一般社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
IEICE Technical Report
CS2021-66,IE2021-25(2021-11)

# 仮想 IoT システムによる日欧にまたがる見守りシステムのプロトタイプ実装と実証評価

金井 謙治 [1)]　中村 健一 [1)]　中里 秀則 [1)2)]

1) 早稲田大学理工学術院総合研究所 〒169-8555 東京都新宿区大久保 3-4-1
2) 早稲田大学 基幹理工学研究科 〒169-8555 東京都新宿区大久保 3-4-1

E-mail: 1) k.kanai@aoni.waseda.jp, 3) nakazato@waseda.jp

**あらまし**　本稿では、都市 OS モデルの実装例の一つとなり得る仮想 IoT システム「VirIoT」の紹介と、そのユースケースの一つとして、見守りシステムの VirIoT 上での実装例について説明する。本見守りシステムについて、プロトタイプ実装を行い、日欧の 5 か所の拠点に実際に配備を行い、システムの性能検証結果を紹介する。
**キーワード**　仮想 IoT システム，スマートシティ，見守りシステム，エッジコンピューティング

# Prototype implementation and its evaluation of EU and JP cross border person watching system using virtual IoT system

Kenji KANAI[1)]　Kenichi NAKAMURA[1)]　and　Hidenori NAKAZATO[2)]

1) Waseda Research Institute for Science and Engineering, Waseda University

3-4-1 Okubo, Shinjuku-ku, Tokyo, 169-8555 Japan

1) Faculty of Science and Engineering, Waseda University
3-4-1 Okubo, Shinjuku-ku, Tokyo, 169-8555 Japan

E-mail: 1) k.kanai@aoni.waseda.jp, 3) nakazato@waseda.jp

**Abstract**　In this paper, we introduce the Virtual IoT System, namely VirIoT, that we develop as one of the actual software implementations for the City OS model. In addition, to show the use case of VirIoT, we discuss the design implementation of EU and JP cross-border person watching system that runs on VirIoT. Finally, we actually deploy the prototype person watching system to five EU and JP sites and validate the system performances.
**Keywords**　VirIoT IoT System，Smart City，Person Watching System，Edge Computing

## 1. はじめに

近年、AI やビッグデータを活用して社会の在り方を根本から変える都市設計の動きが国際的にも急速に進展しており、我が国では内閣府を中心とした「まるごと未来都市」であるスーパーシティの実現に取り組んでいる[1]。このスーパーシティでは、Mobility as a Service (MaaS)に代表される移動サービス、物流、医療・介護、防犯など、都市の住民の生活全般に関わる領域を広くカバーすることで、住民の Quality of Life (QoL)の向上が期待されている。

このようなスーパーシティを実現するための要素技術の一つとしてデータ連携基盤の設計が挙げられており、内閣府ではこのデータ連携基盤のリファレンスアーキテクチャとして「都市 OS モデル」を公開している[2]。都市 OS モデルでは、都市間のサービス連携や横展開を可能とする「相互運用」、都市や地域が生成するデータを仲介する「データ流通」、新たな機能やアーキテクチャの更新容易にする「拡張容易性」を特に重要な機能要件としてまとめている。また、これらを可能とするためには、データ連携基盤の設計段階で下記のような点を考慮するように提案している。

- ✓ データやサービス提供するための Application Programming Interface (API)を外部に公開する点
- ✓ 多種多様なデータを取り扱うために、取り扱うデータモデルを定義する点
- ✓ 都市間にまたがってデータ流通するためのデータ仲介サービス（ブローカー）を保持する点
- ✓ マイクロサービスに代表されるようにビルディングブロック方式にて各機能を構築する点

一方で、都市 OS 自体は、具体的なデータ連携基盤の実装例までは明記しておらず、あくまでリファレンスアーキテクチャの提案に留まっている。具体的な実装例としては、例えば NEC では、NEC 都市 OS として、オープンソースプラットフォームである FIWARE[3]を利用したデータ連携基盤のサービス提供の開始を始めつつある[4]。

筆者らもまた、都市 OS モデルの実装例の一つとして、これまでデータ連携基盤の研究開発を行ってきており[5, 6, 7]、Virtual IoT Platform (VirIoT)として、そのソフトウェアを GitHub にて公開している[8]。筆者らが提案する VirIoT では、NEC 都市 OS と異なり、FIWARE が提供するデータブローカーである NGSIv2 Orion [9]や NGSI-LD OrionLD[10]のみならず、NGSI-LD の別の実装例である Scorpio [11]や oneM2M ベースの Mobius [12]、また、単純な MQTT ブローカーといったような、様々なブローカーやデータモデルに対応しており、これら異なるデータモデルを持つブローカー間でもデータ流通や相互運用可能な点が特徴として挙げられる。

このようなデータ連携機能のみに限らず、都市 OS モデルの重要な機能要件の一つである拡張容易性についても検討している。具体的には、AI によるデータ分析といったデータ処理機能についてもマイクロサービスとして扱い、データ利用者にとって必要となるデータ（例えば、単純なセンサデータのみならず、加工・分析が加えられたデータ）を取得可能な「仮想 IoT デバイス」として、IoT デバイスを仮想的に振舞わせる機能「ThingVisor」も内包している。この ThingVisor によって、データ利用者、サービス提供者は、物理 IoT デバイスやデータ処理そのものを気にする必要が無くなり、自身が必要となる「データ」に着目してデータ取得やデータ流通が可能となる。さらには、ThingVisor を Web ブラウザ上の Graphical User Interface (GUI)を介して、簡単に仮想 IoT デバイスを開発することができるツール「ThingVisor Factory [7]」も具備しており、より迅速かつ容易なサービス設計も実現している。

本稿では、VirIoT のユースケースとして、スーパーシティのサービスの一つである高齢者のひとり歩きや見守りを想定し、VirIoT 上での見守りシステムのプロトタイプ実装例を説明すると共に、実際に日欧の 5 か所（東京、石川、熊本、スペイン、フランス）に展開した際の性能評価について報告する。

## 2. 仮想 IoT システムについて

本章では、筆者らが研究開発している仮想 IoT システム「VirIoT」について概説する。なお、参考文献[5, 6, 7]にて、より詳細な説明が記載されているため、省内な内容はそれらの文献に譲ることとする。

### 2.1. 仮想 IoT システムの概要

図 1 に仮想 IoT システムの概要図を示す。図に示すように、仮想 IoT システムの主要な構成要素として、ThingVisor、ブローカー、vSilo の 3 点が挙げられる。

まず、ThingVisor では、データ利用者やサービス提供者に対して、物理 IoT デバイスを仮想的な IoT デバイスとして振舞わせる機能を担う。具体的な ThingVisor の主要な機能として、物理 IoT デバイスからのデータ取得、統計処理や画像処理といったデータ分析に関わるデータ処理、VirIoT 内で扱うための共通データモデルへのメッセージ変換、さらには、VirIoT 内のブローカーへのデータ発行（Publish）機能が挙げられる。また、ThingVisor の開発においては、サービス提供者が複雑なデータ処理を簡易に実装できるように、ThingVisor Factory のような設計ツールの利用も可能としている。ThingVisor Factory では、予め提供されている機能群を、Web ブラウザ上の GUI から、マウスを利用しドラッグアンドドロップで選択することができ、選択した機能群を線で結ぶことによって、ThingVisor の設計を可能としている。

次に、仮想サイロである vSilo では、データ利用者やサービス提供者が希望する IoT プラットフォームで取り扱うデータモデルへの変換を行う。先に述べたように、VirIoT 内では、VirIoT が選択する共通データモデルで表現されたデータを取り扱うため、最終的にデータ利用者やサービス提供者が取り扱い可能なデータモデルへと、さらに変換する必要がある。この機能を vSilo が担う。また、図 1 からわかるように、データ利用者やサービス提供者は直接、仮想 IoT デバイスと接続されているわけではなく、ブローカー、仮想サイロを経由して、接続される。そのため、仮想サイロは、データ利用者やサービス提供者が構築した仮想 IoT システムへアクセスするためのエンドポイントとしての役割も果たす。つまり、データ利用者やサービス提供者は、この仮想サイロが提供するエンドポイントを経由して、仮想 IoT デバイスの制御が可能となり、例えば、物理 IoT デバイスの状態を変更する、処理パラメータを更新する、といったアクチュエーションも提供される。

最後に、VirIoT のブローカーでは、VirIoT 内で生成されたデータ（ThingVisor で生成されたデータ）の VirIoT 内でのデータ流通の機能を担う。VirIoT では、MQTT に代表される Pub/Sub メッセージングモデルを採用しており、VirIoT 内のデータは、トピック名を介して取得することが可能となる。なお、本ブローカーでは、データプレーンの情報のみならず、コントロールプレーンの情報（例えば、ThingVisor、vSilo のメタ

情報）のやり取りについても、Pub/Sub メッセージモデルを採用している。

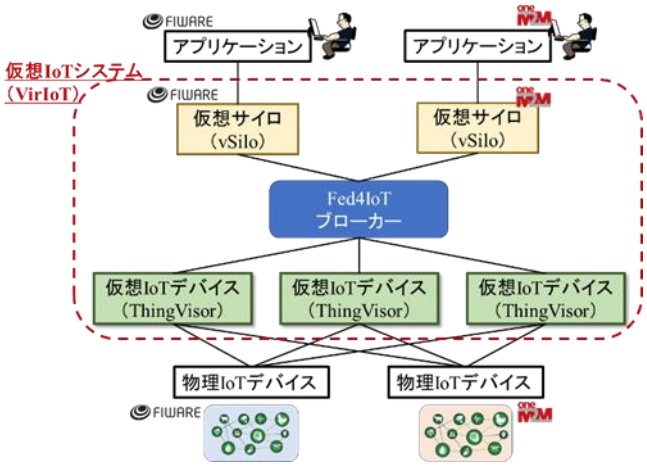


図 1. Fed4IoT が提案する仮想 IoT システムの概要図[5]

## 2.2. 仮想 IoT システムの利点について

現在、仮想 IoT システムは、[8]に示す GitHub のページにて、ソースコードの公開および利用が可能となっている。本仮想 IoT システムは、Docker[13]および Kubernetes[14]上で動作するように実装されている。ThingVisor、vSilo、ブローカーのそれぞれは Docker コンテナとして実装され、コンテナのデプロイやインスタンス化といったオーケストレーション機能については、Kubernetes の API を利用している。特に Kubernetes を利用することで、仮想 IoT システムは、例えば、オートスケーリングやオートヒーリングといった Docker コンテナの管理面において、大きな利点を得ることできる。また、仮想 IoT システム自身としての利点については、ThingVisor や vSilo のポータビリティ性が大きいと考える。アプリケーションに必要となる主要なサービスの要素をモジュールとして ThingVisor や vSilo としてパッケージ化することにより、物理 IoT デバイスへのアクセス方法さえ公開することによって、国境を越えてあらゆる都市や地域において、同様のサービスを迅速に展開できることが可能となる。つまり、すでにサービスインしている場所に加え、新規にサービスインする場所を追加する場合には、アプリケーションに必要となる ThingVisor や vSilo を Kubernetes の API を利用して呼び出すのみで、迅速に展開することができる。

さらに、仮想 IoT システムでは、ThingVisor や vSilo を展開する場所を「ゾーン（例えば、EU ゾーンや JP ゾーン）」として、明示的に選択する機能も持つ。このゾーンによる選択によって、ThingVisor や vSilo のより最適に近い場所へ展開することが可能となり、不必要なトラヒックや通信遅延を抑制することが可能となる。

以上の仮想 IoT システムについて、ユースケースを利用した検証として、第 3 章以降では、高齢者のひとり歩きといった見守りシステムをユースケースに想定し、VirIoT 上での実装例および日欧にまたがってアプリケーションを展開した際の性能検証について述べる。

## 3. VirIoT を利用した見守りシステムの実装例

VirIoT を利用した見守りシステムの概要図を図 2 に示す。ここで想定する見守りシステムは、都市部に設置された監視カメラを利用し、見守り対象者が発見された地理位置情報を出力・可視化するようなシステムである。そのため、以降では監視カメラへのアクセス可能であることを前提に進める。また、サービス提供者側は、警察関係者や地方自治体が警察関係者へ協力を促すといったことを想定しており、監視カメラへのアクセス権を保有しているものとする。

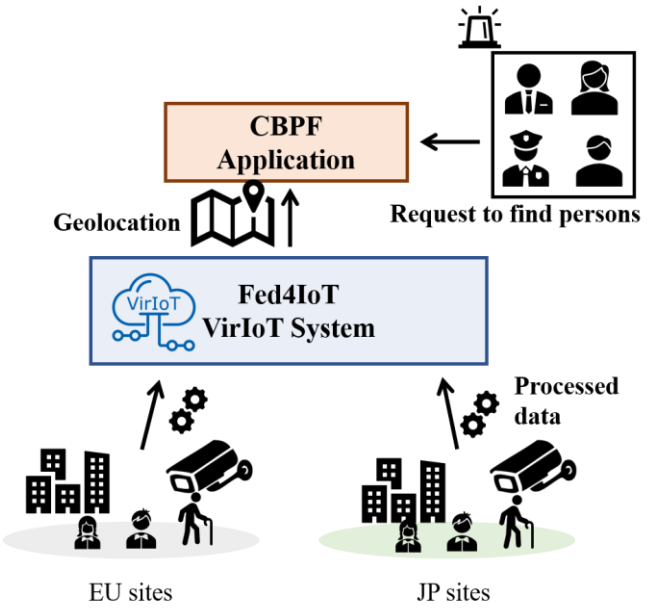


図 2. VirIoT を利用した見守りシステムの概要図.

### 3.1. ThingVisor の実装戦略について

画像を利用した見守りシステムにおいては、画像に写りこむ人に対する個人情報保護（例えば、GDPR）を考慮して、アプリケーションを実装する必要がある。特に、日欧間でのデータ流通において、画像に写っている人からの同意が得られない限りは、画像データの共有は難しい。そのため、エッジコンピューティングの考えに従い、画像データの地産地消を図る。つまり、見守り対象の人物の地理位置を出力するような仮想 IoT デバイスを ThingVisor によって実装する。具体的な主な処理内容としては、監視カメラから取得した画像を利用した人物検知、人の顔領域の特徴点の抽出、

要求された見守り対象者の顔領域の特徴点の抽出、それぞれの顔領域の特徴点のマッチング、さらには、VirIoT で扱えるように ThingVisor としてデータモデルへ変換する機能（TV-Converter）が挙げられる。なお、今回は、顔領域の特徴点抽出およびマッチングについては、顔認識機能として実装する。

これらの処理を VirIoT における ThingVisor として実装する場合には、次の 3 種類の戦略が考えられる。

① **全てをカメラ機能として実装（スマートカメラ）**：まず、最も単純な方法としては、これらの機能をカメラ側に内包してしまい、ThingVisor からは、処理結果を REST API のような API で取得する実装方法（図 3）である。ThingVisor では、カメラから処理結果を受け取った後は、VirIoT で扱うデータモデルへ変換し、ブローカーへデータ転送する（Relay-ThingVisor）。そのため、この方式においては、ThingVisor は最低限な機能のみを持つことから、最も実装が簡単である反面、カメラ自体に機能を持たせる必要があるため、拡張容易性は低いと言える。一方で、画像データは全てカメラ内に閉じることができるため、個人情報保護の観点からもベストと言える。

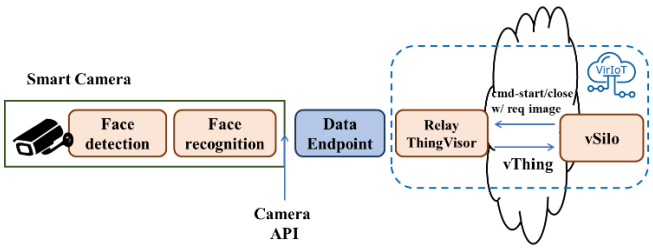


図 3. 全てをカメラ機能として実装する例（Relay ThingVisor）

② **全ての機能を 1 つの ThingVisor として実装（Monolithic ThingVisor）**：次に、必要となるデータ処理機能を一つの ThingVisor として全て実装する方法（図 4）である。この Monolithic ThingVisor では、カメラからの画像を取得後、1 つの Docker コンテナ内で全ての処理を実行する方式となる。そのため、この方式においては、実装については①のスマートカメラに次いで実装しやすい反面、各処理機能を更新しようとすると、ThingVisor 自体を更新する必要があるため、拡張容易性はそこまで高くはない。また、カメラから画像データを取得することになるため、ThingVisor の実行場所もカメラに近いエッジサーバといった場所に限定する必要がある。

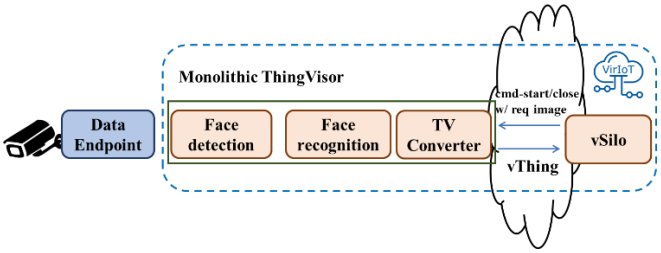


図 4. 全ての処理機能を単一の ThingVisor として実装する例（Monolithic ThingVisor）

③ **各処理機能をそれぞれ ThingVisor として実装（Service chain ThingVisor）**：最後の実装例は、Monolitich ThingVisor のように、全ての処理機能を 1 つの ThingVisor（Docket コンテナ）として実装するのではなく、各処理機能をそれぞれ個別の ThingVisor として実装し、各 ThingVisor をネットワークで結合するサービスチェインによる実装方法（図 5）である。この方式では、各処理機能を個別の ThingVisor として実装するだけでなく、ネットワークで結合する必要があるため、実装については、最も複雑であると言える。一方で、各機能をマイクロサービス化して分割して実装しているため、各機能の更新は、ThingVisor 全体に影響を及ぼすことは無いため、拡張容易性は高いと言える。さらに、各機能の実行場所についても、個人情報保護を考慮する処理については、監視カメラに物理的に近い場所で実行を行い、それ以外については、クラウドで実行など、処理機能の配備についても柔軟性が高いと言える。なお、このような複雑な Service chain ThingVisor の実装においては、ThingVisor Factory を利用することを想定している。その利用による設計の様子を図 6 に示す。

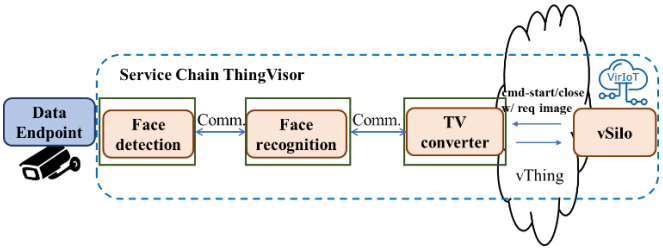


図 5. 処理機能単位に ThingVisor として実装し、通信によって結合する例（Service chain ThingVisor）

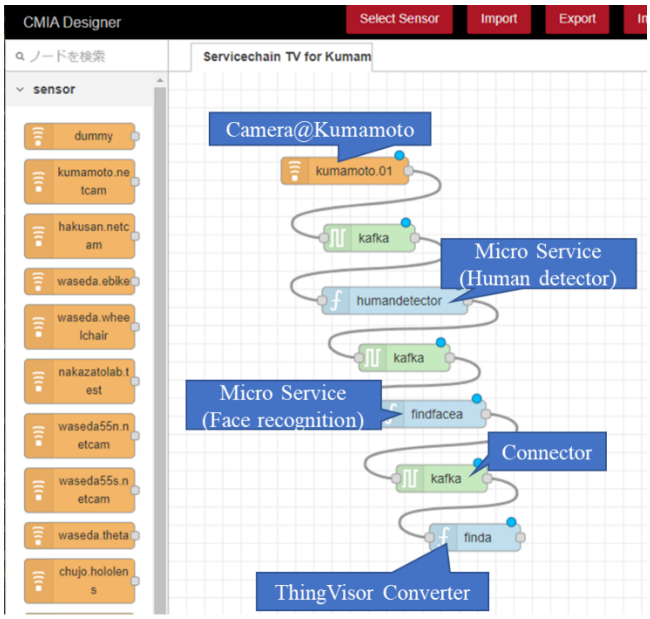


図 6. ThingVisor Factory による Service chain ThingVisor の設計例の様子

### 3.2. ThingVisor, vSilo の配備戦略について

先に説明したように、画像データの取り扱いには、個人情報保護を考慮する必要があるため、今回は、画像データについては地産地消を想定し、クラウド等で画像データの共有を行わない。それに伴い、ThingVisor の配備についても意識する必要がある。図 7 に、各 ThingVisor の実装方式の配備戦略の例を示す。まず、①のカメラに全ての機能を持たせる場合（Relay-ThingVisor）は、画像データがカメラより外に流れないため、ThingVisor の配備先については、エッジ、クラウドのいずれでも可能である。次に、②の Monolithic ThingVisor の場合では、カメラ画像を ThingVisor に入力する必要があるため、クラウドの利用ではなくエッジで処理することが望ましいと言える。最後の③の Service chain ThingVisor の場合は、前述したように、個人情報保護に関わる処理（例えば、顔認識）については、エッジで処理を行い、その後の特徴点のマッチング（顔認識）については、クラウドで処理を行うといったより柔軟性の高い配備ができる。(ただし、顔認識では、ユーザ名といった個人情報の特定は想定しておらず、あくまで個人だと特定できない処理を想定している)。

vSilo については、ユーザのアプリケーションと通信するために、クラウドへ配備することとする。これらの配備戦略については、VirIoT の機能として、ゾーンを指定した ThingVisor、vSilo の配備が可能となっており、各エッジ、クラウドの計算ノードに予めゾーンとなる名前を定義することにより、ThingVisor, vSilo の地理情報を考慮した配備が可能となる。

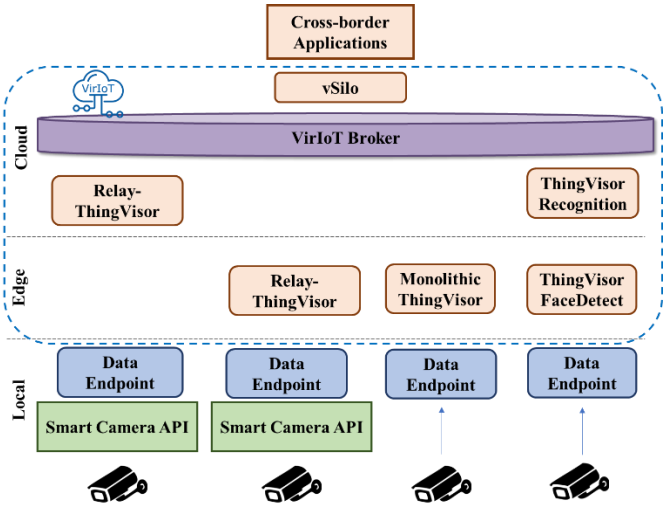


図 7. ThingVisor, vSilo の配備戦略の概要図

## 4. 日欧に展開した際のシステム性能評価

最後に、第 3 章で説明した見守りシステムについて、実際に日欧の 5 か所（ムルシア（スペイン）、グラス（フランス）、東京、白山（金沢）、熊本）に監視カメラおよびエッジ計算ノード、日欧の Azure Cloud Servers に、VirIoT および見守りシステムを配備した。顔検知、顔認識については、Microsoft FaceAPI[15]を利用し、①~③に説明した 3 種類の ThingVisor を実装し、システムの性能検証を行った。

### 4.1. 異なる ThingVisor 実装の比較検証

システムの性能評価の一例として、まずは、図 8 に示すように異なる ThingVisor 実装における比較検証として、ThingVisor の処理時間を評価検証した。処理時間の計測には、エッジサーバとして Intel NUC を利用し、全て同一のマシン上で処理時間の計測を行った。図 8 からわかるように、①スマートカメラ用の ThingVisor は、データ形式の変換と転送のみなため、最も処理時間が短く、全ての処理機能を担う Monolithic ThingVisor の処理時間が最も長い。また、Service Chain ThingVisor は処理の分散が図れている様子が分かる.

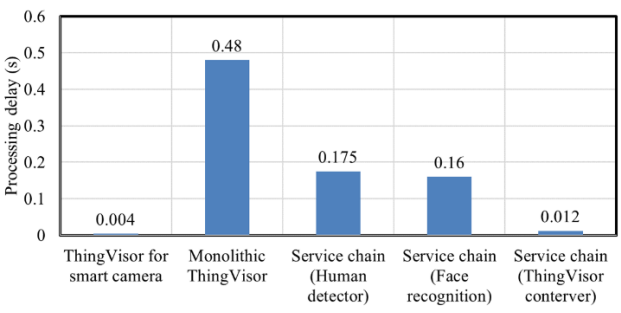

図 8. 異なる ThingVisor 実装における処理時間の比較

### 4.2. 監視カメラと ThingVisor の遅延検証

次に、エッジコンピューティングの効果を検証するために、監視カメラと ThingVisor 間の通信遅延について、その配備場所に伴う影響を比較検証した。ThingVisor は、最も単純なスマートカメラの利用を想定したものとし、データのやり取りに発生する通信遅延のみを検証した。具体的には、ThingVisor を日欧の 2 か所のサーバに、また監視カメラを日欧の 5 か所にそれぞれ配備を行い、計 10（5×２）の配備パターンにおけるそれぞれの通信遅延を比較検証した。ThingVisor とスマートカメラがやり取りする情報には、まずは、スマートカメラを活性化させるためのコマンド（start command）、処理結果の取得（Get data）、さらには、スマートカメラを終了するためのコマンド（close command）の 3 種類が存在する。いずれのメッセージも HTTP を利用した RESTful API で実装されている。各処理の通信遅延を図 9 に示す。図 9 からわかるように、より物理的に近い場所に ThingVisor と監視カメラを配備することによって、大幅に通信遅延を削減できていることがわかる。この結果により、VirIoT によって、適切に各機能を配備することが冗長な通信遅延を大幅に抑えられることが可能となり、その有効性を確認した。

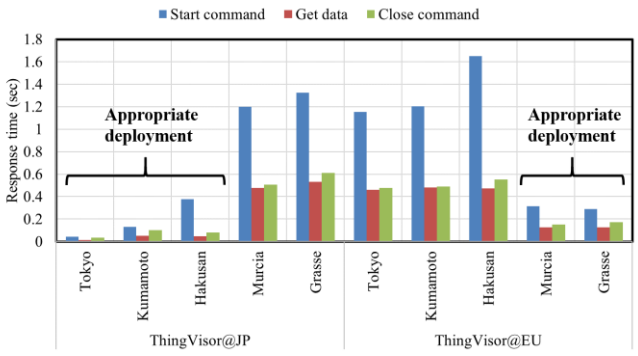


図 9. ThingVisor の配備場所による監視カメラと ThingVisor 間の遅延の比較結果.

## 5. まとめ

本稿では、都市 OS モデルの実装例の一つとなり得る仮想 IoT システム「VirIoT」の紹介と、そのユースケースの一つとして、見守りシステムの VirIoT 上での実装例について説明した。さらに、プロトタイプ実装を行い、日欧の 5 か所の拠点に実際に配備を行い、システムの性能検証結果の一例を示した。今後は、各 ThingVisor の実装方式の違いによる効果をさらに定量的に比較検証を行っていくと共に、VirIoT の異なるユースケースについても示していく予定である。


## 謝辞

本研究成果は，戦略的情報通信研究開発推進事業（国際標準獲得型)「スマートシティアプリケーションに拡張性と相互運用性をもたらす仮想 IoT-クラウド連携基盤の研究開発 (Fed4IoT) 【JPJ000595】」によるものである。


## 文　献


[1] 内閣府国家戦略特区，“「スーパーシティ」構想とは”[online]: https://www.chisou.go.jp/tiiki/kokusentoc/supercity/openlabo/supercitykaisetsu.html

[2] 内閣府，“都市 OS”[online]: https://www8.cao.go.jp/cstp/stmain/a-whitepaper3_200331.pdf

[3] FIWARE: The Open Source Platform for Our Smart Digital Future [online]: https://www.fiware.org/

[4] NEC, “NEC 都市 OS” [online] : https://jpn.nec.com/press/202109/20210908_02.html

[5] 金井，吉田，金光，中里，横谷，向井，中村，上杉，“Things as a Service を実現する Fed4IoT プラットフォームの研究開発”，電子情報通信学会 信学技報 CS2019-69, pp.39-44, 2019 年 11 月.

[6] A. Detti, H. Nakazato, J.A.M. Navarro, G. Tropea, L. Funari, L. Petrucci, J.A.S. Segado, K. Kanai, “VirIoT: A Cloud of Things That Offers IoT Infrastructures as a Service,” Journals of Sensors 2021, Vol.21, Issue 19, Sep. 2021.

[7] K. Kanai, H. Nakazato, H. Kanemitsu, A. Detti, “ThingVisor Factory: Thing Virtualization Platform for Things as a Service,” Proc. of the Workshop on Cloud Continuum Services for Smart IoT Systems, CCIoT’20, pp.7-12, Nov.2020.

[8] Fed4IoT VirIoT [online]: https://github.com/fed4iot/VirIoT

[9] Orion Context Broker [online]: https://github.com/telefonicaid/fiware-orion

[10] Orion Context Broker (Linked Data Extensions) [online]: https://github.com/FIWARE/context.Orion-LD

[11] Scorpio NGSI-LD Broker [online]: https://github.com/ScorpioBroker/ScorpioBroker

[12] Mobius IoT Server Platform [online]: http://developers.iotocean.org/archives/module/mobius

[13] Docker [online]: https://www.docker.com/

[14] Kubernetes [online]: https://kubernetes.io/