Vol.2020-OS-150 No.5
2020/7/30

共有 IoT 資源利用アプリケーションのための データフロープログラミング

村木暢哉[1]　木戸善之[2]　高橋慧智[3]　山田拓哉[4]　伊達進[2]
梅谷麗[1]　石橋靖嗣[1]　下條真司[2]

概要：今日，カメラ，温度センサ，データの一時的な集約を担う IoT ゲートウェイなど多様な IoT 資源がネットワークに接続され，利用可能である．このような IoT 資源から取得される様々なデータを遠隔地から監視，収集・解析，あるいは遠隔地の IoT 資源を制御・操作するといった IoT アプリケーションが数多く開発されつつある．しかし，今日利用可能な IoT アプリケーションでは，その利用する IoT 資源はその IoT アプリケーションに占有的に利用されることを前提としている場合が多い．また，複数の IoT アプリケーションで IoT 資源を共有可能である場合でも，開発者は利用する IoT 資源の情報を予め知り得ないため，IoT 資源の探索・情報収集，その情報を用いた IoT 資源同士のデータフローの構築，IoT 資源へのデータフローの設定をプログラミングしなければならない．本研究では，上述の背景から，多様な IoT 資源が複数の IoT アプリケーション間で共有される状況を想定し，アプリケーション開発者が IoT アプリケーション内のデータフローのみに着眼した開発を可能とする共有 IoT 資源プラットフォームを提案する．具体的には，提案共有 IoT 資源では，共有 IoT 資源を探索，管理する機能とデータフローを記述，実行する機能を分離する．これにより，アプリケーション開発者の IoT アプリケーションの開発を容易にする．本論文の評価では，提案手法と従来手法で，同じ機能をもつ IoT アプリケーションを開発して開発量を比較し，提案手法では開発量が低減していることを確認する．

1. はじめに

近年，データ発生源となるセンサやカメラ，データの一時的な集約と転送や計算処理が可能な IoT ゲートウェイなどの IoT 資源が急増しており， 2018 年の 228 億台から，2025 年には 416 億台に達すると言われている[1]. 今日では，そのような IoT 資源を活用して，農作物の生育状態収集アプリケーション，防犯アプリケーションなど，様々な IoT アプリケーションが開発されつつある．

通常，設置された IoT 資源は，設置目的のみにしか使用されない．例えば自治体が防犯アプリケーションのために電柱に設置した防犯カメラおよび画像データは，防犯アプリケーションでしか利用されない．しかし防犯カメラからの画像データは様々な流用可能なデータとなりうる．流用可能先としては，認知症高齢者追跡アプリケーションというものが挙げられる．認知症を発症した高齢者は自分の意志で行動するが，目的，現在地を見失い，迷子になることで知られている．こうした認知症高齢者の探索，追跡には防犯カメラの映像を解析することが有効であると考えられるが，防犯カメラの使用目的からは逸脱するため，設置した本来の目的以外には利用できない．そのため，認知症高齢者追跡アプリケーションを作る場合，防犯カメラと同等のものを同じように設置する必要がある．

異なる IoT アプリケーションで同じ機能の IoT 資源を利用する場合，こうした重複状態を生み出す．更には，IoT 資源自体も常にデータを取り続けているわけではないため，遊休状態が発生している．重複状態を避け，遊休状態の IoT 資源を他の IoT アプリケーションから利活用するためには，遊休 IoT 資源を管理し，利用希望者に提供するための仕組み，つまり共有 IoT 資源プラットフォームが必要となる．

従来の IoT アプリケーションの開発では，利用する IoT 資源が予め決められているため，IoT 資源の間で発生するデータフローのみをプログラミングすることとなる．しかし，共有 IoT 資源を利用することが前提となる共有 IoT 資源プラットフォームでは，遊休 IoT 資源を探索し，動的に IoT 資源間を接続し，データフローを組み立てることが求められる．

そこで本研究では共有 IoT 資源を探索，管理する機能とデータフローを記述，実行する機能を分離した，共有 IoT 資源プラットフォームの提案を行う．それにより共有 IoT 資源を利用するアプリケーションの開発においても，データフローのみのプログラミングが可能な環境を目指す．また本稿では，簡易的な IoT アプリケーションコードを比較して，アプリケーション開発者の開発量の検証を行う．

2. 技術的課題と技術要素

IoT 資源を共有するためのプラットフォームでは, IoT アプリケーションから共有 IoT 資源および IoT 資源周辺の状況を操作，把握する必要がある．つまり手順としては，遊休 IoT 資源を探索，割当て，データ取得，解放，といった状態遷移が考えられる．それ以外にもデータの通信先，経路の制御, 共有 IoT 資源のアクセス制御などが挙げられる．

1 TIS 株式会社
TIS Inc.
2 大阪大学 サイバーメディアセンター
Cybermedia Center, Osaka University
3 奈良先端科学技術大学院大学 情報科学研究科
Graduate School of Information Science,
Nara Institute of Science and Technology
4 大阪大学 大学院情報科学研究科
Graduate School of Information Science and Technology,
Osaka University

以下，表 1 に共有 IoT 資源プラットフォームに必要な機能要件を列挙する．

表 1　共有 IoT 資源を利活用するための
プラットフォームで必要な機能要件

分類	概要
共有 IoT 資源の探索	利用可能な負荷状況であるか 目的の機能を提供するか 目的のエリアに存在するか 目的の精度を持つか
共有 IoT 資源の通信	どこへデータを送信するか どのように通信するか どの程度の帯域を要するか
共有 IoT 資源の可用性	利用不能になった場合にどうするか
共有 IoT 資源のセキュリティ	誰に対し公開されているか 誰がアクセス可能か

こうした機能，つまりプラットフォームの共有 IoT 資源群から状態などの情報を取得するための有効な手段として，宣言型プログラミングが挙げられる．

宣言型プログラミングは，プログラミング様式の一つで，処理の手順は記述せずに，データの性質，状態だけ宣言してプログラミングすることができる．つまり「座標(緯度，経度)から半径特定範囲内に存在するカメラから，JPEG 解像度 640x480，タイムスタンプ t0 から t1 までの画像を，解析サーバに送信，人の顔画像を抽出し，その後，特定の人間を抽出」といった内容を，カメラや計算機をどのように探すか，カメラにどうアクセスするか，カメラから計算機にどうアクセスするかなどを記述せずにプログラミングすることが出来る．具体的な言語仕様の例として SQL が挙げられる．SQL では，データベースから特定の条件のデータを集約して取得することができ，かつその取得方法や処理手順については記述する必要がない．SQL での開発は，データベースの論理的なアドレス，リレーショナルデータベースなのか，テキストファイルであるのかなどのデータへのアクセス方法は記述する必要がなく，データを取得することが可能となる．

共有 IoT 資源プラットフォームにおいても同様のことが要求される．共有 IoT 資源へのアクセス方法や，アクセス処理手順は，IoT 資源の種別によって異なる．それらを IoT アプリケーションの開発者がそれぞれに実装していると開発量が増大する．そのため共有 IoT 資源プラットフォームには，何らかの宣言型プログラミング様式に基づいた言語と，IoT 資源へのアクセス手段との分離が必要となる．

2.1 関連研究

宣言型プログラミング様式に基づいた言語によって対象を制御するソフトウェアとして、Terraform がある[2]．Terraform は，インフラストラクチャの構成管理ツールであり，利用者は，構成管理情報を HCL[3]という言語で記述する．Terraform を使う構成管理では，利用者は，インフラストラクチャの構築手順は記述しない．その代わりに利用者は，Terraform の実行後に構築，設定されているべきインフラストラクチャの状態を，宣言的に記述する．Terraform を実行すると，様々なインフラストラクチャのプロバイダに対応した Terraform プラグインが宣言を解釈し，インフラストラクチャのプロバイダの API を呼び出して実際の構築の手続きを行う．Terraform は非常駐型のコマンドラインプログラムであり，管理対象の実体情報はインフラストラクチャのプロバイダ側に存在する．共有 IoT 資源を利用する IoT アプリケーションでは，共有 IoT 資源の管理機能が必要であるが，こうした機能を Terraform が単体で持つことはできない．そのため，本稿では HCL と Terraform は利用していない．

Ansible も Terraform 同様，利用者はサーバやネットワークの設定手順ではなく，Ansible の実行後に設定されているべき状態を宣言的に記述する，インフラストラクチャの構成管理ツールである[4]．Ansible では，利用者は，構成管理情報を YAML 形式で記述する[5]．Ansible を利用する場合は，構成管理を行う対象が明確である必要がある．例えば，IP アドレスやポート番号，どのようなプロトコルでアクセスするかの情報が必要である．共有 IoT 資源を利用する IoT アプリケーションでは，実際に利用される IoT 資源を事前に明確に決定できないため，本稿では Ansible は利用していない．

次に，IoT アプリケーション開発に関する関連技術について述べる．IoT アプリケーションの開発で利用されるプログラミングツールとして，Node-RED がある[6]．Node-RED では，データの送受信や，データ受信したときの処理など，手続きをまとめたものはノードと呼ばれ，ノードを連結したものはフローと呼ばれる．IoT アプリケーションの開発者は，コードを記述する，または図 1 のような Node-RED のグラフィカルな画面上でデータの発生や処理，送信のノードを連結し，フローをプログラミングすることができる．開発者は，Node-RED がインストールされた IoT 資源に，Node-RED の画面や API を通じてフローを投入することで，IoT 資源に様々な処理を行わせることができる．

開発者が Node-RED を利用して IoT アプリケーションを開発する場合，利用する IoT 資源の情報が予めわかっている必要がある．具体的に IoT 資源の IP アドレス，ポート番号，プロトコルの情報が必要である．こうした情報は，IoT 資源間の接続，および作成したフローを IoT 資源へ投入する時に用いる．しかし共有 IoT 資源を利用するアプリケーションの開発では，開発者はプログラミングする時点で，利用する IoT 資源を決定することができない．なぜなら，要求する IoT 資源機能を提供する遊休 IoT 資源は，刻一刻と動的に変動しているため，プログラム動作中に利用でき

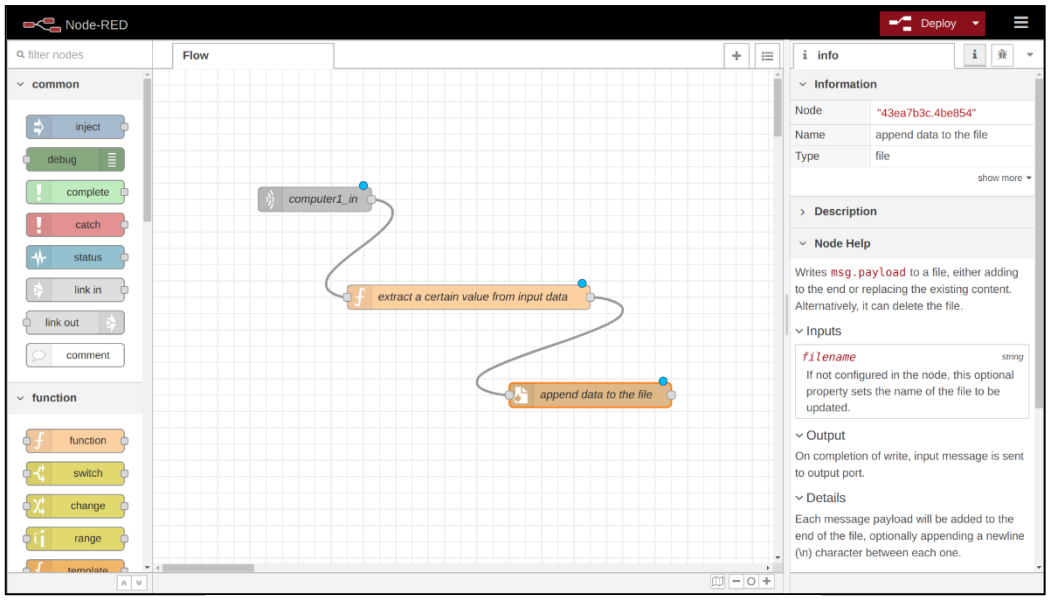

図 1　Node-RED 上での IoT アプリケーション開発

る IoT 資源を指定する必要がある．つまり Node-RED だけで共有 IoT 資源プラットフォームを実装しようとする場合，IoT アプリケーション側で，IoT 資源の IP アドレスなどのネットワーク情報の取得，負荷状況の収集，IoT 資源の選別などの機能を実装しなければならない．その上で Node-RED のフローのプログラミングを行わなければならず，結果としてプログラミングでの開発量が増加する．

3. 提案

本提案では，開発者が共有 IoT 資源の要件を宣言的に記述するフォーマットを定義し，そのフォーマットで記述された IoT アプリケーションを動作させるプラットフォームを実装する．共有 IoT 資源プラットフォームは，プラットフォームサーバ（図 4 の(1)），そして共有 IoT 資源で動作するエージェント（図 4 の(2)）で構成される．アプリケーションは，宣言的に記述したフォーマットを解釈し，動作する．記述した内容はデータフローのみであり，共有 IoT 資源の固有情報や，共有 IoT 資源同士の接続に関わるネットワーク情報は記述しない．プラットフォームサーバは，アプリケーションから要求された IoT 資源の機能，条件をもとに共有 IoT 資源を検索し，利用を決定する．そして利用する共有 IoT 資源は，アプリケーションから要求されたデータフロー通りに動作することとなる．

次に，アプリケーションとして記述する宣言型プログラミングについて述べる．アプリケーションのフォーマットは Ansible で採用されている YAML 形式に従い，そのサンプルを図 2 に示す．開発者がアプリケーションを開発する際には，共有 IoT 資源の固有情報や，共有 IoT 資源同士の接続に関わるネットワーク情報は記述せず，要件を宣言的に記述する．本研究で実装した YAML での要素を表 2 に示す．図 2 で示すコードはデータフローを意味する．共有 IoT 資源を node.name ごとに記述し，node.type として IoT 資源の種別を，node.location で探索する中心位置，node.radius で中心半径を記述する．これを解釈するプラットフォームサーバでは，IoT 資源の種別，エリアをもとに遊休 IoT 資源を探索することができる．またこうした宣言型プログラミングにより，IoT 資源ごとに設定された固有情報に縛られることなく，データフローに基づいたアプリケーション開発が可能となる．

```
nodes:
- name: camera
   type: CameraSource
   location: 35.696031,139.6883343
   radius: 1.0
   load: <0.5
   to:
      - docker
- name: docker
   type: DockerTransformer
   location: 35.696031,139.6883343
   radius: 1.0
   load: <0.3
   args:
      repo: registry.example.com/user/facedetect
   to:
      - fs
- name: fs
   type: FileSystemSink
   location: 35.696031,139.6883343
   radius: 1.0
   load: =0.0
   args:
      prefix: camera
      postfix: .jpg
      data_key: image
```

図 2　IoT アプリケーションのコードフォーマット

表 2　IoT アプリケーションのコードに記述する要素

要素	説明
nodes	共有 IoT 資源を示し，データの発生，変換，吸収を担う node の配列を示す.
node.name	アプリケーション内で有効な，共有 IoT 資源の名称を示す.
node.type	利用したい共有 IoT 資源の種別条件 ● Source はデータの源泉を示し，データが発生するセンサなどが該当する. ● Transformer はデータの変換を示し，データに対する計算を実行するサーバなどが該当する. ● Sink はデータの吸収を示し，データを保存するストレージなどが該当する.
node.location	利用したい共有 IoT 資源が存在する位置情報の中心点を示す.
node.radius	node.location からの存在範囲条件を示す.
node.load	共有 IoT 資源の負荷条件を示す.
node.args	共有 IoT 資源へのパラメータであり，種別毎に存在する.
node.to	アプリケーション内で node.name として定義されている，データを流す先の共有 IoT 資源の名称を示す.

4. 評価

本節では，提案手法である共有 IoT 資源プラットフォーム上で動作するアプリケーションを実際に開発し，開発量，具体的にプログラミングコードのステップ数を評価する．また比較対象としては，Node-RED を用いて共有 IoT 資源プラットフォームと同等の機能を実装したものとする．そのプラットフォーム上で動作する同等のアプリケーションを実装し，ステップ数を比較対象とする．

4.1 開発対象アプリケーションの構成と環境

評価では，一部共通した共有 IoT 資源を利用するアプリケーションを 2 種類作成する．図 3 に，アプリケーション 1 と 2 の構成を示す. アプリケーション 1 は，土壌センサ，日照センサおよび温度センサからデータを取得し，農作物の成長に関する周辺データを観測する IoT アプリケーションである. アプリケーション 1 は 10 個のセンサと 1 台の計算機，および 1 つのデータストアを利用する．アプリケーション 2 は，温度センサを用いた天候変動を観測する IoT アプリケーションである．アプリケーション 2 では，アプリケーション 1 が利用する IoT 資源のうち，2 つのセンサと 1 台の計算機を利用し，データストアを 1 つ利用する．

4.2 提案手法による IoT アプリケーションの開発

提案手法による共有 IoT 資源を利用したアプリケーション開発の構成は図 4 に示す通りである．開発者は，表 2 の内容に従って YAML ファイル（図 4 の(3)）を記述する．共有 IoT 資源プラットフォームサーバ（PF サーバ）は，HTTP サーバとして動作し，開発者は記述した YAML ファイルを PF サーバに投入する．その後，PF サーバは，指定されたエリアのセンサ群を探索し，使用するセンサを決定し，YAML ファイルに構成されたデータフローに従い，データへの処理，転送を行う．

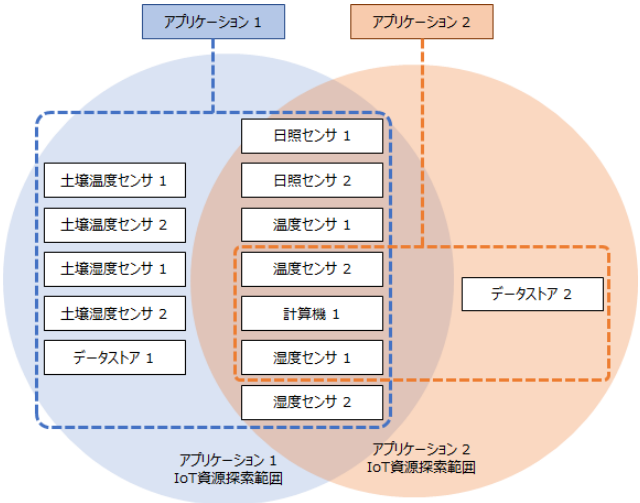

図 3　検証環境とアプリケーションの構成

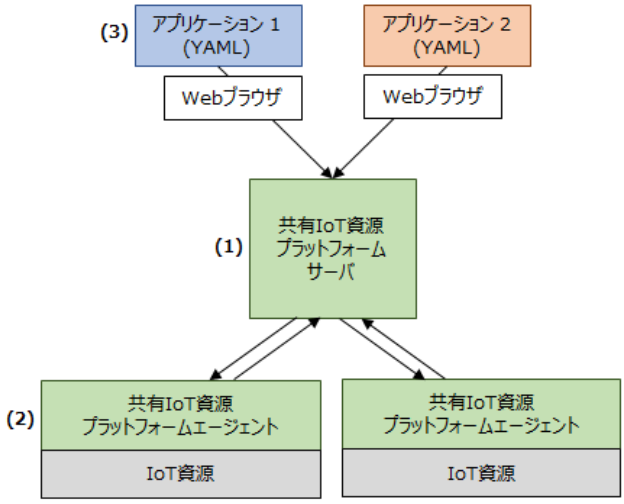

図 4　提案手法で開発した IoT アプリケーションの構成

4.3 Node-RED による IoT アプリケーションの開発

Node-RED を用いて提案手法と同等の機能を実装する場合，アプリケーション側に共有 IoT 資源を探索できる機能が必要となる．更に IoT 資源の種別，位置情報，負荷の条件から利用できる共有 IoT 資源を絞り込む機能を実装する．アプリケーションは，利用できる共有 IoT 資源群の IP アドレス，ポート，プロトコルを取得した後，作成したデータフロー（図 5 の(1)）を各 IoT 資源にインストールされた Node-RED（図 5 の(2)）が解釈し，データの処理，転送を行う．今回開発したアプリケーションでは，提案手法のプラットフォーム上で動作する 2 つのアプリケーションと同

等の機能を実装したため，データフローの他に，IoT 資源の種別，位置情報，負荷の条件から利用できる共有 IoT 資源を絞り込む機能を実装した．また，それぞれの IoT 資源は，位置情報，負荷情報，ネットワーク情報を問い合わせ可能な機能を持つとした（図 5 の(3)）．

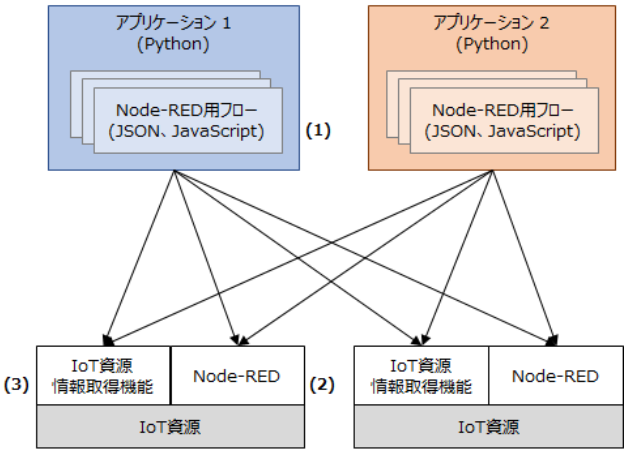

図 5　Node-RED で開発した IoT アプリケーションの構成

4.4 評価結果

提案手法と従来手法で，4.1 に示す IoT アプリケーションを開発した場合の開発量を，表 3 および表 4 に示す．

表 3　アプリケーション 1 に関する，提案手法による開発量と Node-RED による開発量の比較

機能	提案手法による開発量 (ステップ数)	Node-RED による開発量 (ステップ数)
全体	83	846
IoT 資源種別条件の記述	12	49
探索位置条件の記述	24	55
負荷条件の記述	12	11
ネットワークの記述	22	60
IoT 資源利用不可時の記述	0	24
Node-RED 特有の記述	0	509
その他	13	138

表 4　アプリケーション 2 に関する，提案手法による開発量と Node-RED による開発量の比較

機能	提案手法による開発量 (ステップ数)	Node-RED による開発量 (ステップ数)
全体	27	480
IoT 資源種別条件の記述	4	33
探索位置条件の記述	8	32
負荷条件の記述	4	11
ネットワークの記述	6	53
IoT 資源利用不可時の記述	0	24
Node-RED 特有の記述	0	212
その他	5	115

Node-RED での共有 IoT 資源を用いるアプリケーションでは，アプリケーションにデータフロー以外の機能を記述しなければならず，そのためステップ数は，提案手法よりもアプリケーション 1 と 2 でそれぞれ 10.2 倍，17.8 倍に増加することとなった．これは表 3，表 4 に内訳を書いているが，特に Node-RED 固有のフローを解釈するコードや，その他の内訳に含まれている雑多な処理が大半を占める．

一方，提案手法では，共有 IoT 資源を探索する機能を PF サーバ上に実装したため，アプリケーションでは大半がデータフローおよびデータの属性，種類などの定義のみの記述となりステップ数は少なくなっている．特に IoT 資源利用不可時の記述，つまりエラー処理については，これも PF サーバ側で吸収しているため，ステップ数は 0 となっている．

5. まとめ

IoT 資源を有効利用するためのプロジェクトや研究として，OpenIoT や Tacit Computing が存在するが，本稿では，IoT 資源が共有可能である場合に，IoT アプリケーションの開発者がどのようにアプリケーションを開発するかに着目した[7][8]．そして，開発者がデータフローのみを記述して，アプリケーションを容易に開発できる共有 IoT 資源プラットフォームを示すアプローチをとった．評価の結果，提案手法によって，共有 IoT 資源を利用するアプリケーションの開発量が低減されることを確認できた．

本稿で提案したプラットフォームでは，IoT アプリケーションの開発者は，利用する IoT 資源の条件を要件項目ごとに記述していたが，問い合わせ式の記述を可能とすることで，より簡潔に条件を指定できるようになることが考えられる．また，実際のプラットフォームの運用では，アプリケーションの増加に対するプラットフォーム自体の負荷分散や可用性の考慮などが必要となる．これらについては，今後の課題とする．

参考文献

[1] “The Growth in Connected IoT Devices Is Expected to Generate 79.4ZB of Data in 2025, According to a New IDC Forecast”. https://www.idc.com/getdoc.jsp?containerId=prUS45213219.
[2] “Terraform - Use Infrastructure as Code to provision and manage any cloud, infrastructure, or service”. https://www.terraform.io/.
[3] “HCL - HashiCorp Configuration Language”. https://github.com/hashicorp/hcl.
[4] “Ansible - The enterprise solution for building and operating automation at scale”. https://www.ansible.com/.
[5] “YAML - YAML Ain't a Markup Language”. https://yaml.org/.
[6] “Node-RED - Low-code programming for event-driven applications”. https://nodered.org/.
[7] “OpenIoT - The Open Source Internet of Things”. https://github.com/OpenIotOrg/openiot.
[8] M. Kataoka, N. Hosikawa, H. Noguchi, T. Demizu and Y. Yamato, "Tacit computing and its application for open

IoT era，" 2018 15th IEEE Annual Consumer Communications & Networking Conference (CCNC)，Las Vegas，NV，2018，pp．1-5，doi: 10．1109/CCNC．2018．8319240．