2018.9.5 Fujitsu

アーキテクチャ文書の修正案

Servient integration 方法に関して、プラグフェストで試したことをアーキテクチャ文書に反映する。Proxy、Directory、Binding に関する知見をまとめる。これらの内容を Requirement にも追加して、Building Block の話につなげる。

1. Introduction
   アーキテクチャ文書では、WoT で想定するユースケースについて説明したうえで、このユースケースから導き出される Requirement を明確にする。

2. Terminology
   NOTE:使われていない用語が多いように思うので、アーキテクチャ文書を修正するときに、それぞれの用語の必要性を考慮する。

3. Use cases
   NOTE:現時点では変更なしだが、あとの説明に合わせて図の構成は変えるべきかもしれない。
   デバイス、アプリケーション、プロキシ(ゲートウェイ、クラウド)の接続方法に関するバリエーションを記載する。まずは、スマートホームで全てのバリエーションを作る。他の分野でも同様であることを、別のユースケースで示すべき。(Lagally さんが Bundang 会議で示したユースケース等)

4. Functional requirements (informative)
   ユースケースで見ていたように、家電等のデバイスがリモコンから直接制御されたり、スマートフォンからゲートウェイ経由で接続されたり、アプリケーションやコントローラからデバイスをアクセス又は制御する方法には様々な方法があることがわかる。これらのユースケースから、Web of Things に求められる技術要件は以下のとおりである。

   NOTE:WoT は他の規格を橋渡しするという目標から、具体的なインタフェースに変換しやすい抽象インタフェースを提供するという話をいれるべき。(Protocol binding のところで入っているとも言えるが、Binding の範囲が WoT interface（TD を HTTP/CoAP/WebSocket/MQTT 等で通信する）の範囲なのか、他の IoT 規格（ECHONET Lite/BACnet/KNX 等との Binding を含むのか、含むとしたら同じ方法で実現するのか？）との Binding も含むのか、明確にしていく必要がある。

4.1 基本エンティティ（Servient）

WoT を構成する要素としては、デバイス、アプリケーション、プロキシがある。これらは WoT の基本機能を持っており、基本機能が共通のネットワークインタフェース(WoT Interface)をもつ。IoT のデバイス、アプリケーションに関するネットワークインタフェースについては、既に様々なものが存在しているため、WoT Interface は具体的なプロトコルに変換可能な抽象インタフェースとして定義される。

3 つの Servient に共通な機能を定義し、それらの機能で必要となるインタフェースが求められる。インタフェースとしては、デバイスの機能を表現するデバイスモデル（Thing Description みたいなもの）と、そこに記載される情報を転送するトランスファー・プロトコル（Transfer protocol）について規定する。

既に多くの規格があり、媒体となる共通のインタフェースがないと相互の接続ができない。抽象的なインタフェースを規定して、具体的なプロトコルを割り当てられるような仕組みが必要。データモデルについては、家電やビル設備、工場の生産設備等の分野では規格化（OCF、ECHONET Lite、KNX、BACnet、Modbus など）が進んでいる。また、こうしたボキャブラリを整備する動きもあり（iotschema.org)、データモデルの記述方法と外部のボキャブラリを活用する枠組みが必要となる。(→ Thing Description、Protocol Binding)。

また、基本エンティティ内にアプリケーションを実行するインタフェースが必要となる。

・Servient as a basic unit
・Application, device, and proxy servient
・Abstract interface for Servient
・Device model and Transfer protocol
・Description language for device model
・Protocol binding to convert from abstract interface to specific interface (ex. ECHONET Lite)
・Semantic integration to

4.2 Proxy Servient による大規模化可能なアーキテクチャ

デバイスをアプリケーションから直接アクセス、制御するだけでは、大規模なシステムに発展させることができない。ネットワークアーキテクチャから類推されるように、途中にスイッチングハブやアクセスポイントのような中継器を経由して、大規模接続していく必要がある。

プロキシでは接続されたデバイスの情報だけでなく、アプリケーションや他のプロキシがデバイスにアクセスするときのアドレスがプロキシになるように書き換える。

プロキシは Device Servient の最新情報を保持し、Device が休止時のときに、プロキシが代わりに応答することもある。
プロキシをファイアウォールで分けられる 2 つのネットワークを結ぶこともできる。企業内のローカルネットワークとインターネットにそれぞれプロキシを置き、クラウドから企業内(例えば、工場内)のセンサーや設備にアクセスすることを可能とする。2 つのプロキシは、ファイアウォールを超えるために必要な機能をもち、アプリケーションやデバイスの代わりに実現する。

NOTE：プロキシとディレクトリへの登録操作は同じにすべきか。別のインタフェースにすべきか。

Servient という基本ユニットに対して、このユニットを組み合わせることで大規模なシステムを構築可能である。ローカルネットワークとインターネットとの接続(複数のドメインにまたがる IoT デバイスネットワークの接続)

・接続される Device servient の TD 管理
・ルーティング機能(URL の書き換え)
・デバイスの最新状態の保持
・代理応答(デバイス休止時等に、Device の代わりにデータを返す)

・Delegation with the device servient
・Servient integration architecture
・Device management (directory)
・Device ID management (generation, deletion, delegation)

4.3 ディレクトリサービス、デバイスマネジメント(運用管理機能)
多数のデバイスが接続されるようになると、そのデバイスのアドレス(URL)を管理する必要がある。ディレクトリには、ネットワークに接続されるデバイスの ID やプロファイルが登録され、アプリケーションがデバイスに接続するときに、ディレクトリを検索することとで所望のデバイス情報が取得可能となる。
デバイスの灯篭は、マニュアルで登録する方法と、デバイスが自ら登録する方法がある。ディレクトリはそのためのインタフェースを持つことが必要である。また、アプリケーションからデバイス情報を取得するために、登録されているデバイスを一覧表示したり、検索する機能が必要となる。
高度な検索機能を実現するには、そこに登録されているデバイス情報も詳細でなければならない。そのため登録されるデバイス情報(TD)について詳細情報を記載でき

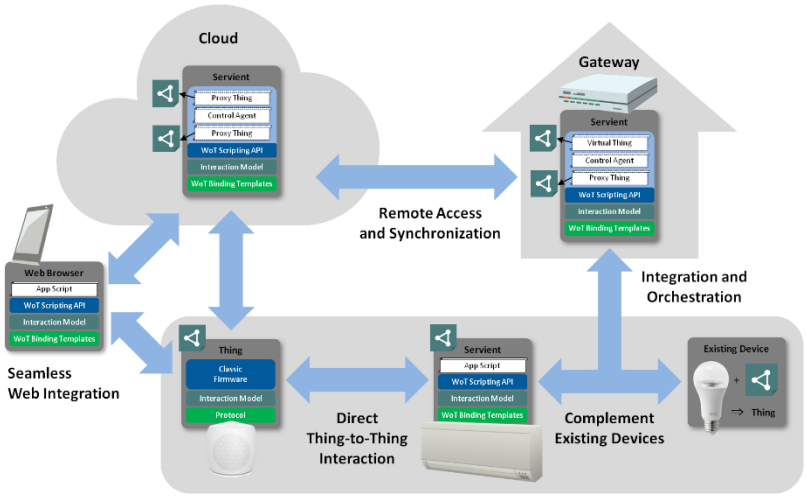
るようにするべきである(may)。

NOTE: デバイスが休止する場合、どのように対応するかは検討が必要である。休止状態を認める場合、デバイスの情報として状態を示すフラグ等が必要となる。これを TD として実現するのか、実装で賄うのかについては議論が必要。

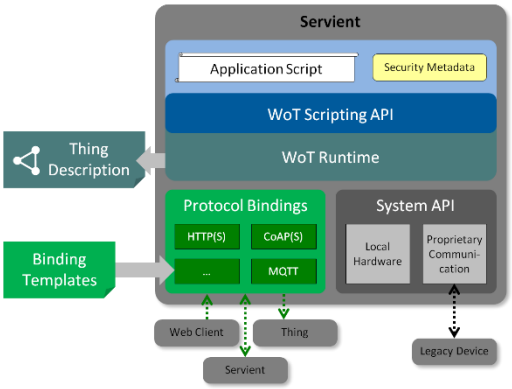
4.4 Security and Privacy

NOTE: this section is needed to be modified based on the discussion on the Security Taskforce.

5. WoT Building Blocks



6. WoT Servient Architecture



7. Deployment scenario

この章では、Servient がどのように連携して全体が動作するかを説明する。

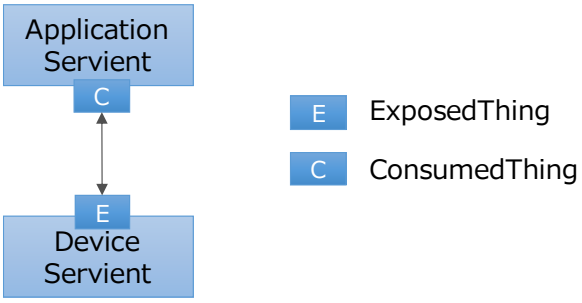
NOTE:　Terminology では、ExposedThing、ConsumedThing の定義があるが、アーキテクチャ文書のなかで、これらの機能や動作を具体的にしておくべき。
以下の説明では、Servient の ExposedThing と ConsumedThing のインタフェースを利用して Servient が通信するように記載している。これらのインタフェースの一部は ScriptingAPI に記載されている。
デバイスの状態管理をどうするのか？今の仕様では、デバイスがサスペンドする場合には存在しないことにしないといけない。Proxy にはデバイスの存在継続性を保持する機能も存在する。デバイスのインスタンスとしてステータスを持つ必要がある（初期⇔アクティブ⇔休止）。

これまで見てきたように、Servient にはアプリケーション、デバイス、プロキシがある。これらの組合せについて、deploy される組合せについてみていく。
・device and application direct connection
・proxy を介した接続
・ローカルエリアネットアークとインターネットのような複数のネットワークを接続

7.1 Servient integrations

7.1.1　Device and Application Servient

最も簡単な構成は、アプリケーションとデバイスが直接接続される。
デバイスの電源が入ったときに存在を広告するか、アプリケーションがデバイスを検索することで解決する。具体的には、ExposedThing は Advertise するか、ConsumedThing が Search する。同じネットワークにある場合には、ブロードキャストすることで実現される。ConsuedThing は、利用する ExposedThing の情報(TD)を保持する。アプリケーションは、ConsumedThing の API を通じて、TD に記載されるデバイス API をアクセスする。



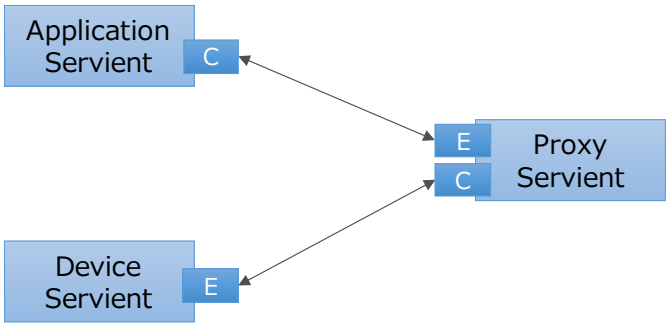
7.1.2　Proxy Servient

デバイスの数が増えてくるとアプリケーションでは、デバイスの TD を全て管理す

ることが難しくなる。これは、TD としては存在していても、デバイス自身が休止するなどして応答できないことがあるためである。Proxy はデバイスが応答できない状況の時に、代理で応答する機能を有す。

Proxy のアドレスは事前に知られており、デバイスは Proxy の consumed thing に対してユニキャストで通知する。Proxy はデバイスの情報(TD)を内部に保持し、その情報を Exposed thing を通じて開示する。その際に、Device servient へのアクセスが Proxy servient に届くように TD の URL を書き換える。Application Servient から Device servient の情報を検索されるときには、Proxy Servient はこの書き換えられた TD を返す。これにより、Device servient のコピーが proxy servient に存在するかのように見える。

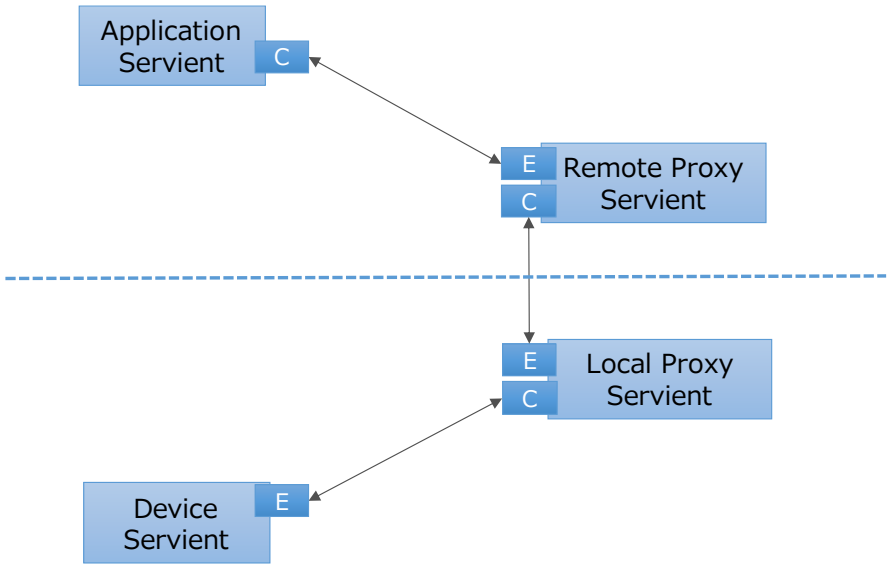
この Device Servient のコピーは Device servient の最新情報を保持するなどして、Device Servient が応答できない状況でも Proxy servient が代理で応答することができる。



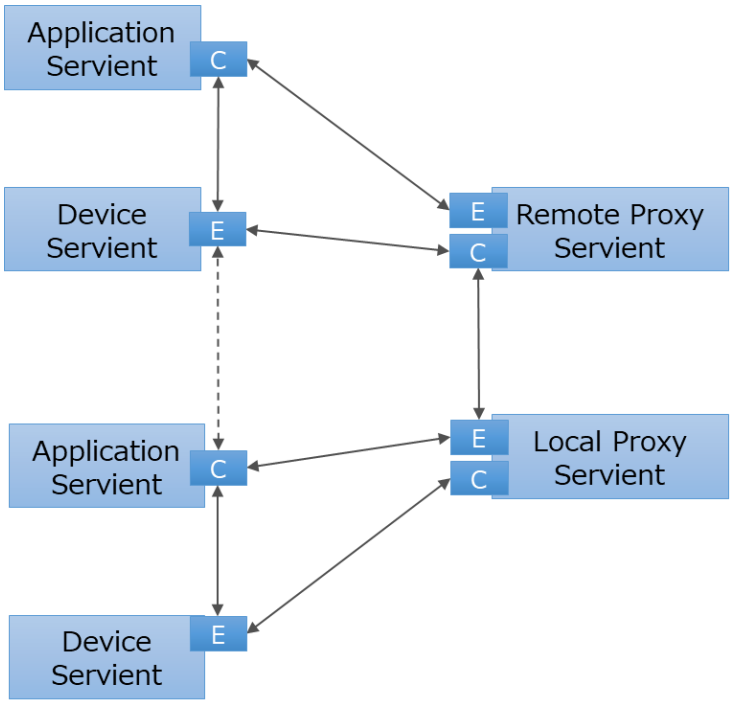
7.1.3 Bridging different networks

アプリケーションとデバイスが異なるネットワーク上に存在するときには、これらの間にファイアウォールが存在し、直接通信することができないため、ファイアウォールを超えるための仕組みが必要になる。

デバイスやアプリケーションが Firewall 越えをサポートすればよいが、ファイアウォールの設定は場所によって異なるため、デバイスやアプリケーションで対応するのは面倒である。Proxy をそれぞれのネットワークの配置し、Proxy 間でファイアウォール越を解決することで、デバイスとアプリケーションは、それぞれ Proxy との通信を実現するだけで、それらが個別にファイアウォールに対応する必要がなくなる。例えば、図では、デバイスがホームネットワークに存在し、アプリケーションがクラウドで提供される場合を意味している。点線の下側の Proxy はホームゲートウェイと呼ばれている装置であり、点線の上の Proxy はクラウドプラットフォームに対応する。このクラウドプラットフォームの API を利用してアプリケーションが動作する。このアプリケーションはスマートフォン等のブラウザで動作する。



これまでの構成をまとめると、図のようになる。それぞれのネットワークには、それぞれ Application servient と Device servient が直接接続されるパスと、Proxy 経由で接続されるパスが存在する。この 2 つのネットワークを結ぶものは Proxy のパスである。なお、ローカルネットワーク側のアプリケーションからインターネット側のデバイスをアクセスするのは、HTTP であれば可能であるのでこのケースは点線で示した。



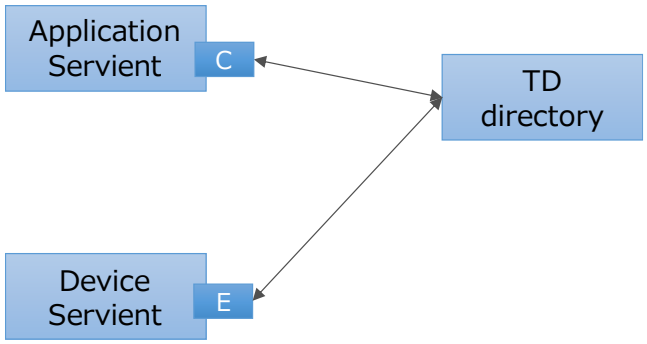
7.2　Thing Description Directory

Device Servient が多数存在するときには、その Device を検索する必要がある。

7.2.1 Registration from Device servient

Device servient(Shadow device も含めて)の数が多くなってくると、ネットワーク上に存在する Device の管理が必要となる。Directory は、Device servient が起動されるときに TD を登録することにより、Application servient から必要な Device servient を検索できるようにする。Directory は Application servient からの要求により、登録されている Device の一覧を返したり、特定の Device の URL を返す。Application servient はこの URL を利用して Device Servient に接続を行う。

Directory から所望の Device を見つけ出すには、さらに semantic search も可能な仕組みを備えることができる。



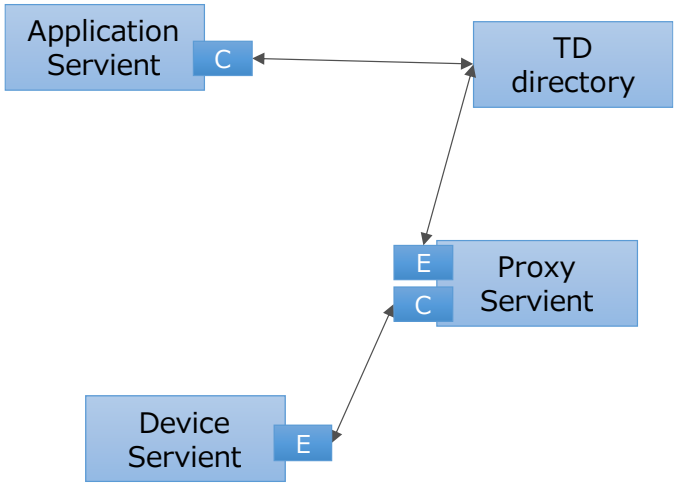
アプリケーションやデバイスの数が増えてくると、デバイスの広告とアプリケーションの検索が増えて、ブロードキャスト通信が増えてしまう。また、ブロードキャストメッセージが届く範囲でないと、アプリケーションはデバイスの存在をすることができないため、ディレクトリを導入する。

ディレクトリは URL が事前に何らかの方法で知られており、デバイスは初めてネットワークに接続されたときに Exposed Thing を通じてディレクトリに登録する。アプリケーションは Consumed Thing を通じて、ディレクトリを検索することによって目的のデバイスの URL を見つけることができる。

7.2.2 Registration from proxy servient

Device servient ではなく、Proxy Servient に登録されている Shadow device を directory に登録することができる。Proxy servient の ExposedThing からディレクトリに対して Shadow device を登録する。

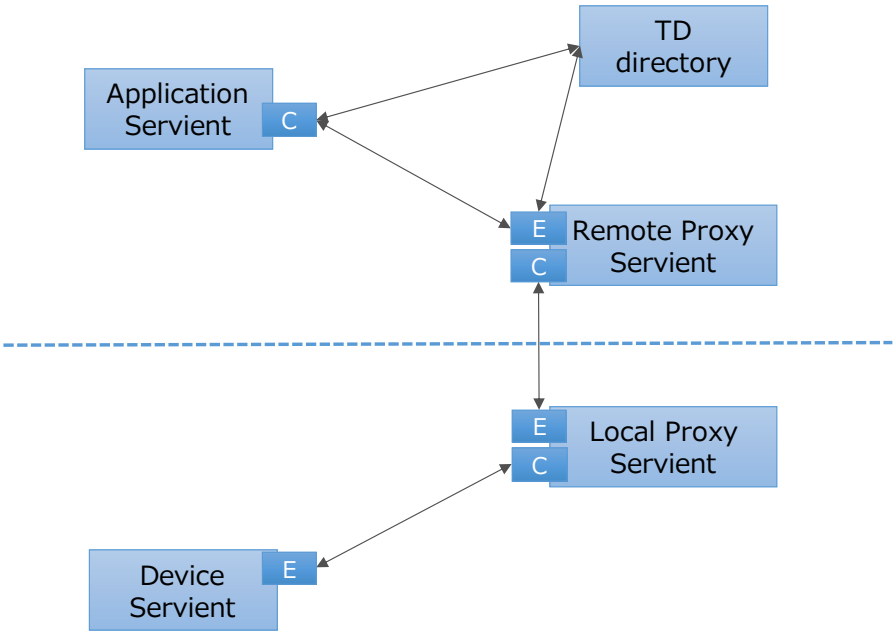
このケースでは、Device Servient は Directory に直接登録しないので、Application servient からはおの存在は直接見えない。Application servient は Directory から Shadow device の URL を取得して Proxy servient 経由で Device Servient を制御する。



Shadow が Proxy にあるという説明を追加するか、Shadow を使わないで説明する。

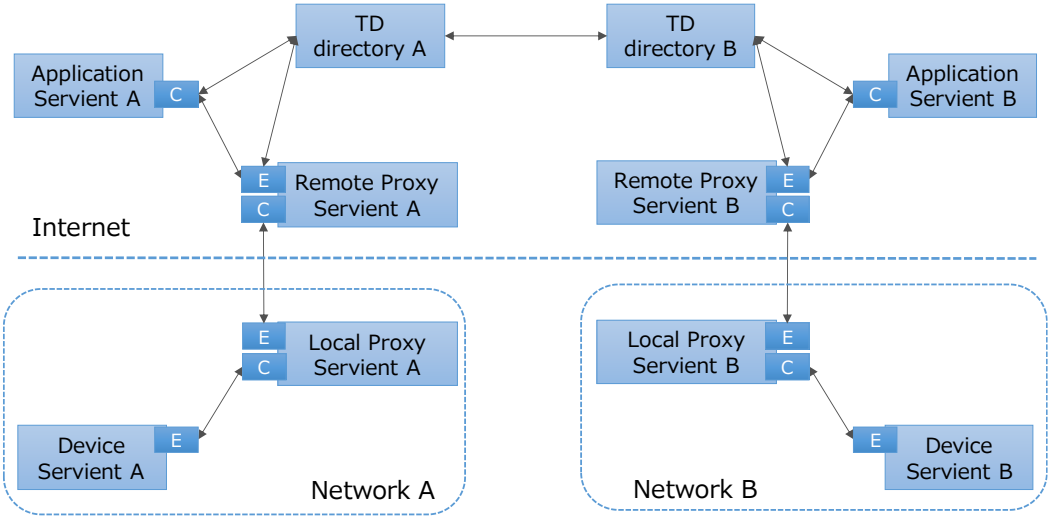
7.2.3 Registration from remote proxy

複数のネットワークにまたがるケースでは、Device Serivent の Shadow が 2 つの Proxy Servient に生成される。リモートネットワークにある Directory に対しては、remote Proxy Servient が登録を行うことで、アプリケーションから検索可能となる。なお、Device Servient の TD はローカルネットワークにある Directory に対して、直接または local Proxy servient 経由で登録することが可能である。



7.2.4 Synchronize two directories

異なるサービス事業者間でお互いの Device servient を利用可能にすることがある。この時には、2 つの Directories を連携することで、例えば、Directory A に接続されている TD を、Directory A が Directory B に同期させることで、Application Servient B から Directory B を通じて、Device Servient A を検索可能になる。

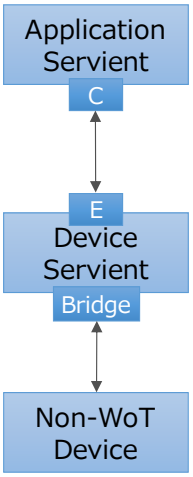


7.3 Legacy entities integration

WoT interface をサポートしない Legacy device を接続する方法について述べる。Servient とつなげるためには、ExposedThing のインタフェースを実現する必要がある。Non-WoT Device のインタフェースを変更できない場合には、他の Servient を利用して Non-WoT Device に対応するインタフェースを実現する必要がある。

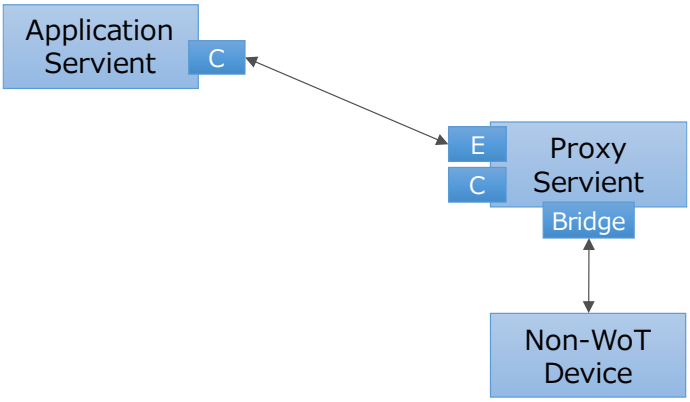
7.3.1 Bridging Legacy device

Non-WoT Device を Device Servient としてアクセスできるように、Device Servient にインタフェースを変換する Bridge を追加する。Bridge は Non-WoT device の既存インタフェースを利用して、TD を作成し、WoT interface(ExposedThing のインタフェース)を実現する。



7.3.2 Legacy Bridge on Proxy Servient

Bridge は Proxy Servient に持たせることも可能である。

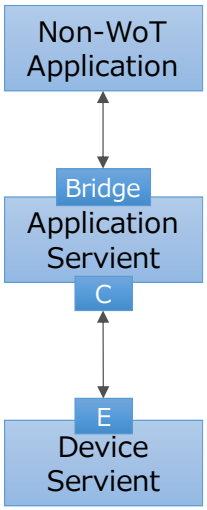


7.3.3 Legacy bridge for applications

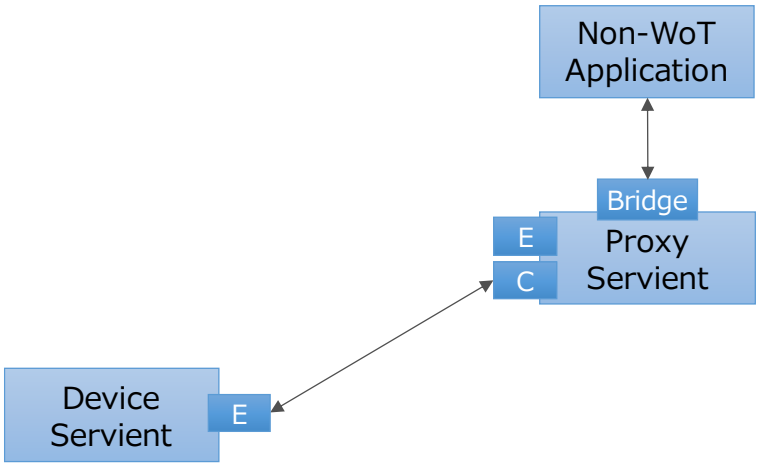
Device と同様に WoT interface を持たない Non-WoT application 向けの Bridge を考えることも可能。この Bridge にはいくつかの方法がある。

oneM2M や他の IoT プラットフォームのように、独自のデバイスモデルを持つ場合には、Bridge はそのデータモデルとの橋渡しをする変換を行う。データモデルを持たない場合には、インタフェースを変換する Bridge を作成する。

NOTE: Oracle IoT Cloud Service は独自のデータモデルを持っており、Bridge はこの変換を行った。富士通はプロキシ上にこの Bridge を作成した。



7.3.4 Legacy bridge for application on proxy



8 Security and Privacy Considerations

9 Summary

Appendix