能を記述できるべきである。

§5.1.7 ネットワーク

- WoT アーキテクチャは、一般的に用いられている複数のウェブプロトコルをサポートすべきである。
- そのプロトコルには、次のようなものが含まれる。
  1. インターネットで一般的に用いられているプロトコル
  2. ローカルエリアネットワークで一般的に用いられているプロトコル
- WoT アーキテクチャでは、複数のウェブプロトコルを用いて同じ機能にアクセスできるべきである。
- WoT アーキテクチャでは、同じ Thing の機能に対して複数のプロトコルを組み合わせて使用できるべきである（例えば、HTTP と WebSocket）。

§5.1.8 デプロイメント

- WoT アーキテクチャは、同じモデルに基づいて、資源制限のあるエッジデバイスやクラウド上の仮想的なモノなど、様々なモノの性能をサポートすべきでである。
- WoT アーキテクチャは、ゲートウェイやプロキシなどの中間エンティティーを用いて、複数レベルのモノの階層をサポートすべきである。
- WoT アーキテクチャは、ネットワークアドレスの変換を考慮して、ローカルネットワークの外部（インターネットや別のローカルネットワーク）からローカルネットワーク内のモノへのアクセスをサポートすべきでである。

§5.1.9 アプリケーション

- WoT アーキテクチャでは、同じモデルに基づくウェブ標準技術を用いて、エッジデバイス、ゲートウェイ、クラウド、UI/UX デバイスなどの、様々なモノのためのアプリケーションを記述できるべきである。

§5.1.10 旧式技術への適合

> 翻訳者のメモ
>
> 5.1.10 項の英語原文タイトルは「Legacy Adoption」だが、その内容より「旧式技術への適合」について記述したものであると考えられるため、むしろ英語原文タイトルは「Legacy Adaptation」であるべきと考えられる。

- WoT アーキテクチャは、旧式の IP プロトコルや非IP プロトコルをウェブプロトコルにマッピングできるようにし、そのような旧式のプロトコルが終了および変換された場合の様々なトポロジーをサポートすべきである。
- WoT アーキテクチャは、既存の IP プロトコルが RESTful アーキテクチャに準拠している場合、そのプロトコルを変換せずに透過的な利用を可能とするべきである。
- WoT アーキテクチャは、デバイスやサービスに対してクライアントやサーバーの役割を強制してはならない。IoT デバイスは、システムアーキテクチャに応じて、クライアントまたはサーバー、あるいはその両方になりえる。エッジサービスとクラウドサービスにおいても同様である。

§5.2 技術要件

§ 4.2 共通パターンは、様々なユースケースを示し、アーキテクチャの構成要素を組み合わるためのパターンを列挙することにより、Web of Things の抽象アーキテクチャを定義したものである。この項では、抽象アーキテクチャから導かれた技術要件について説明する。

§5.2.1 Web of Things の構成要素と Web of Things アーキテクチャ

翻訳者のメモ

本 5.2.1 項の英語原文タイトルは「Components in the Web of Things and the Web of Things Architecture」であり、日本語訳は「Web of Things の構成要素と Web of Things アーキテクチャ」となるが、その記述内容は単に「一部のユースケースにおいてディレクトリが構成要素として用いられること」や「構成要素は一つの構成要素に接続される場合と、複数のネットワークに展開されることがある」ということであり、「構成要素と Web of Things アーキテクチャ」という項タイトルには違和感があるため、文書構成を見直すべきと考えられる。

ユースケースは、デバイスやアプリケーションにアクセスしたり、制御したりするデバイスやアプリケーション、デバイス間にあるプロキシ（例えば、ゲートウェイやエッジデバイス）などの基本的な構成要素を識別するのに役立つ。一部のユースケースで有用な追加的な構成要素は、発見を支援するディレクトリである。

これらの構成要素は、インターネットや、オフィス、工場、その他の施設のフィールドネットワークに接続される。関係するすべての構成要素が一つのネットワークに接続されている場合もあるが、一般的に、構成要素は複数のネットワークに展開される。

§5.2.2 デバイス

デバイスへのアクセスは、デバイスの機能とインターフェースの記述を用いて行われる。この記述を Thing Description（TD）と呼ぶ。Thing Description には、デバイスに関する一般的なメタデータ、機能を表す情報モデル、情報モデルで稼働するためのトランスポートプロトコルの記述、およびセキュリティ情報が含まれる。

一般的なメタデータには、デバイスの識別子（URI）、シリアル番号、製造日、場所などのデバイス情報、その他の人間が読める情報が含まれる。

情報モデルは、デバイスの属性を定義し、デバイスの内部設定、制御機能、通知機能を表す。同じ機能を持つデバイスは、用いるトランスポートプロトコルに関係なく、同じ情報モデルを有する。

> 翻訳者のメモ
>
> 英語原本中では「Information models defines device attributes」となっているが、主語が「Information models」と複数形であることを考慮すると、「define」が正しいと思われる。

WoT アーキテクチャに基づく多くのシステムは、システム領域を越えて利用されるため、関係者は、情報モデルで用いられる語彙とメタデータ（オントロジーなど）に共通の理解を持っているべきである。REST トランスポートに加えて、PubSub トランスポートもサポートされている。

> 翻訳者のメモ
>
> 上記日本語訳中で「システム領域を越えて」としている箇所は、英語原本中で「crossing system Domains」となっているが、「Domains」の頭文字が大文字である理由が不明であり、単に「domains」とすべきであると思われる。

セキュリティ情報には、認証、認可、安全な通信に関する記述が含まれる。デバイスは、TD をデバイスの内部か外部のいずれかに置き、TD をアクセス可能にして、他の構成要素がその TD を見つけてアクセスできるようにする必要がある。

§5.2.3 アプリケーション

アプリケーションは、メタデータ（記述）に基づいてネットワークとプログラムのインターフェースを生成し使
用できる必要がある。

アプリケーションはネットワークを通じてこの記述を取得できなければならないため、ネットワーク上で検索を
行って必要な記述を取得できる必要がある。

§5.2.4 デジタルツイン

デジタルツインは、メタデータ（記述）に基づいて内部でプログラムのインターフェースを生成し、そのプログラムのインターフェースを用いて仮想デバイスを表す必要がある。ツインは、仮想デバイス用の記述を作成し、それを外部で利用できるようにしなければならない。

仮想デバイスの識別子は新たに割り当てる必要があるため、元のデバイスとは異なる。これにより、仮想デバイスと元のデバイスが明確に別のエンティティーとして認識されるようになる。トランスポートとセキュリティのメカニズムと仮想デバイスの設定は、必要に応じて元のデバイスと異なる可能性がある。仮想デバイスは、ツインから直接提供される記述を持っているか、外部で利用できる記述を持っている必要がある。いずれの場合も、他の構成要素がデジタルツインに関係付けられるデバイスを見つけて利用できるように、記述を提供する必要がある。

§5.2.5 ディスカバリ

デバイス、アプリケーション、およびツインからデバイスと仮想デバイスの TD にアクセスするためには、TD を共有する共通の方法が必要である。ディレクトリは、デバイスとツイン自身が自動で、またはユーザが手動で記述を登録できる機能を提供することで、この要件を満たすことができる。

デバイスと仮想デバイスの記述は、外部エンティティーにより検索できる必要がある。ディレクトリは、デバイスの記述や情報モデルの一般的な記述に含まれているキーワードなどの検索キーを用いて検索を処理できる必要がある。

§5.2.6 セキュリティデバイスと仮想デバイスに関するセキュリティ情報は、デバイスの記述に記載する必要がある。これには、認証・認可およびペイロード暗号化に関する情報が含まれる。

WoT アーキテクチャは、Basic、Digest、Bearer、OAuth2.0 などの、ウェブで一般的に用いられている複数のセキュリティメカニズムをサポートすべきである。

§5.2.7 アクセシビリティ

Web of Things は、主に機械同士の通信を対象としている。関係のある人間は通常、Thing をアプリケーションに統合する開発者である。エンドユーザは、アプリケーションのフロントエンド、またはデバイス自身が提供する物理的なユーザインターフェースと対面する。どちらも W3C WoT 仕様の範囲外である。ユーザではなく IoT に焦点を当てていることから、アクセシビリティは直接的な要件ではないため、この仕様では扱っていない。

しかし、アクセシビリティには興味深い側面がある。上記の要件を満たすことで、機械はデバイスのネットワーク向け API を処理できる。アクセシビリティツールは、これを利用して、異なるモダリティのユーザインターフェースを提供できるため、物理的デバイスや IoT 関連のアプリケーションを用いる際の障壁が取り除かれる。

§6. WoT アーキテクチャ

この章は規定である。

4 章のユースケースに対処し、5 章の要件を満たすために、Web of Things (WoT) は、いわゆる Consumer が使用できるウェブの Thing – 通常は単に Thing と呼ばれる – の概念の上に構築される。この項では、W3C Web of Things の全体アーキテクチャを定義するための背景と規定的な言明を提供する。Web of Things は、様々な領域の利害関係者に対応するため、ウェブ技術の特定の側面、特にハイパーメディアの概念について詳しく説明する。

§6.1 概要

Thing は、物理的または仮想的なエンティティー（例えば、デバイスや部屋）の抽象化であり、標準化されたメタデータで記述される。W3C WoT では、記述メタデータは WoT Thing Description (TD) [WOT-THING-DESCRIPTION] でなければならない (MUST)。Consumer は、JSON [RFC8259] に基づく TD 表現形式を解析し処理できなければならない (MUST)。基礎となる情報モデルはグラフベースであり、そのシリアライゼーションは JSON-LD 1.1 [JSON-LD11] と互換性があるため、この形式は従来の JSON ライブラリか、JSON-LD プロセッサのいずれかで処理できる。TD の処理に JSON-LD プロセッサを用いると、RDF トリプルへの変換、セマンティックな推論、オントロジー用語に基づいて与えられたタスクの実行などの、セマンティックな処理がさらに可能になり、Consumer がより自律的に行動できるようになる。TD はインスタンス固有であり（つまり、Thing の種類ではなく個々の Thing を記述する)、デフォルトでは、外付けかつテキスト形式の Thing の（ウェブ）表現である。HTML ベースのユーザインターフェース、物理エンティティーの単なる画像、さらに、閉じたシステム内のウェブ以外の表現など、その他の Thing の表現が存在しえる (MAY)。

しかし、Thing であるためには、少なくとも一つの TD 表現が利用できなければならない (MUST)。WoT Thing Description は、Consumer が Thing の機能を発見して解釈し（セマンティックなアノテーションを介して)、Thing との相互作用時に様々な実装（例えば、様々なプロトコルやデータ構造）に適応できるようにする、機械が理解できる標準化された表現形式であり、それにより、様々な IoT プラットフォーム、つまり、様々なエコシステムや標準にまたがる相互運用が実現される。

図 15 Consumer と Thing の相互作用

Thing は、仮想エンティティーの抽象化にもなりえる。仮想エンティティーは、一つ以上の Thing の合成物である（例えば、いくつかのセンサーとアクチュエーターで構成される部屋）。合成物の一つのオプションは、仮想エ

ンティティーに関する機能のスーパーセットを含んだ一つの統合された WoT Thing Description を提供することである。合成物がかなり複雑な場合は、その TD は合成物内の下位階層にある Thing にリンクすることができる。中心的な TD はエントリーポイントとして機能し、一般的なメタデータのみ、そして、場合によっては包括的な機能を含む。これにより、より複雑な Thing の特定の側面をグループ化できる。

リンクは、階層的な Thing に対してのみでなく、Thing とその他の資源との一般的な関係にも適用される。リンク関係型は、例えば、照明を制御するスイッチや、モーションセンサーで監視している部屋など、Thing 間の関連性を表現する。Thing に関連付けられるその他の資源には、マニュアル、予備の部品のカタログ、CAD ファイル、GUI や、その他のウェブ上の文書がある。全体として、Thing の間のウェブリンクにより、人間と機械の両

方が Web of Things をたどっていけるようになる。これは、利用可能な Thing のカタログを管理する Thing Directory を提供する（通常は、TD 表現をキャッシュすることによる）ことでさらに容易になる。要約すると、モノのウェブを形成するために、WoT Thing Description を他の Thing やウェブ上の他の資源にリンクしてもよい（MAY）。

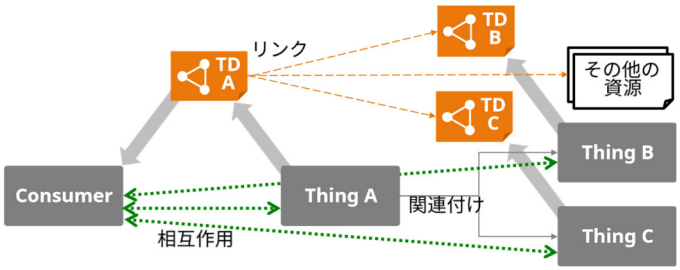

図 16 リンクされた Thing

Thing の WoT インターフェースであるネットワーク向けインターフェースを介した相互作用を実現するためには、ソフトウェアスタックを備えたネットワーク化されたシステム構成要素上で Thing をホストしなければならない。この例の一つは、Thing の抽象化の背後にある物理エンティティーのインターフェースとなっている、センサーとアクチュエーターを備えた組み込みデバイスで実行されている HTTP サーバーである。しかし、W3C WoT は、Thing を提供する場所を強制しておらず、IoT デバイスに直接置いたり、ゲートウェイなどのエッジデバイスや、クラウドに置くことができる。

典型的なデプロイメントの課題は、通常、IPv4 ネットワークアドレス変換（NAT）やファイアウォールデバイスによりインターネットからローカルネットワークに到達できないというシナリオである。この状況を改善するために、W3C WoT は Thing と Consumer の間に Intermediary を認めている。

Intermediary は Thing のプロキシとして機能できる。その場合、Intermediary は、元の Thing と似た WoT Thing Description を持っているが、それは Intermediary が提供する WoT インターフェースを指し示す。

Intermediary は、既存の Thing に機能を追加したり、複数の利用可能な Thing から新しい Thing を構成し、それにより仮想エンティティーを形成することもできる。Intermediary は、WoT Thing Description を持ち、WoT

インターフェースを提供するため、ウェブ［REST］のような階層化したシステムアーキテクチャの Thing と見分けがつかない場合があり、Consumer には Thing と同じに見える。WoT Thing Description の識別子は、同じ元の Thing または最終的に一意の物理エンティティーを表す複数の TD を関連づけることができなくてはならない（MUST）。

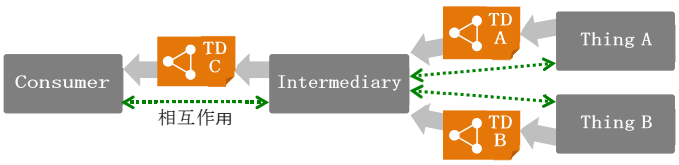

図 17 Intermediary

制限のあるローカルネットワークの別の改善策は、WoT インターフェースを、ローカルネットワーク内の Thing から公開されている到達可能な Consumer への接続を確立するプロトコルにバインドすることである。

Thing を Consumer と束ねて、Thing と Thing で相互作用することができる（MAY）。通常、Consumer の振る舞いはソフトウェアの構成要素に埋め込まれており、それは Thing の振る舞いも実装している。Consumer の振る舞いの設定は、Thing を通じて公開することができる（MAY）。

W3C WoT の概念は、デバイスレベル、エッジレベル、クラウドレベルという、IoT アプリケーションに関連するすべてのレベルに適用できる。これにより、様々なレベルで共通のインターフェースと API が促進され、Thing と Thing、Thing とゲートウェイ、Thing とクラウド、ゲートウェイとクラウド、さらにはクラウドフェデレーション、つまり、IoT アプリケーションに関する二つ以上のサービス提供者のクラウドコンピューティング環境の相互接続などの、様々な統合パターンが可能になる。図 18 は、§ 4.3 要約でまとめたユースケースに対処するために、上記で紹介した WoT の概念を適用して組み合わせる方法の概要を示している。

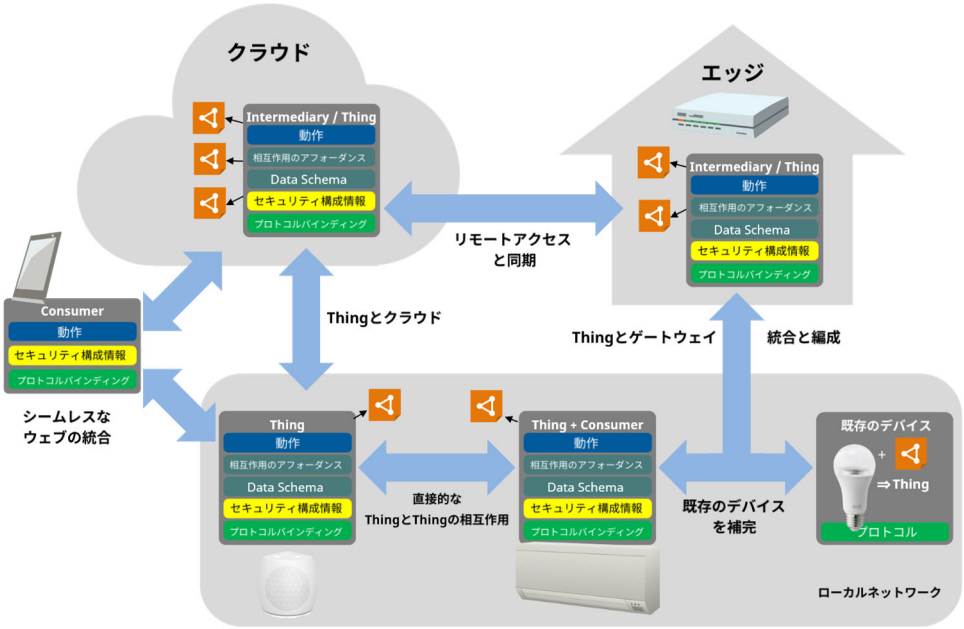

図18 W3C WoTの抽象アーキテクチャ

§6.2 アフォーダンス

W3C WoT の中心を成すのは、機械が理解できるメタデータ（つまり、）の提供である。理想的には、このようなメタデータは自己記述的であり、Thing がどのような機能を提供するかと、その提供された機能をどのように用いるかを Consumer が識別できる。この自己記述性の鍵は、アフォーダンスの概念にある。

アフォーダンスという用語は、生態心理学に由来するが、「『アフォーダンス』とは、物の知覚された実際の特性、主に、物がどのように利用されうるかを決定する基本的な特性を指す」という Donald Norman の定義に基づくヒューマンコンピューターインタラクション［HCI］の分野で採用された。［NORMAN］

これに関する例は、取っ手のあるドアである。ドアの取っ手はアフォーダンスであり、ドアを開くことができることを示唆する。人間にとって、ドアの取っ手は通常、どのようにドアが開くかも示す。アメリカのノブは、回転することを示唆し、ヨーロッパのレバーハンドルは押し下げることを示唆する。

REST アーキテクチャスタイル［REST］の中核となる基盤の一つであるハイパーメディアの原則は、ウェブをナビゲートし、ウェブアプリケーションを制御する方法に関する明確な知識を情報の利用者が得られるように、ウェブで利用できる情報を他の情報にリンクすることを求めている。ここでは、情報と制御の同時表示（ハイパーリンクの形式で提供）は、ウェブクライアントにウェブアプリケーションを動作させる手段を提供する（afford）メカニズムである。このコンテキストでは、アフォーダンスはハイパーリンクの記述であり（例えば、リンク関係型とリンクターゲット属性を介した）、ナビゲート方法を提案したり、リンクされている資源における動作方法をウェブクライアントに提案する可能性がある。したがって、リンクはナビゲーションアフォーダンスを提供する。

このハイパーメディアの原則から導き出された Web of Things では、相互作用のアフォーダンスを、可能な選択肢を Consumer に示して記述した Thing のメタデータと定義しており、それによって Consumer が Thing と相互作用を行うことができる方法を提案する。一般的な相互作用のアフォーダンスはナビゲーションであり、これ

はリンクをたどることで作動し、それによって Consumer が Web of Things を閲覧できるようになる。§ 6.4 相互作用モデルは、3 種類の W3C WoT の相互作用のアフォーダンスである Property、Action、Event を定義している。

全体として、この W3C WoT の定義は、物理的なモノを作成する HCI とインタラクションデザイナー、およびウェブサービス全般に取り組んでいる REST とマイクロサービスコミュニティと連携を図っている。

§6.3 Web Thing

Web Thing には、図 19 に示しているように、その動作、その相互作用のアフォーダンス、そのセキュリティ設定、そのプロトコルバインディングという四つのアーキテクチャの側面がある。Thing の動作の側面には、自律的な動作と相互作用のアフォーダンスのハンドラーの両方が含まれる。相互作用のアフォーダンスは、特定のネットワークプロトコルやデータエンコーディングを参照せずに、抽象操作を通じてどのように Consumer が

Thing と相互作用できるかのモデルを提供する。プロトコルバインディングは、個々の相互作用のアフォーダンスを特定のプロトコルの具体的なメッセージにマッピングするために必要な詳細を追加する。一般的に、様々な具体的なプロトコルを用いて、一つの Thing の中であっても、相互作用のアフォーダンスの様々なサブセットをサポートできる。Thing のセキュリティ設定の側面は、相互作用のアフォーダンスへのアクセスと、関連すると

Private Security Data の管理を制御するために用いられるメカニズムを表す。

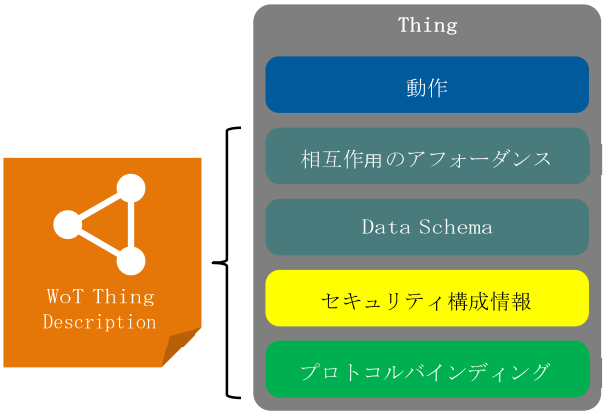

図19 Thingのアーキテクチャの側面

§6.4 相互作用モデル

当初、ウェブ資源とは通常、ウェブクライアントが容易に取得できるウェブ上の文書を表していた。ウェブサービスの導入により、資源は、あらゆる種類の動作を実装できる、より汎用的な相互作用のエンティティーとなった。この非常に高度な抽象化により、多種多様な相互作用の可能性があるため、アプリケーションと資源との間を疎結合することが難しくなっている。その結果、執筆時点で、一般的なAPI記述は、アプリケーションの意図（application intent）から資源のアドレス、メソッド、リクエストペイロード構造、応答ペイロード構造、および予期されるエラーへの静的マッピングで構成されている。これにより、ウェブクライアントとウェブサービスが密結合されることとなる。

W3C WoTの相互作用モデルは、アプリケーションの意図（application intent）から具体的なプロトコル操作へのマッピングを形式化する仲介的な抽象化を導入し、相互作用のアフォーダンスをどのようにモデル化するかの可能性を絞り込む。

ナビゲーションのアフォーダンス（つまり、ウェブリンク）に加えて、Thingは、この仕様で定義しているProperty、Action、Eventという他の3種類の相互作用のアフォーダンスを提供できる（MAY）。このナローウエスト（narrow waist）は、ConsumerとThingの分離を可能とする一方で、これら4種類の相互作用のアフォーダンスにより、IoTのデバイスおよびサービスで見られるほぼすべての相互作用の可能性をモデル化することができる。

§6.4.1 Property

Propertyは、Thingの状態を公開する相互作用のアフォーダンスである。Propertyによって公開される状態は、検索可能（読み取り可能）でなければならない（MUST）。オプションでPropertyによって公開される状態は、更新可能（書き込み可能）である（MAY）。Thingは、変更後に新しい状態をプッシュすることにより、

Propertyを監視可能にすることを選択できる（MAY）（資源の監視［RFC7641］を参照）。書き込み専用の状態は、

Actionを介して更新すべきである。

プロトコルバインディングを用いているが、データが完全には指定されていない場合（例えば、メディアタイプにより）、Propertyには公開の状態に関するデータスキーマを一つ含めることができる（MAY）。

Propertyの例は、センサー値（読み取り専用）、ステートフルなアクチュエーター（読み書き）、設定パラメータ

（読み書き）、モノの状態（読み取り専用または読み書き）、または計算結果（読み取り専用）である。

§6.4.2 Action

Action は、Thing の機能を呼び出すことができる相互作用のアフォーダンスである。Action は、直接公開されていない状態を操作したり（Property を参照）、一度に複数の Property を操作したり、内部ロジックに基づいて Property を操作したりすることができる（MAY）（例えば、トグル）。Action の呼び出しは、経時的に状態（アクチュエータを介した物理的な状態を含む）を操作する Thing のプロセスを始動させることもできる（MAY）。

用いるプロトコルバインディングでデータが完全に指定されていない場合（例えば、メディアタイプにより）、Action にはオプションの入力パラメータと出力結果のデータスキーマを含めることができる（MAY）。

Action の例は、複数の Property を同時に変更すること、照明の明るさを暗くしたり（減光）、独自の制御ループアルゴリズムなどの非公開なプロセスを用いるなどして経時的に Property を変更すること、文書の印刷などの

長時間にわたるプロセスを呼び出すことである。

§6.4.3 Event

Event は、Thing から利用者にデータを非同期的にプッシュする出来事の発生源を記述する相互作用のアフォーダンスである。ここでは状態ではなく、状態の遷移（つまり、イベント）が伝達される。Event は、Property として公開されていない条件によって始動させることができる（MAY）。

使用されるプロトコルバインディングによる（例えば、メディアタイプを通じての）データの指定が完全には行われていない場合、Event には、その出来事に関するデータのためのデータスキーマ、および（例えば、

Webhook コールバック URI により登録するために）可能な登録制御メッセージを含めることができる（MAY）。

Event の例は、アラームや時系列のサンプルなどの、定期的にプッシュされる離散的な出来事である。

§6.5 ハイパーメディア制御

ウェブでは、アフォーダンスは情報と制御の同時表示であり、その情報はユーザが選択肢を取得するアフォーダンスになる。人間にとって、その情報は通常、ハイパーリンクを記述または装飾しているテキストや画像である。制御は、少なくともターゲット資源の URI が含まれているウェブリンクで、ウェブブラウザで逆参照できる（つまり、リンクをたどることができる）。しかし、さらにウェブリンクが関係型とターゲット属性で記述されている場合、機械は意味のある方法でリンクをたどることもできる。ハイパーメディア制御は、どのようにアフォーダンスを作動させるかに関する機械が理解可能な記述である。ハイパーメディア制御は通常、ウェブサーバーから発信され、ウェブクライアントがそのウェブサーバーと相互作用を行っている間にインバンド（in-band）で見つけられる。このようにして、現在の状態や認証などの他の要因を考慮に入れることにより、ウェブサーバーはウェブアプリケーションを通じてクライアントを動的に駆動できる。これは、クライアントにプレインストール、またはハードコーディングする必要があるアウトオブバンド（out-of-band）のインターフェースの記述とは対照的である（例えば、RPC、WS-* ウェブサービス、固定の URI メソッド応答定義を持つ HTTP サービス）。

W3C WoT は、ウェブをナビゲートするための確立した制御であるウェブリンク［RFC8288］と、あらゆる種類の操作を可能にするより強力な制御としてのウェブフォームという 2 種類のハイパーメディア制御を用いる。リンクは既に、CoRE リンク形式［RFC6690］、OMA LWM2M［LWM2M］、OCF［OCF］などの他の IoT 標準や IoT プラットフォームで用いられている。フォームは、W3C WoT 以外に、IETF で定義されている制約付き RESTful アプリケーション言語（CoRAL）［CoRAL］でも導入されている新しい概念である。

§6.5.1 リンク

リンクにより、Consumer（または広義ではウェブクライアント）は、コンテキストとリンクターゲットの関係に応じて、現在のコンテキスト（ウェブブラウザで現在表示されている資源の表現など）を変更したり、追加の資源を現在のコンテキストに含めたりすることができる。Consumer は、ターゲット URI を逆参照することで、つまりリンクをたどって資源の表現を取得することでこれを行う。

> 翻訳者のメモ
>
> 英語原本中で「cf. the set of resource representations currently rendered in the Web browser」となっている箇所は、直前の「current context」の例示であると思われるため、「cf.」というよりはむしろ

> 「e.g.」（例えば）という形で訳した。

W3C WoT は、Web Linking [RFC8288] の定義に従っており、リンクは次のもので構成される。

- リンクのコンテキスト
- 関係型
- リンクターゲット
- オプションで、ターゲット属性

リンク関係型は、ABNF [RFC5234] LOALPHA * (LOALPHA / DIGIT / "." / "-") （例えば、stylesheet）に準拠していなければならない IANA [IANA-RELATIONS] に登録されている定義済みトークンか、URI の形式の拡張型 [RFC3986] のいずれかである。拡張関係型は、大文字と小文字を区別しない比較方法により、文字列として比較しなければならない (MUST)。 （異なる形式でシリアル化されている場合は、URI に変換すべき)。それにも関わらず、拡張関係型には、すべて小文字の URI を用いるべきである (SHOULD)。 [RFC8288]

Web of Things では、リンクは発見に用いたり、Thing 間の関係（例えば、階層的か、機能的か）やウェブ上の他の文書（例えば、マニュアルや、CAD モデルなどの代替表現）との関係を表したりするために用いる。

§6.5.2 フォーム

フォームにより、Consumer（または広義のウェブクライアント）は、URI の解決を超える操作を実行できる（例えば、Thing の状態を操作するなど)。Consumer は、フォームに記入して、その送信ターゲットに送信することでこれを行う。これには通常、リンクが提供できるよりも詳細な（リクエスト）メッセージの内容に関する情報が必要である（例えば、メソッド、ヘッダフィールド、またはその他のプロトコルのオプション)。フォームはリクエストテンプレートと考えることができ、提供者は、独自のインターフェースと状態に従って情報の一部を事前に入力し、Consumer（または、一般的にはウェブクライアント）が入力する部分を空白のままにする。

W3C WoT は、フォームを新しいハイパーメディア制御と定義している。CoRAL の定義は実質的に同じであり、したがって互換性があることに注意すること [CoRAL] 。フォームは次のもので構成される。

- フォームのコンテキスト操作
- 型
- 送信ターゲット
- リクエストメソッド
- オプションでフォームフィールド

フォームは、「あるフォームのコンテキストに関する操作型の操作を実行するために、送信ターゲットにリクエストメソッドのリクエストを発信せよ」という表明と考えることができ、オプションのフォームフィールドで、必要なリクエストをさらに記述できる。

フォームのコンテキストと送信ターゲットは、両方とも国際化資源識別子 (IRI;Internationalized Resource

Identifier）[RFC3987] でなければならない（MUST）。しかし、一般的なケースでは、多くのプロトコル（HTTP など）が IRI をサポートしていないため、それらは URI [RFC3986] になることもある。

フォームのコンテキストと送信ターゲットは、同じ資源または異なる資源を指し示すことができ（MAY）、その場合、送信ターゲットの資源はコンテキストの操作を実装する。

操作型は、操作のセマンティクスを示す。操作型は、リンク関係型と同様の形で示される。

- よく知られた（well-known）操作型は、ABNF LOALPHA * (LOALPHA / DIGIT / "." / "-") に従わなければならない（MUST）。よく知られた操作型は、大文字と小文字を区別しない比較により比較されなければならない（MUST）。この仕様で定義している Web of Things のよく知られた操作型を表 1 に示す。

- 事前に定義されている操作型は、アプリケーションが選択した拡張操作型によって拡張できる（MAY）。拡張操作型は、型を一意に識別する URI [RFC3986] でなければならない（MUST）。拡張操作型は、大文字と小文字を区別しない比較により、文字列として比較されなければならない（MUST）。それにも関わらず、拡張操作型には、すべて小文字の URI を用いるべきである（SHOULD）。

リクエストメソッドでは、送信ターゲットの URI スキームで識別されるプロトコルでの標準的なメソッド集合のうちの一つを特定しなければならない（MUST）。

フォームフィールドはオプションであり、特定の操作に期待されるリクエストメッセージをさらに指定できる（MAY）。これはペイロードに限定されず、プロトコルのヘッダにも影響する可能性があることに注意すること。フォームフィールドは、URI スキームで指定されている送信ターゲットに用いられるプロトコルに依存することもある（MAY）。例えば、HTTP ヘッダフィールド、CoAP オプション、リクエストペイロードのパラメータ（つまり、完全なコンテンツタイプ）などのプロトコルに依存しないメディアタイプ [RFC2046]、または期待される応答に関する情報などである。

表 1 Web of Things のよく知られた操作型

操作型	説明
readproperty	対応するデータを検索するための、Property のアフォーダンスに関する読み取り操作
writeproperty	対応するデータを更新するための、Property のアフォーダンスに関する書き込み操作
observeproperty	Property が更新されたときに新しいデータにより通知を受けるための、Property のアフォーダンスに関する監視操作
unobserveproperty	対応する通知を停止するための、Property のアフォーダンスに関する監視解除操作
invokeaction	対応する Action を実行するための、Action のアフォーダンスに関する呼び出し操作
subscribeevent	Event が発生したときに Thing によって通知を受けるための、Event のアフォーダンスに関する登録操作
unsubscribeevent	対応する通知を停止するために、Event のアフォーダンスに関する登録解除操作
readallproperties	すべての Property のデータを 1 回の相互作用で検索するための、Thing に関する readallproperties（すべての Property の読み込み）操作
writeallproperties	すべての書き込み可能な Property のデータを 1 回の相互作用で更新するための、Thing に関する writeallproperties（すべての Property の書き込み）操作
readmultipleproperties	選択したプロパティーのデータを 1 回の相互作用で検索するための、Thing に関する readmultipleproperties（複数の Property の読み込み）操作
writemultipleproperties	選択した書き込み可能な Property のデータを 1 回の相互作用で更新するための、Thing に関する writemultipleproperties（複数の Property の書き込み）操作

編集者のメモ

本仕様の執筆時点では、（上記の）よく知られた操作型は、WoT 相互作用モデルに由来する一定の集合である。他の仕様では、それぞれの文書形式やフォームのシリアライゼーションにとって有効な、よく知られた操作型を追加定義してもよい。本仕様の今後のバージョンや別の仕様では、将来、WoT 関連仕様の範囲を越えて適用される可能性のある拡張やより汎用的な Web フォームモデルを可能にするために IANA レジストリを設定してもよい。

§6.6 プロトコルバインディング

プロトコルバインディングは、相互作用のアフォーダンスから、HTTP [RFC7231] 、CoAP [RFC7252] 、

MQTT［MQTT］などの特定プロトコルの具体的なメッセージへのマッピングである。これは、ネットワークに接続するインタフェースを介して相互作用のアフォーダンスをどのように作動させるかを Consumer に示す。プロトコルバインディングは、相互運用性をサポートするために REST［REST］の統一インターフェース

(Uniform Interface) 制約に従う。したがって、すべての通信プロトコルが W3C WoT のプロトコルバインディングの実装のために適格なわけではない。なお、要件は以下の言明で示す通り。

§ 6.2 アフォーダンスで示したドアの例では、プロトコルバインディングは、ノブかレバーかのレベルでドアの取っ手に対応しており、それは、ドアがどのように開くかを示している。

§6.6.1 ハイパーメディア駆動相互作用のアフォーダンスには、一つ以上のプロトコルバインディングが含まれていなければならない

(MUST)。相互作用のアフォーダンスを作動させる方法を自己記述的にするために、プロトコルバインディングをハイパーメディア制御（§ 6.5 ハイパーメディア制御を参照）としてシリアル化しなければならない

(MUST)。ハイパーメディア制御は、対応する相互作用のアフォーダンスを提供している Thing を管理する発信元から発信されなければならない (MUST)。その発信元は、Thing自体である場合もあり、実行時に（現在の状態に基づいて、IP アドレスなどのネットワークパラメータを含めて）TD ドキュメントを生成するか、最新のネットワークパラメータのみが挿入されたメモリ上の情報から TD ドキュメントを提供する。その発信元は、ネットワークパラメータや内部構造（例えば、ソフトウェアスタック）を含む、Thing の完全で最新の知識を持つ外部エンティティーにもなりえる。これにより、Thing と Consumer の間の疎結合が可能になり、独立したライフサイクルと進化が可能になる。ハイパーメディア制御は、Thing の外部でキャッシュされ、鮮度を判断するためにメタデータのキャッシュが利用できる場合にはオフライン処理に使用される (MAY)。

翻訳者のメモ

上記で言う「発信元」に関して、§ 6.5 ハイパーメディア制御で以下の通り触れられていることから、原文における The hypermedia controls MUST originate from the authority managing the Thing that is

providing the corresponding Interaction Affordance. 中の authority は「ハイパーメディア制御の発信元」を意味しており、通常、ウェブサーバーである。

> ハイパーメディア制御は通常、ウェブサーバーから発信され、ウェブクライアントがサーバーと相互作
> 用を行っている間にインバンド（in-band）で発見される。このようにして、ウェブサーバーは、現在の状態や承認などの他の要因を考慮に入れることにより、ウェブアプリケーションを通じてクライアントを動的に駆動できる。

§6.6.2 URI

W3C WoT の条件を満たしているプロトコルは、IANA に登録されている関連する URI スキーム［RFC3986］を持たなければならない (MUST)（[IANA-URI-SCHEMES] を参照）。ハイパーメディア制御は、URI［RFC3986］に依拠してリンクと送信ターゲットを識別する。これにより、URI スキーム（「:」までの最初の構成要素）は、Thing との相互作用のアフォーダンスに用いる通信プロトコルを識別する。W3C WoT は、これらのプロトコルを転送プロトコルと呼ぶ。

§6.6.3 メソッドの標準的な集合

W3C WoT の条件を満たしているプロトコルは、先験的に知られている標準的なメソッドの集合に基づいていなければならない (MUST)。標準的なメソッドの集合は、例えばプロキシにより相互作用のアフォーダンスの中間処理を可能にしたり、プロトコルバインディング［REST］間での変換を行うために、メッセージを自己記述的なものにする。さらに、これにより、Consumer は、当該 Consumer 向けの Thing 固有のコードやプラグインを回避しながら、HTTP、CoAP、MQTT などの一般的な転送プロトコルの、再利用可能なプロトコルスタックを持てるようになる。

相互作用のアフォーダンスを作動させたときに交換されるすべてのデータ（別名、コンテンツ）は、プロトコルバインディングのメディアタイプ［RFC2046］により識別されなければならない（MUST）。メディアタイプは、例えば、JSON の application/json［RFC8259］または CBOR の application/cbor［RFC7049］などの、表現形式を識別するためのラベルである。メディアタイプは IANA によって管理されている。

一部のメディアタイプでは、用いる表現形式を完全に指定するために追加のパラメータが必要になる場合がある。例は、text/plain; charset=utf-8 や application/ld+json; profile="http://www.w3.org/ns/jsonld#compacted"である。これは、Thing に送信されるデータを記述する際に特に考慮する必要がある。HTTP の content coding［RFC7231］などのデータに関する標準的な変換も存在するかもしれない。プロトコルバインディングは、メディアタイプのみよりも詳細に表現形式を指定する追加情報を持つことができる（MAY）。

多くのメディアタイプは、その要素に追加的なセマンティクスを提供しない汎用的なシリアライゼーション形式のみを識別することに注意すること（例えば、XML、JSON、CBOR）。したがって、対応する相互作用のアフォーダンスは、交換されるデータにより詳細な構文メタデータを提供するデータスキーマを宣言すべきである（SHOULD）。

§6.7 WoT システム構成要素とその相互接続性

§ 6.1 概要の節では、Thing、Consumer、Intermediary などの抽象的な WoT アーキテクチャの構成要素の観点から WoT アーキテクチャについて説明した。これらの抽象的な WoT アーキテクチャの構成要素がソフトウェアスタックとして実装され、WoT アーキテクチャで特定の役割を担う場合、そのようなソフトウェアスタックを Servient と呼ぶ。WoT アーキテクチャに基づくシステムには Servient が含まれ、システムの目標を達成するために相互に通信を行う。

この節では、システム構成図を用いて、複数の Servient が連携して WoT アーキテクチャに基づくシステムを構築する方法を説明する。

Thing は、Servient によって実装できる。Thing の中には、Servient のソフトウェアスタック内に公開された Thing と呼ばれる Thing の表現が含まれており、その WoT インターフェースを Thing の Consumer が利用できるようにする。この公開された Thing は、モノの動作を実装するために、Servient 上のその他のソフトウェア構成要素（例えば、アプリケーション）により使用することができる。

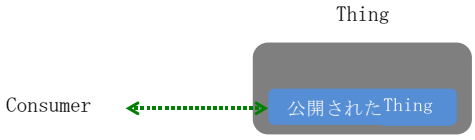

図 20 Thing としての Servient

一方、Consumer は常に Servient によって実装される。これは、（Consumer は）Thing Description（TD）形式を処理できる必要があるとともに、TD に含まれるプロトコルバインディング情報を通じて設定できるプロトコルスタックを持つ必要があるためである。

Consumer の中では、Servient のソフトウェアスタックが、利用される Thing と呼ばれる Thing の表現を提供し、Thing と相互作用するために TD を処理する必要がある、Servient 上で実行されるアプリケーションから利用できるようにする。

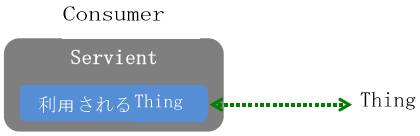

図21 ConsumerとしてのServient

Servient のソフトウェアスタック内の利用される Thing のインスタンスは、アプリケーションからプロトコルレベルの複雑性を分離するのに役立つ。これは、アプリケーションに代わって、公開される Thing と通信を行う。

同様に、Intermediary はさらに、Servient によって実装される別の WoT アーキテクチャの構成要素である。Intermediary は、Thing とその Consumer の間に位置し、(Thing に対する) Consumer と (Consumer に対する) Thing の両方の役割を果たす。Intermediary の中では、Servient のソフトウェアスタックに、Consumer (利用される Thing) と Thing (公開される Thing) の両方の表現が含まれている。

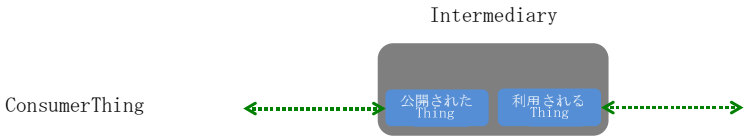

図 22 Intermediary としての Servient

§6.7.1 直接通信

図 23 は、Thing Description を介して相互作用のアフォーダンスを公開している Thing と、相互作用のアフォーダンスにより を用いる Consumer との間の直接通信を示している。両方の Servient が同じネットワークプロトコルを用い、相互にアクセスできる場合に、直接通信が適用される。

図 23 Consumer と Thing の高レベルなアーキテクチャ

公開された Thing は、Thing の抽象化のソフトウェア表現であり、Thing が提供する相互作用のアフォーダンスの WoT インターフェースを提供する。

利用される Thing は、Consumer によって利用されているリモートの Thing のソフトウェア表現であり、アプリケーションに対するリモートのモノへのインターフェースとして機能する。Consumer は、TD ドキュメントを解析して処理することにより、利用される Thing のインスタンスを生成できる。Consumer と Thing との相互作用は、利用される Thing と公開された Thing がそれらの間の直接ネットワーク接続を介してメッセージを交換することによって実行される。

§6.7.2 間接通信

図 24 では、Consumer と Thing が Intermediary を介して互いに接続されている。Intermediary は、Servient 同

士が異なるプロトコルを用いている場合や、認証が必要でアクセス制御（例えば、ファイアウォール）を提供している異なるネットワーク上にある場合に必要である。

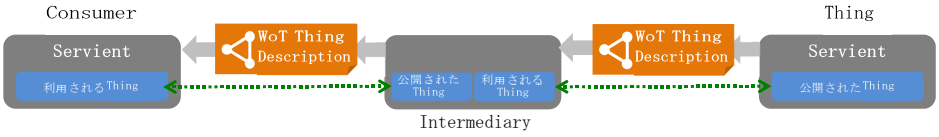

図 24 Intermediary を用いた高レベルなアーキテクチャ

Intermediary は、公開された Thing と利用される Thing の機能を組み合わせる。Intermediary の機能には、Consumer と Thing との間で相互作用のアフォーダンスのメッセージを中継すること、オプションで、応答を高速化するために Thing のデータをキャッシュすること、Intermediary によって Thing の機能が拡張されたときに通信を変換することが含まれる。Intermediary の中では、利用される Thing が、Thing の中の公開された Thing のプロキシオブジェクトを作成しており、Consumer は、自身の利用される Thing を通じて、そのプロキシオブジェクト（つまり、Intermediary の中の、公開された Thing）にアクセスできる。

Consumer と Intermediary は、Intermediary と Thing の間で用いられるプロトコルとは異なるプロトコルで通信できる。例えば、Intermediary は、CoAP を用いる Thing と、HTTP を用いる Consumer のアプリケーションとの間の橋渡しを提供できる。

Intermediary と Thing の間で、複数の異なるプロトコルが用いられている場合でも、Consumer は、Intermediary を通じ、ある一つのプロトコルを用いてそれらの Thing と間接的に通信できる。認証についても同じことが言える。Consumer の、利用される Thing は、ある一つのセキュリティメカニズムを用いて、Intermediary の、公開された Thing と認証を行う必要があるが、Intermediary が、複数の異なる Thing による認証を行うためには、複数のセキュリティメカニズムが必要となる場合がある。

通常、Intermediary は、発信元である Thing の、Thing Description に基づいて、そのプロキシオブジェクト用の Thing Description を生成する。各ユースケースごとの要件に応じて、プロキシオブジェクト用の TD は、元の Thing の TD と同じ識別子を用いるか、新たな識別子が割り当てられる。必要に応じて、Intermediary によって生成された TD には、他の通信プロトコルのインターフェースを含むことができる（MAY）。

§7. WoT 構成要素

この章は規定である。

Web of Things（WoT）の構成要素により、WoT の抽象アーキテクチャに準拠したシステムを実装できる。この構成要素の詳細は、別の仕様で定義しており、この章では、概要と要約を示す。

WoT 構成要素は、§ 6.3 Web Thing で論じ、図 19 で示している Thing の各アーキテクチャの側面をサポートする。個々の構成要素を図 25 の抽象的な Thing のコンテキストで示している。これは抽象的な図であり、特定の実装を表しているものではない。代わりに、構成要素と Thing の主要なアーキテクチャの側面との関係を示している。この図では、WoT 構成要素を黒の輪郭線で強調表示している。分野横断的な関心事であるセキュリティ

は、公開されている構成要素と保護されている非公開の構成要素とに分けている。WoT スクリプティング API はオプションであり、バインディングテンプレートは参考情報である。

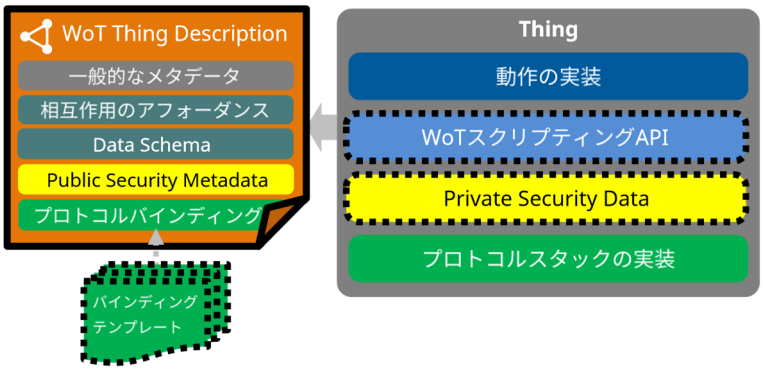

図 25 Thing のアーキテクチャの側面に対する WoT 構成要素の関係

以下の節では、WoT Thing Description、WoT バインディングテンプレート、WoT スクリプティング API という個々の WoT 構成要素に関する追加情報を提供する。セキュリティは、分野横断的な関心事であるが、4 番目の構成要素と考えることができる。

§7.1 WoT Thing Description

WoT Thing Description（TD）仕様［WOT-THING-DESCRIPTION］は、セマンティックな語彙に基づく情報モデルと JSON に基づくシリアル化表現を定義している。TD は、人間が読めて機械が理解できる方法で Thing に豊かなメタデータを提供する。生の JSON としての処理に加えて、実装で JSON-LD［JSON-LD11］とグラフデータベースを使用してメタデータの強力なセマンティック処理を行えるように、TD の情報モデルと表現形式は、どちらもリンクトデータ［LINKED-DATA］に準じている。

Thing Description（TD）は、名前、ID、内容記述などの一般的なメタデータで Thing のインスタンスを記述し、関係する Thing やその他の文書へのリンクを介して関連メタデータを提供することもできる。TD には、

§ 6.4 相互作用モデルで定義している相互作用モデルに基づく相互作用のアフォーダンスのメタデータ、Public

Security Metadata と、プロトコルバインディングを定義している通信メタデータも含まれている。TD は、提供されているサービスと関連資源（どちらもハイパーメディアコントロールを使用して記述される）について知るためのエントリポイントを提供するため、Thing の index.html と考えることができる。

理想的には、TD は、Thing自体によって作成および（または）提供され、ディスカバリ時に取得される。しかし、Thing に資源の制限がある場合（例えば、メモリ空間や電力が限られている場合）や、Web of Things の一

部となるように既存のデバイスが改造されている場合には、外部で提供することもできる。ディスカバリを改善し（例えば、制約のあるデバイスに対する）デバイス管理を容易にするための一般的なパターンは、TD をディレクトリに登録することである。Consumer は、TD のキャッシングメカニズムを、Thing が更新されて新しいバージョンの TD を取得する必要がある場合に通知してくれる通知メカニズムと組み合わせて用いることが推奨される。

セマンティックな相互運用性のために、TD は領域固有の語彙を利用でき、その語彙には明示的な拡張ポイントが提供される。しかし、特定の領域固有の語彙の開発は現在、W3C WoT 標準化活動の範囲外である。

有用でありえる外部 IoT 語彙の三つの例は、SAREF［SAREF］、Schema Extensions for IoT［IOT-SCHEMA-ORG］、および W3C Semantic Sensor Network Ontology［VOCAB-SSN］である。TD におけるこのような外部語彙の使用はオプションである。将来的に、追加の領域固有の語彙が開発され、TD で用いられるかもしれない。

> 翻訳者のメモ
>
> 英語原本において、「W3C Semantic Sensor Network ontology」となっている箇所は、上記の通り「W3C Semantic Sensor Network Ontology」が正しいと思われる。

全体として、WoT Thing Description の構成要素は、二つの方法で相互運用性を促進する。まず、TD は、Web of Things における機械間通信を可能にする。次に、TD は、IoT デバイスにアクセスしてそのデータを利用できるアプリケーションを作成するために必要なすべての詳細情報を開発者が文書化し、取得するための共通の統一形式として機能する。

§7.2 WoT バインディングテンプレート

この節は参考情報である。

すべてのコンテキストに適した単一のプロトコルはないため、IoT はデバイスへのアクセスに様々なプロトコルを用いる。したがって、Web of Things の中心的な課題は、多くの様々な IoT プラットフォーム（例えば、

OCF、oneM2M、OMA LWM2M、OPC UA）と、特定の標準には準拠していないけれども適切なネットワークプロトコルを介して適格なインターフェースを提供するデバイスとの相互作用を可能にすることである。WoT は、プロトコルバインディングを通じてこの多様性に取り組んでおり、これは様々な制約を満たさなければならない（§ 6.6 プロトコルバインディングを参照）。

非規定的な WoT バインディングテンプレート仕様［WOT-BINDING-TEMPLATES］は、様々な IoT プラットフォームと相互作用を行う方法に関するガイダンスを提供する通信メタデータの青写真のコレクションを提供する。

特定の IoT デバイスやサービスを記述する場合に、対応する IoT プラットフォームのバインディングテンプレートを用いて、そのプラットフォームをサポートするために Thing Description で提供しなければならない通信メタデータを検索できる。

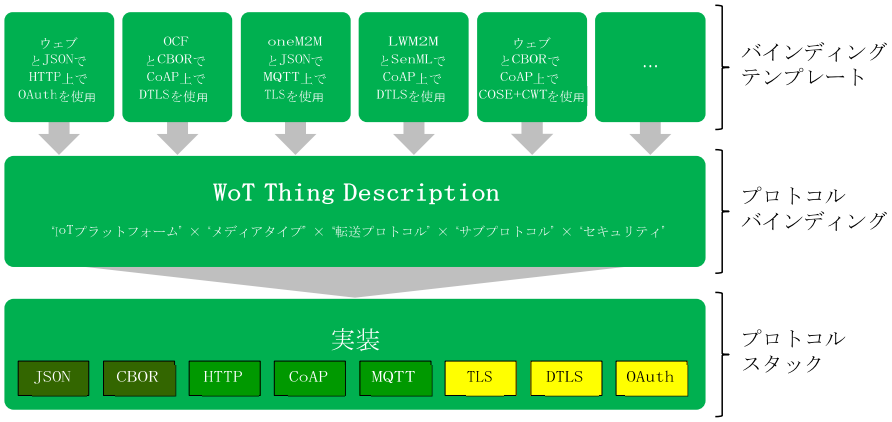

図 26 バインディングテンプレートからプロトコルバインディングへ

図 26 は、バインディングテンプレートの適用方法を示す。WoT バインディングテンプレートは、IoT プラットフォームごとに一度だけ作成され、そのプラットフォームのデバイスのすべての TD で再利用できる。TD を処理している Consumer は、対応するプロトコルスタックを組み込み、TD で提供される情報に従ってスタック（またはそのメッセージ）を設定することにより、必要なプロトコルバインディングを実装しなければならない。

プロトコルバインディングの通信メタデータは、次の五つの次元にまたがっている。

- IoT プラットフォーム:

  IoT プラットフォームは、プラットフォーム固有の HTTP ヘッダフィールドや CoAP オプションなど、アプリ

ケーション層で独自の変更を導入することがよくある。フォーム（§ 6.5.2 フォームを参照）に必要な情報を含め、使用されるアプリケーション層プロトコル用に定義した追加のフォームフィールドにこれらの微調整を適用することができる。

- メディアタイプ:

  IoT プラットフォームは、多くの場合、データの交換に用いられる表現形式（別名、シリアライゼーション）が多様である。メディアタイプ［RFC2046］は、これらのフォーマットを識別するが、メディアタイプのパラメータでそれ以上の指定が可能である。フォームには、HTTP の既知のコンテンツタイプフィールドなどの追加のフォームフィールドにメディアタイプとオプションのパラメータを含めることができ、これによって、メディアタイプとその潜在的なパラメータが組み合わされる（例えば、text/plain; charset=utf-8）。

- 転送プロトコル:

  Web of Things では、アプリケーション固有のオプションやサブプロトコルのメカニズムのない、基礎となる標準的なアプリケーション層プロトコルに転送プロトコルという用語を用いる。フォーム送信ターゲットの URI スキームには、例えば、HTTP、CoAPS、WebSocket（それぞれ http:、coaps:、ws:を介する）などの転送プロトコルを識別するために必要な情報が含まれている。

- サブプロトコル:

  転送プロトコルには、うまく相互作用を行うために知っていなければならない拡張メカニズムがある。このようなサブプロトコルは、URI スキームだけでは識別できず、明示的に宣言しなければならない。例は、ロングポーリング［RFC6202］やサーバー送信イベント［HTML］などの HTTP のプッシュ通知回避策である。フォームでは、サブプロトコルを識別するために必要な情報を、追加のフォームフィールドに含めることができる。

- セキュリティ:

  セキュリティメカニズムは、通信スタックの様々な層に適用でき、多くの場合、互いに補完するために一緒に使用できる。例は、(D)TLS［RFC8446］/［RFC6347］、IPSec［RFC4301］、OAuth［RFC6749］、および ACE［RFC7744］である。セキュリティの分野横断的な性質により、正しいメカニズムを適用するために必要な情報は、Thing の一般的なメタデータ内で提供するか、相互作用のアフォーダンスやフォームごとに特殊化することができる。

§7.3 WoT スクリプト API

この節は参考情報である。

WoT スクリプティング API は、ウェブブラウザ API に似た ECMAScript ベースの API［ECMAScript］を提供することにより、IoT アプリケーション開発を容易にする W3C WoT のオプションの「便利な」構成要素である。スクリプティングランタイムシステムを WoT ランタイムに統合することにより、WoT スクリプティング API は、Thing、Consumer、Intermediary の動作を定義するポータブルなアプリケーションのスクリプトの使用を可能にする。

従来、IoT デバイスのロジックはファームウェアに実装されており、その結果、比較的複雑な更新プロセスを含む、組み込み開発と同様の生産性の制約が生じる。対照的に、WoT スクリプティング API は、ウェブブラウザとあまり違いのない IoT アプリケーションのランタイムシステムで実行される再利用可能なスクリプトにより、デバイスのロジックの実装を可能にし、生産性の向上と統合コストの削減を目指している。さらに、標準的な API により、アプリケーションモジュールの移植が可能になる。例えば、計算集約型のロジックをデバイ

スからローカルゲートウェイに移動させたり、タイムクリティカルなロジックをクラウドからゲートウェイやエッジノードに移動させたりできる。

翻訳者のメモ

英語原本中で「compute-intense」となっているのは、「compute-intensive」の間違いと思われる。

非規定的な WoT スクリプティング API 仕様 [WOT-SCRIPTING-API] は、スクリプトが WoT Thing Description を発見、取得、利用、作成、公開できるようにするプログラミングインターフェースの構造とアルゴリズムを定義する。WoT スクリプティング API のランタイムシステムは、他の Thing とその相互作用のアフォーダンス

(Property、Action、Event) に対するインターフェースとして機能するローカルオブジェクトをインスタンス化する。これにより、スクリプトが Thing を公開すること、つまり相互作用のアフォーダンスを定義および実装し、Thing Description を公開することもできるようになる。

§ 7.4 WoT セキュリティとプライバシーに関するガイドライン

この節は参考情報である。

セキュリティは分野横断的な関心事であり、システム設計のすべての側面において考慮すべきである。WoT アーキテクチャでは、TD の Public Security Metadata のサポートなどの特定の明示的な機能、および WoT スクリプティング API の設計における懸念の分離によって、セキュリティがサポートされる。各構成要素の仕様には、その構成要素の特定のセキュリティとプライバシーに関する留意点の議論も含まれている。別の参考情報仕様で

ある WoT セキュリティとプライバシーに関するガイドライン [WOT-SECURITY] は、分野横断的なセキュリティとプライバシーに関するガイダンスを追加提供する。

§ 8. 抽象的な Servient のアーキテクチャ

この章は参考情報である。

§ 6.7 WoT システム構成要素とその相互接続性で定義しているように、Servient は、前の項で示した WoT 構成要素を実装するソフトウェアスタックである。Servient は、Thing を提供および公開したり、および (または) Thing を利用することができる (つまり、Consumer を提供することができる)。プロトコルバインディングに応じて、サーバーとクライアントの両方の役割を実行できることから、Servient という混成語が命名された。

前章では、WoT 構成要素が概念的にどのように相互に関連し、それらが抽象 WoT アーキテクチャにどのように対応するかについて説明している (§ 6. WoT アーキテクチャを参照)。これらの概念を実装する場合、特定の技術的側面を考慮した、より詳細な観点が必要となる。この賞では、Servient の実装に関するアーキテクチャの詳細について説明する。

図 27 は、(オプションの) WoT スクリプティング API という構成要素を用いた Servient の実装を示している。ここでは、WoT ランタイムは、WoT 固有の側面の管理に加えて、アプリケーションスクリプトの解釈と実行も行う、スクリプティングランタイムシステムである。WoT スクリプティング API をサポートする Servient は、通常、強力なデバイスやエッジノード、またはクラウド上で実行される。WoT アーキテクチャは、WoT ランタイムのアプリケーション向け API を JavaScript/ECMAScript に制限しない。他のランタイムシステムを用いて

Servient を実装することもできる。

§ 8.8.1 ネイティブな WoT API の項では、WoT スクリプティング API という構成要素を用いない、代替の Servient 実装を示している。WoT ランタイムは、アプリケーション向け API に任意のプログラミング言語を使用できる。通常、それは、Servient のソフトウェアスタックのネイティブ言語である。例えば、組み込み Servient

ではC/C++、クラウドベースのServientではJavaが挙げられる。それは、アプリケーションスクリプトの利点を、低い資源利用と組み合わせるLuaなどの代替スクリプト言語であってもよい。

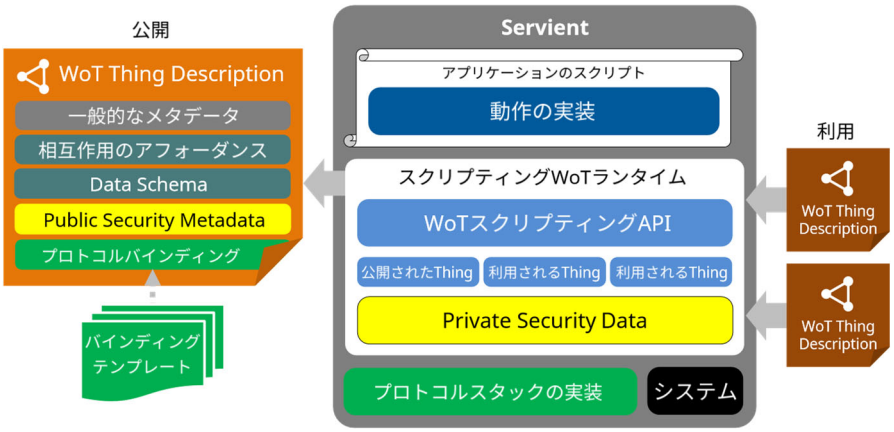

図27 WoTスクリプティングAPIを用いたServientの実装

図27で示した各モジュールの役割と機能については、以下の節で説明する。

§8.1 動作の実装

動作は、Thingの全体的なアプリケーションロジックを定義し、いくつかの側面を持っている。

動作には、Thingの自律的な動作（例えば、センサーのサンプリングまたはアクチュエータの制御ループ）、相互作用のアフォーダンスのハンドラー（つまり、アフォーダンスが作動したときに実行される具体的なアクション）、Consumerの動作（例えば、Thingの制御またはマッシュアップの実現）、およびIntermediaryの動作（例えば、単にThingを代理する、または仮想エンティティーを編成する）が含まれる。Servient内の動作の実装は、どのようなThing、Consumer、Intermediaryがこの構成要素で提供されるかを定義する。

図27は、オプションのWoTスクリプティングAPIという構成要素を実装しているServientを示しており、JavaScript［ECMAScript］で記述されている移植可能なアプリケーションスクリプトが動作を定義する。アプリケーションスクリプトはWoTランタイムの一部であるスクリプティングランタイムシステムによって実行される（WoTスクリプティングAPIまたはその他のスクリプトベースのAPIを提供する場合）。アプリケーションスクリプトは、共通のWoTスクリプティングAPIの定義に対して記述されているため、移植可能であり、そのため、 WoTスクリプティングAPIという構成要素を備えた任意のServientによって実行できる。これにより、例えば、ネットワーク要件を満たすためにConsumerをクラウドからエッジノードに移動させたり、増大する資源の需要を満たすためにIntermediaryをクラウドに移動させたりするなど、システム構成要素間でアプリケーションロジックを移動することができる。移植可能なアプリケーションにより、Servientのデプロイ後に追加の動作を

「インストール」できる。

原則として、相互作用のアフォーダンスがWoTインターフェースを介して外部に示されている限り、Thingの動作を定義するために任意のプログラミング言語とAPIを使用できる。アプリケーション向けAPIとプロトコルスタックの間の適応は、WoTランタイムにより処理される。WoTスクリプティングAPIという構成要素を用いない動作の実装については、§ 8.8.1 ネイティブなWoT APIを参照のこと。

§8.2 WoT ランタイム

技術的には、Thing の抽象化とその相互作用モデルはランタイムシステムに実装される。この WoT ランタイムは、動作の実装に関する実行環境を整備し、Thing を公開および（または）利用できるため、WoT Thing

Description を取得し、処理し、シリアライズし、提供することができなければならない。

すべての WoT ランタイムには、動作実装用のアプリケーション向けインターフェース（つまり、API）がある。

図 27 に示される、オプション機能である WoT スクリプティング API という構成要素は、Thing の抽象化に従ったアプリケーション向けインターフェースを定義し、実行中にアプリケーションのスクリプトを介して動作の実装

がデプロイできるようにする。これも、コンパイル中にのみ使用できます。コンパイル時にのみ使用できる、

WoT スクリプティング API 以外の API に関しては、§ 8.8.1 ネイティブな WoT API を参照のこと。一般的に、アプリケーションのロジックは、WoT ランタイムの管理面、特に Private Security Data に対する不正アクセスを防止するために、分離された実行環境で実行されるべきである。マルチテナント方式の Servient では、異なる利

用者に対して実行環境の分離を追加する必要がある。

WoT ランタイムは、Thing のライフサイクル、より正確にはそのソフトウェアの抽象化と記述を管理するための特定の操作を提供する必要がある。ライフサイクル管理（LCM）システムは、これらのライフサイクル操作を Servient 内にカプセル化し、内部インターフェースを用いてライフサイクル管理を実現してもよい。このような操作の詳細は、実装によって異なる。WoT スクリプティング API には LCM 機能が含まれているため、このようなシステムの一つの可能な実装となる。

WoT ランタイムは、動作の実装をプロトコルバインディングの詳細から切り離すため、Servient のプロトコルスタック実装とインタフェースしなければならない。また、WoT ランタイムは通常、例えば、接続されているセンサーやアクチュエーターなどのローカルなハードウェアにアクセスしたり、ストレージなどのシステムサービスにアクセスしたりするために、根底にあるシステムともインタフェースする。これらのインターフェースは両方とも実装に固有のものだが、WoT ランタイムは実装された Thing の抽象化に必要な適応を提供しなければならない。

§8.3 WoT スクリプティング API

WoT スクリプティング API の構成要素は、WoT Thing の記述仕様［WOT-THING-DESCRIPTION］に厳密に従った ECMAScript API を定義する。これは、動作の実装とスクリプトを使った WoT ランタイムとの間のインターフェースを定義する。さらに、例えば、ウェブブラウザ API 向けの jQuery と同様に、スクリプティング API にもとづいて、よりシンプルな API を実装してもよい。

詳細に関しては、［WOT-SCRIPTING-API］を参照すること。

§8.4 公開された Thing と利用される Thing の抽象化

WoT ランタイムは、TD に基づいて Thing のソフトウェア表現をインスタンス化する。このソフトウェア表現は、動作の実装のためのインターフェースを提供する。

公開された Thing の抽象化は、ローカルで提供され、Servient のプロトコルスタックの実装を介して外部からアクセス可能な Thing を表す。動作の実装は、メタデータと相互作用のアフォーダンスの定義と、自律的な動作の提供により、公開された Thing を完全に制御できる。

利用される Thing の抽象化は、通信プロトコルを用いてアクセスする必要のある、リモートで提供される

Consumer用の Thing を表す。利用される Thing は、プロキシオブジェクトかスタブである。動作の実装は、メタデータの読み取りと、対応する TD に記述されているとおりの相互作用のアフォーダンスの作動に限定される。利用される Thing は、独自または旧式の通信プロトコルの背後にあるローカルなハードウェアやデバイスなどのシステムの機能を表すこともできる。この場合、WoT ランタイムは、システム API と利用される Thing との間に必要な適合処理を行わなければならない。さらに、対応する TD を提供し、例えば、アプリケーション向け API を介して WoT ランタイムにより提供される（WoT スクリプティング API [WOT-SCRIPTING-API] で定義されている discover () メソッドなどの）何らかの発見メカニズムを拡張することにより、その TD を動作の実装において利用できるようにしなければならない。

WoT スクリプティング API を用いる場合、公開された Thing と利用される Thing は、JavaScript のオブジェクトであり、アプリケーションのスクリプトによって作成、操作、破棄できる。しかし、アクセスは、例えば、マルチテナント方式の Servient など、セキュリティのメカニズムにより制限される場合がある。

§8.5 Private Security Data

Thing と相互作用するための秘密鍵などの Private Security Data も WoT ランタイムによって概念的に管理されるが、意図的にアプリケーションが直接アクセスできないようにされている。実際に、最も安全なハードウェアの実装では、このような Private Security Data は個別の分離されたメモリ（例えば、安全な処理要素または TPM）に格納され、操作の抽象的な集合のみ（分離されたプロセッサとソフトウェアスタックによって実装される可能性さえある）が提供され、攻撃対象領域を制限してこのデータの外部漏洩を防止する。Private Security Data は、承認を行い、相互作用の完全性と機密性を保護するために、プロトコルバインディングによって透過的に用いられる。

> 翻訳者のメモ
>
> 文頭の「Private security data」も、「3. 用語」で定義された「Private Security Data」として扱うべきと考えられる。

§8.6 プロトコルスタックの実装

Servient のプロトコルスタックには公開された Thing の WoT インターフェースが実装され、それは Consumer が（利用される Thing を介して）リモートの Thing の WoT インターフェースにアクセスするために用いられる。

これにより、ネットワークを介して相互作用するための具体的なプロトコルのメッセージが作成される。

Servient には複数のプロトコルを実装できるため、複数のプロトコルバインディングをサポートすれば、様々な IoT プラットフォームとの相互作用が可能となる。

多くの場合、標準プロトコルを用いている場合には、汎用プロトコルスタックを用いて、プラットフォーム固有のメッセージ（例えば、HTTP（S）固有のもの、CoAP（S）固有のもの、MQTT ソリューションのものなど）を作成できる。このケースでは、Thing Description の通信メタデータを利用して、適切なスタック（例えば、正当なヘッダフィールドを備えた HTTP や、正当なオプションを備えた CoAP）を選択し設定する。メディアタイプ [RFC2046] によって識別される、期待されるペイロード表現形式（JSON、CBOR、XML など）のパーサーとシリアライザーも、これらの汎用プロトコルスタックで共有できる。詳細に関しては、[WOT-BINDING-TEMPLATES] を参照のこと。

§8.7 システム API

WoT ランタイムの実装は、Thing の抽象化により、通信プロトコルを介してアクセスできているかのように、ローカルなハードウェアやシステムサービスを動作の実装に提供できる。このケースでは、WoT ランタイムは、動作の実装が、プロトコルスタックの代わりにシステムと内部的に連動する利用される Thing をインスタンス化できるようにすべきでである。これは、アプリケーション向けの WoT ランタイム API によって提供される発見メカニズムの結果に、そのようなシステムの Thing（ローカルの WoT ランタイムでのみ入手可能）を列挙することで行える。

デバイスは、物理的に Servient の外部に置くこともできるが、独自のプロトコルや WoT インターフェースとしての条件を満たしていないプロトコルを介して接続することもできる（§ 6.6 プロトコルバインディングを参照)。このケースでは、WoT ランタイムは、独自の API を介して、そのようなプロトコル（例えば、ECHONET

Lite、BACnet、X10、I2C、SPI など）で旧式デバイスにアクセスできるが、Thing の抽象化によって、それを動作の実装に公開することも選択できる。その後で、Servient は、旧式デバイスへのゲートウェイとして機能する
ことができる。これは、WoT Thing Description を用いて旧式デバイスを直接記述できない場合にのみ行うべきである。

動作の実装は、独自の API やその他の手段を介してローカルなハードウェアやシステムサービス（例えば、ストレージ）にアクセスすることもできる。しかし、これは移植性を妨げるため、W3C WoT 標準化の範囲外である。

§8.8 代替の Servient と WoT 実装

WoT スクリプティング API の構成要素はオプションである。代替の Servient を実装することができ、その場合、 WoT ランタイムは、アプリケーションロジックのための代替 API を提供するが、これは、任意のプログラミング
言語で記述できる。

さらに、W3C WoT を意識していないデバイスやサービスであっても、それらに整形式の WoT Thing Description を提供できる場合は、利用できる。このケースでは、TD は、ブラックボックス的な実装を持つ Thing の WoT インターフェースを記述する。

§8.8.1 ネイティブな WoT API

開発者が WoT スクリプティング API を用いずに、Servient の実装を選択しうる理由は様々である。メモリやコンピュータ資源が不足しているために、開発者が必要なソフトウェアスタックまたはフル機能のスクリプトエンジンを使用できないことが原因である場合がある。または、開発者が独自のユースケース（例えば、独自の通信プロトコル）をサポートするために、特定のプログラミング環境や言語でしか利用できない特定の機能やライブラリを用いなければならない場合もありえる。

このケースでは、WoT スクリプティング API の代わりに別のアプリケーション向けインターフェースを用いて同等の抽象化と機能を公開しつつ、依然として WoT ランタイムを使用できる。後者の場合を除き、§ 8. 抽象的な

Servient のアーキテクチャのすべての構成要素の説明は図 28 でも有効である。

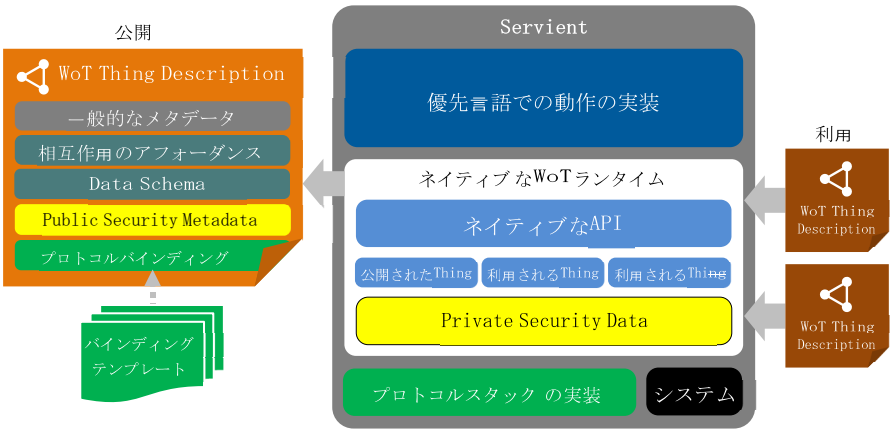

図28 ネイティブなWoT APIを用いたServientの実装

§8.8.2 既存デバイスの Thing Description

既存の IoT デバイスやサービスを W3C Web of Things に統合し、これらのデバイスやサービスに関する Thing Description を作成することにより、それらを Thing として用いることもできる。このような TD は、手動でもツールやサービスを用いても作成できる。例えば、TD は、別のエコシステムに依存している機械可読形式で提供されるメタデータの自動翻訳を提供するサービスにより生成できる。しかし、これは、対象とするデバイスがプロトコルバインディングで記述できるプロトコルを用いている場合にのみ行えることである。これに関する要件は§ 6.6 プロトコルバインディングで示される。これまでの議論の多くは、Thing が自身の Thing Description を提供することも暗示している。これは便利なパターンだが、必須ではない。特に、既存のデバイスを変更して自身の Thing Description を直接提供することはできない場合がある。このケースでは、ディレクトリやその他の外部の別の配信メカニズムなどのサービスを用いて Thing Description を別途提供する必要がある。

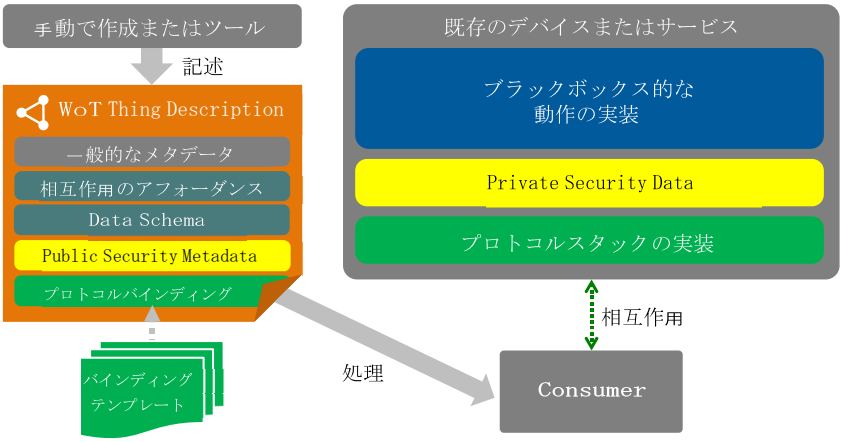

図29 既存のIoTデバイスのW3C WoTへの統合

§9. WoT のデプロイメント例

この章は参考情報である。

この章では、Thing と Consumer の役割を実装しているデバイスとサービスを様々な具体的なトポロジーとデプロイメントシナリオで相互接続する際に、Web of Things (WoT) の抽象アーキテクチャをインスタンス化する様々な方法の例を示す。これらのトポロジーとシナリオは規定的ではないが、WoT 抽象アーキテクチャによって認められ、サポートされている。

特定のトポロジーについて論じる前に、まず Thing と Consumer が WoT ネットワークで担うことができる役割と、それらが持っている公開された Thing と利用される Thing のソフトウェア抽象化との関係について振り返

る。公開された Thing と利用される Thing は、それぞれ Thing と Consumer の役割をする Servient の動作の実装に内部的に利用できる。

§9.1 Thing と Consumer の役割

Thing の役割を持つ Servient は、Thing Description（TD）に基づいて公開された Thing を作り出す。TD は公開され、Consumer または Intermediary の役割を持つ他の Servient から利用できるようになります。TD は様々な

方法で公開することができる。TD は、Thing Directory サービスなどの管理システムに登録されている場合もあるし、Thing が TD へのリクエストを受信するとリクエスト元に TD を提供するという場合もある。アプリケーションのシナリオによっては、TD を Thing に静的に関連付けることすら可能である。

Consumer の役割を持つ Servient は、なにがしかの発見メカニズムを用いて Thing の TD を取得し、その取得した TD に基づいて利用される Thing を作成する。具体的な発見メカニズムは、個々のデプロイメントシナリオに依存している。それは、Thing Directory などの管理システムや、発見プロトコル、あるいは静的な割り当て等により提供される可能性がある。

しかし、特定可能な人物に関連付けられているデバイスが記述されている TD は、プライバシー情報の推測に用いられる可能性があることに注意する必要がある。したがって、そのような TD の配信に関する制約を具体的な TD の発見メカニズムに組み込まなければならない。可能であれば、特定のユースケースで絶対に必要な場合を除いて、ID や人間が読み取れる情報をフィルターで除外するなど、TD により公開されている情報を制限する措置も講じなければならない。プライバシーに関する課題は、§ 10. セキュリティとプライバシーへの配慮でおおまかに論じられており、より詳細な議論が［WOT-THING-DESCRIPTION］仕様で提供されている。

接続されたセンサーとアクチュエーターの相互作用などの、デバイスの内部システム機能もまた、オプションとして、利用される Thing の抽象化として表現することができる。

利用される Thing でサポートされる機能は、プログラミング言語のインターフェースを介して Consumer の動作の実装に提供される。WoT スクリプティング API では、利用される Thing はオブジェクトにより表現される。

Thing で実行されている動作の実装（つまり、アプリケーションのロジック）は、公開された Thing が提供するプログラミング言語インターフェースを用いることにより、相互作用のアフォーダンスを介して Consumer と連動することができる。

Thing は必ずしも物理デバイスを表現するとは限らない。Thing は、複数デバイスの集合体、またはゲートウェイやクラウドで実行される仮想サービスを表現する可能性もある。同様に、Consumer は、ゲートウェイやクラウドで実行されているアプリケーションやサービスを表現する可能性がある。また、Consumer は、エッジデバイスに実装することもできる。Intermediary において、一つの Servient が、一つの WoT ランタイムを共有しながら、Thing と Consumer の両方の役割を同時に果たしている。

§9.2 WoT システムのトポロジーとデプロイメントシナリオ

この節では、WoT システムの様々なトポロジーとデプロイメントシナリオについて論じる。これらはパターンの例にすぎず、他の相互接続トポロジーも可能である。ここで説明するトポロジーは、Web of Things のユースケース（§ 4. ユースケース）と、そこから抽出された技術要件（§ 5. 要件）に基づくものである。

§9.2.1 同じネットワーク上の Consumer と Thing

図 30 で示している最もシンプルな相互接続のトポロジーでは、Consumer と Thing は同じネットワーク上にあり、仲介なしに互いに直接通信を行える。このトポロジーになるユースケースの一つは、Consumer がゲートウェイで実行されているオーケストレーションサービス（orchestration service）やその他の IoT アプリケーションで、Thing がセンサーやアクチュエータに接続しているデバイスである場合である。しかし、クライアント/ サーバーの関係は簡単に逆転可能で、クライアントは、ゲートウェイやクラウド上で Thing として実行されているサービスにアクセスする Consumer の役割を持つデバイスになりえる。

図 30 同じネットワーク上の Consumer と Thing

Thing がクラウド内にあり、Consumer がローカルネットワーク上にある場合（スマートホームのユースケースの例については図 1 を参照）は、実際のネットワークトポロジーはより複雑になる。例えば、NAT トラバーサルが必要で、ある種の発見は認められていない場合がある。このようなケースでは、後に論じるより複雑なトポロジーの一つの方が適切である。

§9.2.2 Intermediary を介して接続された Consumer と Thing

Intermediary は、ネットワーク上で Thing と Consumer の両方の役割を担い、WoT ランタイム内で公開された Thing と利用される Thing の両方のソフトウェア抽象化をサポートする。Intermediary は、デバイスとネットワークとの間のプロキシやデジタルツインに使用できる。

§9.2.2.1 プロキシとして機能する Intermediary

Intermediary のシンプルな応用の一つは、Thing に対するプロキシである。Intermediary がプロキシとして機能する場合、二つの別々のネットワークまたはプロトコルとのインターフェースを持っている。これには、TLS エンドポイントの提供など、付加的なセキュリティのメカニズムの実装が伴う場合がある。一般的に、プロキシによって相互作用が変わることはないため、Intermediary によって公開される TD の相互作用は、利用される

TD の相互作用と同じであるが、接続メタデータは変更されている。

このプロキシのパターンを実装するために、Intermediary は、Thing の TD を取得して利用される Thing を作成する。これにより、同じ相互作用のアフォーダンスを持つソフトウェアの実装として Thing のプロキシオブジェクトが作成される。次に、新しい識別子と、場合によっては新しい通信メタデータ（プロトコルバインディング）および（または）新しい Public Security Metadata を備えたプロキシオブジェクトの TD が作成される。最後に、

この TD に基づいて公開された Thing が作成され、Intermediary は、適切な公開メカニズムを介して、他の Consumer や Intermediary に TD を通知する。

図 31 プロキシとして機能する Intermediary を介した Consumer と Thing の接続

§9.2.2.2 デジタルツインとして機能する Intermediary

より複雑な Intermediary は、デジタルツインとして知られている。デジタルツインは、プロトコルの変更や、ネットワーク間の通訳を行う場合も行わない場合もあるが、状態のキャッシング、更新の先延ばし、対象デバイスの動作の予測シミュレーションなどの追加サービスを提供する。例えば、IoT デバイスの電力が限られていれば、あまり頻繁には起動せず、デジタルツインと同期した直後にスリープ状態に戻ることを選択できる。このケースでは通常、デジタルツインは電力の制約が少ないデバイス上（クラウドやゲートウェイなど）で実行され、制約のあるデバイスの代わりに相互作用に応答することができる。現在のプロパティーの状態に対するリクエストを、デジタルツインがキャッシュされた状態を用いて満たす場合もある。対象としている IoT デバイスがスリープ状態にあるときに到着したリクエストはキューに入れられ、起動時に送信される。このパターンを実装するためには、Intermediary、つまりデジタルツインは、いつデバイスが起動しているかを知る必要がある。

Thing としてのデバイスの実装には、そのための通知メカニズムを含める必要がある。これは、別の Consumer/Thing のペアを用いて、またはこの目的のための Event の相互作用を用いて実装できる。

§9.2.3 クラウドサービスから制御されるローカルネットワーク内のデバイス

スマートホームのユースケースでは、ホームネットワークに接続されているデバイス（センサーと家電）は監視されることが多く、場合によってはクラウドサービスによる制御も行われている。通常、デバイスが接続されているホームネットワークとクラウドの間には NAT デバイスが存在している。NAT デバイスは、IP アドレスを変換するだけでなく、接続を選択的にブロックするファイアウォールのサービスを提供していることが多い。ローカルデバイスとクラウドサービスは、通信がゲートウェイをうまく通過できた場合にのみ相互に通信を行える。

ITU-T 勧告 Y.4409/Y.2070［Y.4409-Y.2070］で採用されている典型的な構造を図 32 で示している。この構造には、ローカルとリモートの両方の Intermediary がある。ローカルの Intermediary は、複数の Thing の相互作用のアフォーダンスを一つの公開された Thing（の集合）へと集約させ、そのすべてを（すべての相互作用が、共通のベースとなるサーバが持つ一つの URL 名前空間にマッピングされていて、一つのポートを用いている HTTP などの）共通のプロトコルにマッピングすることができる。これにより、ローカルの Intermediary が NAT デバイスをトラバースできる集約型プロトコルを用いていて、そのサービスをインターネットで公開する（STUN、

TURN、DyDNS などの）何らかの方法を持っていると仮定すると、NAT デバイスの背後にあるすべての Thing にアクセスするためのシンプルな方法がリモートの Intermediary に提供される。さらに、ローカルの

Intermediary は、Thing のプロキシの機能を果たすことができるため、接続された Thing で、それぞれ（HTTP、MQTT、CoAP などの）異なるプロトコルや様々なエコシステムの規定が用いられていたとしても、

Thing で用いられている様々なプロトコルを Consumer が意識しなくてもよいように、公開された Thing はそれらを一つのプロトコルに集約することができる。

図 32 にはリモートの Intermediary に接続された二つのクライアントがあり、これは、NAT の境界の背後にあるサービスを集約しており、付加的なプロトコル変換やセキュリティのサービスを提供できる。特に、ローカルの

Intermediary は容量が限られたネットワーク上にある可能性があり、そのサービスをすべてのユーザが直接利用できるようにすることは現実的ではない。このケースでは、ローカルの Intermediary へのアクセスはリモートの Intermediary にのみ提供される。次に、リモートの Intermediary は、より一般的なアクセス制御メカニズムを実装し、キャッシングやスロットリングを実行して、過剰なトラフィックから消費者を保護することもできる。また、これらの消費者は、（HTTPS などの）オープンなインターネットに適した一つのプロトコルを用いて

Intermediary と通信すると考えられ、それによってクライアントの開発はよりシンプルになる。

このトポロジーでは、Consumer と Thing の間に NAT とファイアウォールの機能があるが、ローカルとリモートの Intermediary が連携してファイアウォールをトンネリングさせ、すべての通信を通すため、Consumer と Thing はファイアウォールについて何も知っている必要がない。ペアになった Intermediary は、アクセス制御とトラフィック管理を提供することにより、ホームデバイスの保護も行う。

翻訳者のメモ

英語原文中で「between the consumers and things」は、本来、大文字の「Consumers and Things」とすべきものと考えられる。

また、「so the consumers and things need to know」においても、「so the Consumers and Things need to know」とすべきものと考えられる。

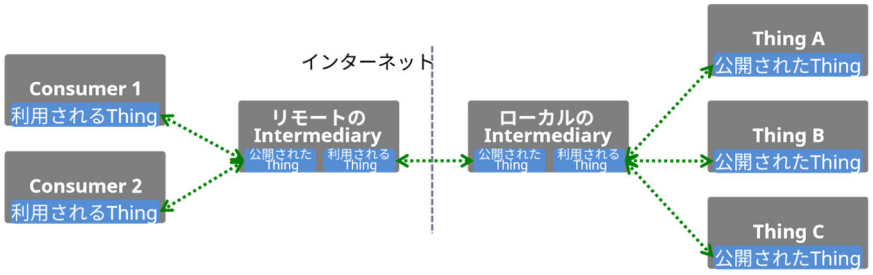

図 32 ペアになった Intermediary を介して Thing として実装されたローカルデバイスに接続された Consumer として実装されたクラウドアプリケーション

より困難なケースでは、NAT とファイアウォールトラバーサルが示されているとおりに機能しない場合がある。特に、ISP がパブリックにアクセス可能なアドレスをサポートしていなかったり、STUN/TURN および（または）DyDNS がサポートされていなかったり利用できない場合がある。このケースでは、Intermediary は、クライアント/サーバーの役割を逆にして初期接続を設定し（ローカルの Intermediary が最初にクラウド上のリモートの Intermediary に接続して）、次に、ペアとなった Intermediary が（例えば、TLS を用いて接続を保護した安全な WebSocket を用いて）トンネルを確立する。このトンネルは、特注のプロトコルを用いて Intermediary の間のすべての通信をエンコードするために使うこともできる。このケースでは、通常のブラウザ/ウェブサーバーの相互作用と同様に、ローカルの Inermediary からリモートの Intermediary へと、標準ポートを用いて

HTTPS で初期接続を行うことができる。これにより、ほとんどのホームファイアウォールを通過でき、接続が外向きであるため、ネットワークアドレス変換による問題が発生しない。しかし、特注のトンネリングプロトコルが必要な場合でも、リモートの Intermediary はこの特注のプロトコルを標準の外部プロトコルに変換することができる。接続された Consumer と Thing は、この変換について知っている必要はない。この例を、Thing と

Consumer の両方が NAT の境界の両側で接続できるユースケースに拡張することもできる。しかし、そのためには、二つの Intermediary の間に双方向のトンネルを確立する必要もある。

§9.2.4 Thing Directory を用いた発見

クラウド上のサービスによってローカルデバイス（および場合によってはサービス）を監視または制御できるようになると、その上で様々な付加的なサービスを構築できる。例えば、クラウドアプリケーションは、収集されたデータの分析に基づいてデバイスの動作条件を変更できる。

しかし、リモートの Intermediary がクライアントアプリケーションにサービスを提供しているクラウドプラットフォームの一部である場合、クライアントは、例えば、接続されているデバイスのディレクトリにアクセスすることにより、デバイス情報を見つけることができる必要がある。下の図では、シンプルにするために、す

べてのローカルデバイスが Thing として実装され、すべてのクラウドアプリケーションが Consumer として実装されていると想定している。Thing として実装されているローカルデバイスのメタデータをクラウドアプリケーションで利用できるようにするために、そのメタデータを Thing Directory サービスに登録することができる。このメタデータは、具体的には、リモートの Intermediary によって提供される Public Security Metadata と通信メタデータ（プロトコルバインディング）を反映するように変更されたローカルデバイスの TD である。その後で、 Thing Directory に照会することにより、クライアントアプリケーションは、ローカルデバイスと通信するために必要なメタデータを取得し、その機能を実現できるようになる。

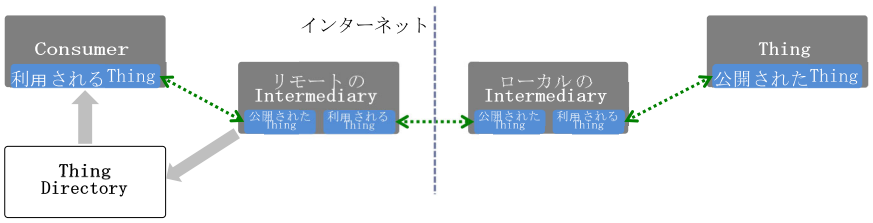

図 33 Thing Directory を用いたクラウドサービス

上図では示していない、より複雑な状況では、Thing として機能するクラウドサービスもありえる。この場合も、Thing Directory に自身を登録することができる。Thing Directory はウェブサービスであるため、NAT やファイアウォールデバイスを介してローカルデバイスから見ることができるべきであり、そのインターフェース

を自身の TD により提供することさえ可能である。そして、Consumer として動作するローカルデバイスは、Thing Directory を介してクラウド内の Thing を発見し、直接、Thing に接続することができる。あるいは、例えば、プロトコル変換が必要な場合、ローカルにある Intermediary を介して、Thing に接続することができる。

§9.2.5 複数の領域にまたがるサービス間の接続

それぞれが異なる IoT プラットフォームに基づいている複数のクラウドのエコシステムは、連携してより大きな「システムのシステム」というエコシステムを構築できる。下の図は、前述のクラウドアプリケーションのエコシステムの構造に基づいて、「システムのシステム」を作成するために相互接続された二つのエコシステムを示している。あるエコシステムのクライアント（つまり、下記の Consumer A）が別のエコシステムのサーバー（つまり、下記の Thing B）を利用する必要がある場合を考えてみよう。このエコシステムを横断するアプリケーションとデバイスの統合を実現するメカニズムは複数存在する。以下では、これを実現する方法を示すために、二つの方法について、それぞれ図を用いて説明する。

§9.2.5.1 Thing Directory の同期を介した接続

図 34 では、二つの Thing Directory が情報を同期させており、それによって、Consumer A は Thing Directory A を介して Thing B の情報を取得できる。既に説明したように、リモートの Intermediary B は Thing B の影（shadow）の実装を維持する。この影のデバイスの TD を取得することにより、Consumer A は、リモートの Intermediary B を介して Thing B を使用できる。

> 翻訳者のメモ
>
> 上記説明で用いられている「影（shadow)」については、「デジタルツイン」とも呼ばれる、デバイスの仮想表現であることを明示的に説明するべき.

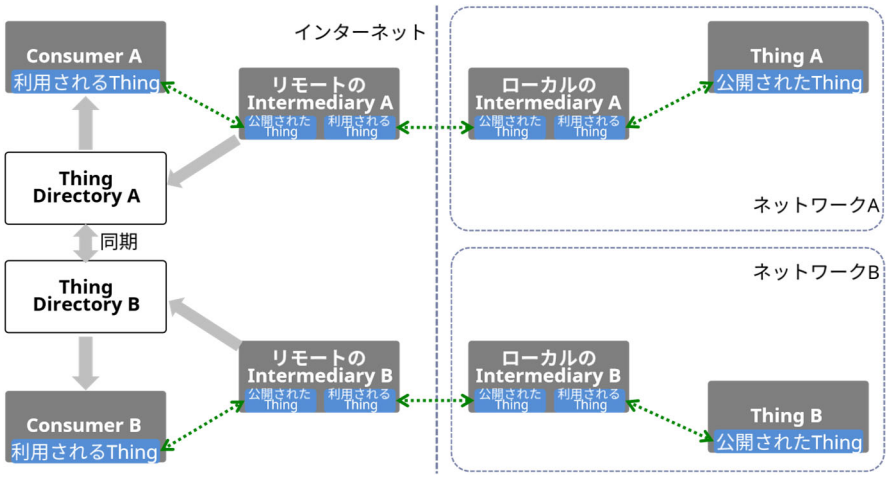

図 34 Thing Directory の同期を介した複数クラウドの接続

§9.2.5.2 プロキシの同期を介した接続

図 35 では、二つのリモートの Intermediary がデバイス情報を同期させている。Thing B の影がリモートの Intermediary B で作成されると同時に、影の TD がリモートの Intermediary A に同期される。次に、リモートの Intermediary A は、自分自身の中に Thing B の影を作成し、TD を Thing Directory A に登録する。この方法であれば、Thing Directory 間の同期は必要ない。

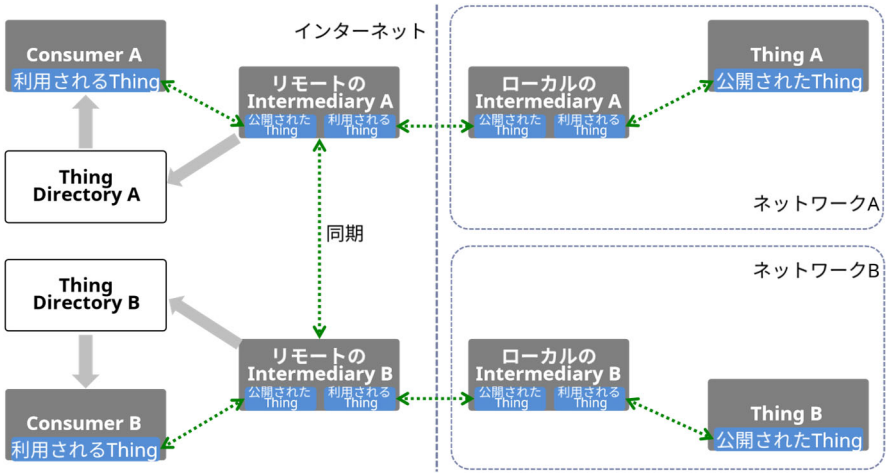

図 35 Intermediary の同期を介した複数クラウドの接続

§10. セキュリティとプライバシーへの配慮

この章は参考情報である。

セキュリティとプライバシーは分野横断的な課題であり、すべての WoT 構成要素と WoT 実装において考慮する必要がある。この章では、具体的な WoT 実装のセキュリティとプライバシーの保護に役立つように、一般的な課題とガイドラインをまとめる。しかし、これらは一般的なガイドラインに過ぎず、この文書で記述しているような抽象アーキテクチャ自体がセキュリティとプライバシーを保証することはできない。その代わりに、具体的な実装の詳細について熟考する必要がある。セキュリティとプライバシーに関する課題の、より詳細で完全な分析は、WoT セキュリティとプライバシーに関するガイドライン仕様 [WOT-SECURITY] を参照のこと。

全体として、WoT の目標は、セキュリティを含めて、IoT デバイスと IoT サービスの既存のアクセス方法とプロパティーを記述することである。一般的に、W3C WoT は、何を実装するのかを規定することではなく、何が存在するのかを記述することを目指している。既存システムの記述は、仮に理想的なセキュリティの動作を備えていなかったとしても、そのシステムを正確に描写すべきである。システムのセキュリティの脆弱性を明確に

理解することは、セキュリティ脅威の軽減に役立つ – しかし、もちろん、そのようなデータを、悪用する可能性のある人に配信する必要はない。

しかし、特に新しいシステムの場合は、WoT アーキテクチャにより、セキュリティとプライバシーのベストプラクティスを利用できるようになるべきである。一般的に、WoT のセキュリティアーキテクチャは、接続する IoT プロトコルとシステムの、目的とメカニズムをサポートしなければならない。このようなシステムでは、セキュリティ要件とリスクの許容度が多岐に渡るため、これらの要因に基づいてセキュリティメカニズムも多岐に渡る。

IoT デバイスは自律的に動作する必要があり、多くの場合、個人データへのアクセスを有しており、加えて（または）安全を重視するシステムの制御下にありえるため、セキュリティとプライバシーは、IoT の領域において特に重要である。IoT デバイスは、IT システムとは異なるリスク、場合によってはより高いリスクにさらされやすい。IoT システムを保護し、他のコンピューターシステムへの攻撃を仕掛けるために用いられることのないようにすることも重要である。

一般的に、セキュリティとプライバシーを保証することはできない。安全でないシステムを WoT によって安全なシステムに変えることはできないが、WoT アーキテクチャが害を及ぼさないようにする必要があり、少なくとも、WoT アーキテクチャが記述しサポートする対象システムと同程度には、セキュリティとプライバシーをサポートすべきである。

§10.1 WoT Thing Description に関するリスク

WoT Thing Description（TD）に含まれているメタデータには、潜在的に機密性がある。ベストプラクティスとして、TD は完全性保護メカニズムおよびアクセス制御ポリシーとともに用いるべきであり、権限を持つユーザにのみ提供を行うべきである。

これらの点に関する詳細や議論については、WoT Thing Description 仕様のセキュリティとプライバシーに関する考察の項を参照すること。

§10.1.1 Thing Description の Private Security Data に関するリスク

TD は、Public Security Metadata のみの利用を目指している。TD の作成者は、TD に Private Security Data が TD に含まれないようにしなければならない。Public Security Metadata と Private Security Data は厳密に分離されるべきである。TD には Public Security Metadata のみが含まれているべきであり、Consumer が権限を持つ場合にのみ、システムとしてアクセスするために何をする必要があるのかを知らせる。そして、権限付与は別途管理されている非公開情報に基づくべきである。

TD の仕様で定義されている TD セキュリティスキームは、Private Security Data のエンコーディングを意図的にサポートしていない。しかし、人間が読める記述などのその他のフィールドが誤用されてこの情報のエンコードが行われたり、そのような情報をエンコードする拡張メカニズムにより新しいセキュリティスキームが定義され、展開されるリスクがある。

軽減策:

TD と TD で用いられる拡張機能の作成者は、TD には Public Security Metadata のみが保存されるようにしなければならない。

§10.1.2 Thing Description の個人識別可能情報に関するリスク

Thing Description には、様々な種類の個人識別可能情報が含まれている可能性がある。明示的ではない場合でも、TD と、それにより特定可能な人物とを関連付けると、その人物に関する情報を推測できるようになる。例えば、携帯デバイスによって公開された、フィンガープリンティング可能な TD との関連付けによる位置情報の特定は、追跡のリスクになりえる。特定のデバイスのインスタンスを識別できない場合でも、TD で表現され

るデバイスの種類が人物に関連付けられていれば、個人情報となる可能性がある。例えば、ユーザに病状があると推測するために、医療機器が利用されるかもしれない。

一般的に、TD 内の個人識別可能情報はできる限り制限すべきである。しかし、回避できない場合もありえる。TD 中に直接的な PII と推論可能な PII の両方が存在している可能性があるということは、TD を別形式の PII のように扱うべきであることを意味する。それは、安全な方法で保存され送信されるべきであり、承認されたユーザにのみ提供されるべきであり、限られた回数だけキャッシュされるべきであり、リクエストに応じて削除されるべきであり、ユーザの同意を得て提供された目的にのみ用いられるべきであり、そうでない場合は、PII の使用に関するすべての要件（法的要件を含む）を満たすべきである。

軽減策:

TD での PII の保存はできる限り最小限に抑えるべきである。TD に明示的な PII がなくても、追跡のリスクや特定のプライバシーリスクが存在する場合がある。このリスクを最小限に抑えるために、一般的に PII が含まれているかのように TD を扱い、他の PII と同じ管理方針に従うべきである。それは、承認された利用者にのみ提供すべきである。特定のユースケースに不要な情報は、できる限り TD で公開すべきではない。例えば、TD 内の情報を識別する明示的な型とインスタンスも、ユースケースで必要でない場合には含めるべきではない。ユースケースで必要である場合でも、追跡のリスクを最小限に抑えるために、できる限り、グローバルに一意な識別子ではなく、分散型の限定的な範囲の識別子を用いるべきである。一部のユースケースでは、フィンガープリンティングのリスクを減らすために、人間が読める記述などのその他の形式の情報も削除可能である。

§10.1.3 Thing Description のコミュニケーションメタデータに関するリスク

WoT バインディングテンプレートは、そのプラットフォームが WoT での使用の条件を満たしていると見なされるように、基盤となる IoT プラットフォームで採用されているセキュリティメカニズムを正しくサポートしなければならない。IoT を大規模に展開するために必要なネットワークの相互作用が自動化されているため、オペレーターは、セキュリティ方針に準拠した方法で Thing が公開され利用されることを保証する必要がある。

軽減策:

TD の作成者は、できる限り、WoT バインディングテンプレートで提供されている、入念なチェック済みの通信メタデータを用いるべきである。WoT バインディングテンプレートでカバーされていない IoT エコシステム用に TD を生成する場合、IoT プラットフォームのすべてのセキュリティ要件が満たされていることを保証する必要がある。

§10.2 WoT スクリプト API のセキュリティとプライバシーに関するリスク

WoT ランタイムの実装と WoT スクリプト API には、システムへの悪意のあるアクセスを防ぎ、マルチテナント方式の Servient におけるスクリプトを分離するメカニズムがあるべきである。より具体的には、WoT スクリプト API を WoT ランタイムの実装と併用する場合は、以下のセキュリティとプライバシーに関するリスクを考慮し、推奨される軽減策を実装すべきである。

§10.2.1 クロススクリプトのセキュリティとプライバシーに関するリスク

基本的な WoT のセットアップにおいては、WoT ランタイム内で実行されるすべてのスクリプトは信頼できると見なしたうえで製造者が配信を行うため、実行する各スクリプトのインスタンス間で厳密な分離を行う必要はない。しかし、デバイスの性能、展開のユースケースシナリオ、リスクレベルによっては、そうすることが望ましい場合がある。例えば、あるスクリプトで機密性の高いプライバシーに関わる PII データが扱われてい

て、十分な監査が行われていれば、同じシステム内の他のスクリプトがランタイム中に侵害された場合にデータが露出するリスクを最小限に抑えるために、そのスクリプトを残りのスクリプトのインスタンスから分離することが望ましいかもしれない。別の例は、一つの WoT デバイス上に異なるテナントが共存している場合である。このケースでは、WoT ランタイムのインスタンスでそれぞれ異なるテナントが提供されているため、それらを分離する必要がある。

軽減策:

プライバシーに関わるデータや、その他のクリティカルなセキュリティデータをスクリプトで扱う場合には、WoT ランタイムはスクリプトのインスタンスとそのデータを分離すべきである。同様に、WoT ランタイムの実装では、ある WoT デバイスに複数のテナントがある場合には、WoT ランタイムのインスタンスとそのデータの分離を行うべきである。このような分離は、デバイスで利用できるプラットフォームのセキュリティメカニズムを用いて WoT ランタイム内で実行できる。詳細に関しては、WoT セキュリティとプライバシーに関するガイドライン仕様 [WOT-SECURITY] の「WoT Servient シングルテナント」と「WoT Servient マルチテナント」の章を参照のこと。

§ 10.2.2 物理デバイス直接アクセスのセキュリティとプライバシーに関するリスク

直接公開されているネイティブなデバイスのインターフェースをスクリプトで使用できる場合は、スクリプトが侵害されたり誤動作したりすると、基盤となる物理デバイス（および潜在的に周辺環境）が被害を受ける可能性がある。そのようなインターフェースで、入力に対する安全性のチェックが不足していれば、基盤となる物理デバイス（または環境）は安全ではない状態となる可能性がある。

軽減策:

WoT ランタイムは、ネイティブなデバイスのインターフェースをスクリプトの開発者に直接公開することを避けるべきである。代わりに、WoT ランタイムの実装では、ネイティブなデバイスのインターフェースにアクセスするためのハードウェア抽象化レイヤーを提供すべきである。このようなハードウェア抽象化レイヤーは、デバイス（または環境）を安全でない状態にする可能性のあるコマンドの実行を拒否すべきである。さらに、スクリプトが侵害された場合の物理的な WoT デバイスへの被害を減少させるために、特定のスクリプトに対して公開またはアクセスできるインターフェースの数を、スクリプトの機能に応じて最小限に抑えることが重要である。

§ 10.3 WoT ランタイムのセキュリティとプライバシーに関するリスク

§ 10.3.1 プロビジョニングと更新のセキュリティリスク

WoT ランタイムの実装で、製造後のプロビジョニングや、WoT ランタイム自身、スクリプト、または関連するデータ（セキュリティ証明書を含む）の更新がサポートされている場合、それは主要な攻撃ベクトル（attack vector）になる可能性がある。攻撃者は、更新やプロビジョニングの工程の間に上記の要素を変更しようとしたり、攻撃者のコードとデータのプロビジョニングを直接行ったりすることができる。

軽減策:

製造後のプロビジョニングやスクリプト、WoT ランタイム自身、または関連するデータの更新は、安全な方法で行うべきである。安全な更新と製造後のプロビジョニングに関する推奨は、WoT セキュリティとプライバシーに関するガイドライン仕様 [WOT-SECURITY] にある。

§ 10.3.2 セキュリティ証明書保管のセキュリティとプライバシーに関するリスク

通常、WoT ランタイムは、ネットワークで動作するために、WoT デバイスにプロビジョニングを行ったセキュリティ証明書を保管する必要がある。攻撃者がこれらの証明書の機密性や完全性を侵害できれば、資産にアクセスできるようになったり、他の WoT モノ、デバイス、またはサービスになりすましたり、DoS（Denial-Of-Service）攻撃を仕掛けたりすることができる。

軽減策:

WoT ランタイムは、プロビジョニング済みのセキュリティ証明書を安全に保管し、完全性と機密性を保証すべきである。一つの WoT 対応デバイスに複数のテナントがある場合、WoT ランタイムの実装では、各テナントのプロビジョニング済みのセキュリティ証明書の分離を保証すべきである。さらに、プロビジョニング済みのセキュリティ証明書が侵害されるリスクを最小限に抑えるために、WoT ランタイムの実装は、プロビジョニング済みのセキュリティ証明書にクエリを実行するスクリプトの API を公開すべきではない。そのような証明書（または、それを利用するけれども公開はしないという、ずっとましな抽象的操作であっても）には、それらを用いるプロトコルバインディングの実装のみがアクセスできるべきである。

§ A. 最近の仕様変更

§ 勧告案からの変更

- 規定的な変更はなく、軽微な編集上の修正あり、外部参照には変更なし。

§ 最初の勧告候補からの変更

用語の項を参考情報に。

- 要約とはじめにを再編成。
- プライバシーに関する議論と軽減策を拡大。
- 図中のテキストを、文中のテキスト変更に適合。
- 定義を更新・拡張。参考文献を更
- 新。
  - 規定的な参考文献を追加: RFC2046
  - 規定的な参考文献を削除または参考情報の項に移動: IANA-RELATIONS, MQTT, RFC4395, RFC6838, RFC7049, RFC7231, RFC7252
- アクセシビリティのフィードバックに基づいて図の色を調整。

誤植や大文字の用法など、編集上の軽微な修正。

§ 最初の公開草案からの変更

- ユースケースを改訂・拡張。
- 要件を再編成。
- 抽象的なアーキテクチャの定義。
- 用語を改訂・明確化。実装と展開への追
- 加。
- セキュリティとプライバシーに関する留意点の追加。

§B. 謝辞

この文書への貢献に対し、Michael McCool、Takuki Kamiya、Kazuyuki Ashimura、Sebastian Käbisch、Zoltan Kis、Elena Reshetova、Klaus Hartke、Ari Keränen、Kazuaki Nimura、Philippe Le Hegaret に特に感謝する。

この文書の改善につながったサポート、技術情報、提案に対し、W3C のスタッフ、および W3C Web of Things 利害団体（WoT IG）とワーキンググループ（WoT WG）のすべての関係者に感謝する。

WoT WG は、［WOT-PIONEERS-1］［WOT-PIONEERS-2］［WOT-PIONEERS-3］［WOT-PIONEERS-4］などの刊行物の形で学術的イニシアチブとして開始された「Web of Things」の概念に関する先駆的な取り組みにも感謝する。これにより、2010 年から Web of Things の国際ワークショップ (https://webofthings.org/events/wot/) が毎年開催されようになった。

最後に、WoT IG の創設から 2 年にわたってリードし、Thing Description を含む WoT 構成要素の概念にグループを導いてくれた Joerg Heuer に特に感謝する。

§C. 参考文献

§C.1 規定的な参考文献

[RFC2046]
Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types. N. Freed; N. Borenstein. IETF. November 1996. Draft Standard. URL: https://tools.ietf.org/html/rfc2046

[RFC2119]
Key words for use in RFCs to Indicate Requirement Levels. S. Bradner. IETF. March 1997. Best Current Practice. URL: https://tools.ietf.org/html/rfc2119

[RFC3986]
Uniform Resource Identifier (URI) : Generic Syntax. T. Berners-Lee; R. Fielding; L. Masinter. IETF. January 2005. Internet Standard. URL: https://tools.ietf.org/html/rfc3986

[RFC3987]
Internationalized Resource Identifiers (IRIs) . M. Duerst; M. Suignard. IETF. January 2005. Proposed Standard. URL: https://tools.ietf.org/html/rfc3987

[RFC5234]
Augmented BNF for Syntax Specifications: ABNF. D. Crocker, Ed.; P. Overell. IETF. January 2008. Internet Standard. URL: https://tools.ietf.org/html/rfc5234

[RFC8174]
Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words. B. Leiba. IETF. May 2017. Best Current Practice. URL: https://tools.ietf.org/html/rfc8174

[RFC8259]
The JavaScript Object Notation (JSON) Data Interchange Format. T. Bray, Ed.. IETF. December 2017. Internet Standard. URL: https://tools.ietf.org/html/rfc8259

[RFC8288]
Web Linking. M. Nottingham. IETF. October 2017. Proposed Standard. URL: https://httpwg.org/specs/rfc8288.html

§C.2 参考情報の参考文献

[CoRAL]
The Constrained RESTful Application Language (CoRAL) . Klaus Hartke. IETF. March 2019. Internet-Draft. URL: https://tools.ietf.org/html/draft-hartke-t2trg-coral

[CoRE-RD]
CoRE Resource Directory. M. Koster; C. Bormann; P. van der Stok; C. Amsuess. IETF. 13 June 2019. Internet-Draft. URL: https://tools.ietf.org/html/draft-ietf-core-resource-directory-21

[ECMAScript]
ECMAScript Language Specification. Ecma International. URL: https://tc39.github.io/ecma262/

[HCI]
The Encyclopedia of Human-Computer Interaction, 2nd Ed. Interaction Design Foundation. 2013. URL: https://www.interaction-design.org/literature/book/the-encyclopedia-of-human-computerinteraction-2nd-ed

[HTML]
HTML Standard. Anne van Kesteren; Domenic Denicola; Ian Hickson; Philip Jagenstedt; Simon Pieters. WHATWG. Living Standard. URL: https://html.spec.whatwg.org/multipage/

[IANA-RELATIONS]
Link Relations. IANA. URL: https://www.iana.org/assignments/link-relations/

[IANA-URI-SCHEMES]
Uniform Resource Identifier (URI) Schemes. IANA. URL: https://www.iana.org/assignments/urischemes/uri-schemes.xhtml

[IEC-FOTF]
Factory of the future. IEC. October 2015. URL: https://www.iec.ch/whitepaper/pdf/iecWPfuturefactory-LR-en.pdf

[IOT-SCHEMA-ORG]
Schema Extensions for IoT Community Group. URL: https://www.w3.org/community/iotschema/

[ISO-IEC-2382]
Information technology - Vocabulary. ISO. 2015. URL: https://www.iso.org/obp/ui/#iso:std:isoiec:2382:ed-1:v1:en

[ISO-IEC-27000]
Information technology - Security techniques - Information security management systems - Overview and vocabulary. ISO. 2018. URL: https://www.iso.org/obp/ui/#iso:std:iso-iec:27000:ed5:v1:en

[ISO-IEC-29100]
Information technology - Security techniques - Privacy framework. ISO. 2011. URL: https://www.iso.org/obp/ui/#iso:std:iso-iec:29100:ed-1:v1:en

[JSON-LD11]
JSON-LD 1.1. Gregg Kellogg; Pierre-Antoine Champin; Dave Longley. W3C. 16 March 2020. W3C Candidate Recommendation. URL: https://www.w3.org/TR/2020/CR-json-ld11-20200316/

[LINKED-DATA]
Linked Data Design Issues. Tim Berners-Lee. W3C. 27 July 2006. W3C-Internal Document. URL: https://www.w3.org/DesignIssues/LinkedData.html

[LWM2M]
Lightweight Machine to Machine Technical Specification: Core. OMA SpecWorks. August 2018. Approved Version: 1.1. URL: http://openmobilealliance.org/release/LightweightM2M/V1_1-20180710-A/OMA-TS-LightweightM2M_Core-V1_1-20180710-A.pdf

[MQTT]
MQTT Version 3.1.1 Plus Errata 01. Andrew Banks; Rahul Gupta. OASIS Standard. December 2015. URL: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html

[NORMAN]
The Psychology of Everyday Things. Donald A. Norman. Basic Books. 1988.

[OCF]
OCF Core Specification. Open Connectivity Foundation. April 2019. Version 2.0.2. URL: https://openconnectivity.org/developer/specifications

[REST]
REST: Architectural Styles and the Design of Network-based Software Architectures. Roy Thomas Fielding. University of California, Irvine. 2000. PhD thesis. URL: https://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf

[RFC4301]
Security Architecture for the Internet Protocol. S. Kent; K. Seo. IETF. December 2005. Proposed Standard. URL: https://tools.ietf.org/html/rfc4301

[RFC6202]
Known Issues and Best Practices for the Use of Long Polling and Streaming in Bidirectional HTTP. S. Loreto; P. Saint-Andre; S. Salsano; G. Wilkins. IETF. April 2011. Informational. URL: https://tools.ietf.org/html/rfc6202

[RFC6347]
Datagram Transport Layer Security Version 1.2. E. Rescorla; N. Modadugu. IETF. January 2012. Proposed Standard. URL: https://tools.ietf.org/html/rfc6347

[RFC6690]
Constrained RESTful Environments (CoRE) Link Format. Z. Shelby. IETF. August 2012. Proposed Standard. URL: https://tools.ietf.org/html/rfc6690

[RFC6749]
The OAuth 2.0 Authorization Framework. D. Hardt, Ed.. IETF. October 2012. Proposed Standard. URL: https://tools.ietf.org/html/rfc6749

[RFC7049]
Concise Binary Object Representation (CBOR) . C. Bormann; P. Hoffman. IETF. October 2013. Proposed Standard. URL: https://tools.ietf.org/html/rfc7049

[RFC7231]
Hypertext Transfer Protocol (HTTP/1.1) : Semantics and Content. R. Fielding, Ed.; J. Reschke, Ed.. IETF. June 2014. Proposed Standard. URL: https://httpwg.org/specs/rfc7231.html

[RFC7252]
The Constrained Application Protocol (CoAP) . Z. Shelby; K. Hartke; C. Bormann. IETF. June 2014.
Proposed Standard. URL: https://tools.ietf.org/html/rfc7252

[RFC7641]
Observing Resources in the Constrained Application Protocol (CoAP) . K. Hartke. IETF. September 2015. Proposed Standard. URL: https://tools.ietf.org/html/rfc7641

[RFC7744]
Use Cases for Authentication and Authorization in Constrained Environments. L. Seitz, Ed.; S. Gerdes, Ed.; G. Selander; M. Mani; S. Kumar. IETF. January 2016. Informational. URL: https://tools.ietf.org/html/rfc7744

[RFC8446]
The Transport Layer Security (TLS) Protocol Version 1.3. E. Rescorla. IETF. August 2018. Proposed Standard. URL: https://tools.ietf.org/html/rfc8446

[SAREF]
Smart Appliances REFerence (SAREF) ontology. ETSI. November 2015. URL: https://sites.google.com/site/smartappliancesproject/ontologies/reference-ontology

[VOCAB-SSN]
Semantic Sensor Network Ontology. Armin Haller; Krzysztof Janowicz; Simon Cox; Danh Le Phuoc; Kerry Taylor; Maxime Lefrancois. W3C. 19 October 2017. W3C Recommendation. URL: https://www.w3.org/TR/2017/REC-vocab-ssn-20171019/

[WOT-BINDING-TEMPLATES]
Web of Things (WoT) Binding Templates. Michael Koster; Ege Korkan. W3C. 30 January 2020. W3C Note. URL: https://www.w3.org/TR/2020/NOTE-wot-binding-templates-20200130/

[WOT-PIONEERS-1]
Mobile Service Interaction with the Web of Things. E. Rukzio, M. Paolucci; M. Wagner, H. Berndt; J. Hamard; A. Schmidt. Proceedings of 13th International Conference on Telecommunications (ICT 2006), Funchal, Madeira island, Portugal. May 2006. URL: https://pdfs.semanticscholar.org/3ee3/a2e8ce93fbf9ba14ad54e12adaeb1f3ca392.pdf

[WOT-PIONEERS-2]
Putting Things to REST. Erik Wilde. UCB iSchool Report 2007-015, UC Berkeley, Berkeley, CA, USA. November 2007. URL: http://dret.net/netdret/docs/wilde-irep07-015-restful-things.pdf

[WOT-PIONEERS-3]
Poster Abstract: Dyser - Towards a Real-Time Search Engine for the Web of Things. Benedikt Ostermaier; B. Maryam Elahi; Kay Romer; Michael Fahrmair; Wolfgang Kellerer. Proceedings of ACM SenSys 2008, Raleigh, NC, USA. November 2008. URL:

https://www.vs.inf.ethz.ch/publ/papers/ostermai-poster-2008.pdf

[WOT-PIONEERS-4]
A Resource Oriented Architecture for the Web of Things. Dominique Guinard; Vlad Trifa; Erik Wilde. Proceedings of Internet of Things 2010 International Conference (IoT 2010). Tokyo, Japan.
November 2010. URL: https://ieeexplore.ieee.org/abstract/document/5678452

[WOT-SCRIPTING-API]
Web of Things (WoT) Scripting API. Zoltan Kis; Daniel Peintner; Johannes Hund; Kazuaki Nimura. W3C. 28 October 2019. W3C Working Draft. URL: https://www.w3.org/TR/2019/WD-wot-scriptingapi-20191028/

[WOT-SECURITY]
Web of Things (WoT) Security and Privacy Guidelines. Elena Reshetova; Michael McCool. W3C. 6 November 2019. W3C Note. URL: https://www.w3.org/TR/2019/NOTE-wot-security-20191106/

[WOT-THING-DESCRIPTION]
Web of Things (WoT) Thing Description. Sebastian Kabisch; Takuki Kamiya; Michael McCool; Victor Charpenay; Matthias Kovatsch. W3C. 9 April 2020. W3C Recommendation. URL: https://www.w3.org/TR/2020/REC-wot-thing-description-20200409/

[Y.4409-Y.2070]
ITU-T Rec. Y.4409/Y.2070 (01/2015) Requirements and architecture of the home energy management system and home network services . ITU-T. January 2015. Recommendation. URL: https://www.itu.int/rec/T-REC-Y.2070-201501-I

↑