Vol.2021-DPS-188 No.20
Vol.2021-EIP-93 No.20
2021/9/10

FedIoT: Primitive API による 自律分散的な IoT システム間連携機構

嶋田 恵大[1,a)]　渡邊 大記[1,b)]　近藤 賢郎[2,c)]　寺岡 文男[3,d)]

概要：従来の IoT（Internet of Things）は目的別かつ closed な IoT システムを構築している．今後は IoT システムで得たセンサデータを外部ユーザに提供する IoTSP（IoT Service Provider）の出現が考えられる．本稿では，IoTSP を想定した IoTSP 間連携を実現するための機構である FedIoT を提案する．FedIoT では全 IoTSP で共通のオントロジ OntoFedIoT とインタフェースを定義し，これらをスキーマ化することで曖昧性を排除した IoTSP 間の連携を容易にした．FedIoT により，ユーザは広範囲かつ多種多様なデータを取得可能になる．本稿では FedIoT の Proof of Concept 実装を行い，基本性能を評価した．

FedIoT: An Autonomous and Decentralized IoT System Federation Mechanism with Primitive API

1. はじめに

従来の IoT（Internet of Things）は目的別かつ closed な IoT システムを構築している．例えば，工作機器の異常検知のために工場内に構築した IoT システムから得たセンサデータは，IoT システムを構築した企業のみが利用している．IoT システムを利用するユーザは，広範囲に設置したセンサから多種多様なセンサデータを取得し，データを多様な目的のために使用したいという需要がある．この需要に応えるためには，1 つの IoT システムでは以下の 2 つの理由から実現困難である．1 つ目は，1 つの IoT システムでカバーできる地理的領域の広さの問題である．例えば，ある IoT システムは関東地方，別の IoT システムは関西地方にしか気象センサを設置できず，各 IoT システムは限られた地理的領域の気象データのみの収集となる．2 つ目は，1 つの IoT システム内に設置されるセンサの多様さの問題である．例えば，ある IoTSP システムは関東地方に気象センサしか設置できず，別の IoT システムは関東地方に交通量センサしか調達できない場合がある．

そこで本稿では，ユーザに IoT サービスを提供する `IoTSP`（`IoT Service Provider`）が自律分散的に連携する機構である `FedIoT`（`Federated IoT`）を提案する．IoTSP は IoT システムで得たセンサデータを契約したユーザに提供する．例えば，IoTSP1 は気象センサを関東地方に設置し，IoTSP2 は気象センサを関西地方に設置し，IoTSP3 は交通量センサを関東地方に設置するとし，これらの IoTSP が自律分散的に連携する．これは，多数の ISP（Internet Service Provider）（AS（Autonomous System））が自律分散的に Internet を構成しているのと同様である．つまり，Internet ではユーザは 1 つの ISP と契約すれば世界中のエンドノードとの通信が可能になる．同様に FedIoT では，ユーザは 1 つの IoTSP と契約することで IoTSP 間の連携範囲でデータ取得が可能になる．FedIoT は IoTSP だけでなく，センサを所有しているが IoTSP のようなシステムを構成する余裕がない組織とも連携可能にする．このような組織を `Thing Provider` と呼ぶ．Thing Provider としては自治体や商店街，個人主等が考えられる．

FedIoT がユーザに提供する基本サービスとして以下の 2 種類を想定する．1 つ目は地理的領域とデータの種類を指定し，該当するセンサの ID（の集合）を取得するものである．2 つ目はセンサの ID（の集合）と時間範囲を指定し，

[1] 慶應義塾大学大学院理工学研究科
Graduate School of Science and Technology, Keio University
[2] 慶應義塾情報セキュリティインシデント対応チーム
Computer Security Incident Response Team, Keio University
[3] 慶應義塾大学理工学部
Faculty of Science and Technology, Keio University
[a)] dash@inl.ics.keio.ac.jp
[b)] nelio@inl.ics.keio.ac.jp
[c)] latte@itc.keio.ac.jp
[d)] tera@keio.jp

該当するセンサ値を取得するものである．ビッグデータ処理等はユーザの利用するアプリケーションに任せるとし，FedIoT の範囲外とする．これは Internet が End-to-End のデータ交換を提供し，具体的なサービスはアプリケーションに任せるのと同様である．

FedIoT を実現するためには以下の 3 つの課題が存在する．1 つ目は IoTSP 内でのセンサの設置場所や種類等を示すコンテキスト情報の表現形式である．このコンテキスト情報の表現形式が IoTSP ごとに異なる場合，異なる IoTSP が収集したセンサデータの利用が困難である．既存の表現形式には，オントロジにより統一的な表現をしている例 [1–3] が存在する．FedIoT では IoTSP 間で共有するオントロジ `OntoFedIoT` を定義し，スキーマ化する．Internet では IP による接続が可能であるのと同様に，共通オントロジにより IoTSP 間でデータへのアクセスを容易にする．2 つ目は IoTSP 間のインタフェース（I/F）である．この I/F が複雑であると，IoTSP 間の接続が困難である．そのため，FedIoT では I/F をできるだけ簡素にした Primitive API（Application Programming Interface）を定義し，スキーマ化する．3 つ目は IoTSP 間におけるコンテキスト情報の共有手法である．FedIoT では IoTSP がそれぞれのセンサに公開・非公開のポリシを付加することを可能にし，Internet における BGP（Border Gateway Protocol）のように IoTSP 間でコンテキスト情報を伝搬可能にする．

本稿では FedIoT の PoC（Proof of Concept）実装を行い，データ検索における基本性能を評価した．

2. 関連研究

2.1 IoT におけるオントロジの利用

IoT システムでは，気温，風向等様々なセンサを利用している．そこで文献 [1–3] では IoT システムにおけるセンサの統一されたオントロジ表現を定義している．このオントロジにより，IoT システムにおけるセンサの属性やセンサ間の関係を定義し，ユーザは統一的なセンサの検索を実現している．特に文献 [1] では位置情報も含んだオントロジを定義することで，位置情報によるセンサの検索を可能にしている．

2.2 oneM2M

oneM2M [3] は M2M（Machine to Machine）システムや IoT システムの標準化団体であり，IoT システム間の接続を定義している．**図 1** に oneM2M のアーキテクチャを示す．各 oneM2M サービスプロバイダには 1 台の IN（Infrastructure Node）が存在し，IN 内には CSE（Common Services Entity）が存在する．IN 内の CSE が持つ Mcc’ という I/F で他の oneM2M サービスプロバイダと接続する．oneM2M はコンテキスト情報を厳密に定義しており，

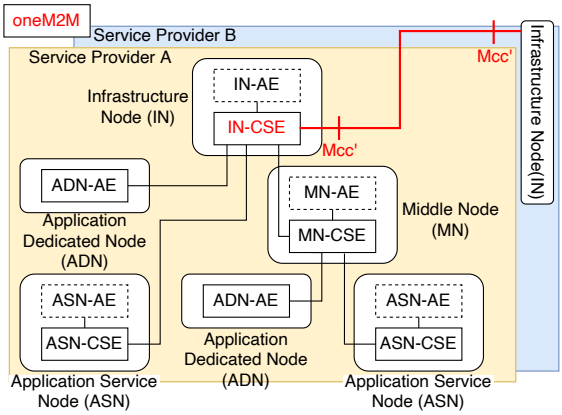

図 1: oneM2M のアーキテクチャ [4]

コンテキスト情報の共有やアクセス制御にも対処している．

しかし CSE は多くの機能を含み，非常に複雑な構成となっている．そのため異なる IoT システム間の I/F は非常に複雑であり，IoT システムとの接続が困難である．またコンテキスト情報によるデータ検索先の IoT システムを決める手法が定義されていない．そのため，全てのコンテキスト情報を IoTSP 間で共有する必要がある場合がある．

2.3 FIWARE

FIWARE [5] は欧州の Future Internet Platform Project の 1 つである．FIWARE は NGSI（Next Generation Service Interface）[6] という標準を拡張したコンテキスト情報モデルにより，IoT システムやその他システムのデータ相互運用に焦点を当てた基盤を構築する．**図 2** に FIWARE のアーキテクチャを示す．FIWARE は IoT システムを構築するためのモジュールを多く作成しており，これらのモジュール内におけるデータの取り扱い方やモジュール間の通信が NGSI により共通化されている．モジュールの中でも，Context Broker がコンテキスト情報を操作及び保存している．異なる IoT システム間で Context Broker が連携することで，IoT システム間連携を実現する．FIWARE は NGSI によりコンテキスト情報の形式と I/F が統一されている．認証認可のモジュールも存在するため，情報へのアクセス制御が可能である．

しかし，NGSI では IoT システム内で扱うコンテキスト情報の用語の意味はアプリケーションごとに異なる．つまり，ある IoT システムで保存されているコンテキスト情報の意味が，他の IoT システムでは別の意味で使用されている場合がある．すなわち IoTSP 間のコンテキスト情報に曖昧性が存在している．また IoT システム間でのコンテキスト情報の共有は想定しておらず，データ検索の度にコンテキスト情報を参照するためネットワークや IoT システムへの負荷が大きい．

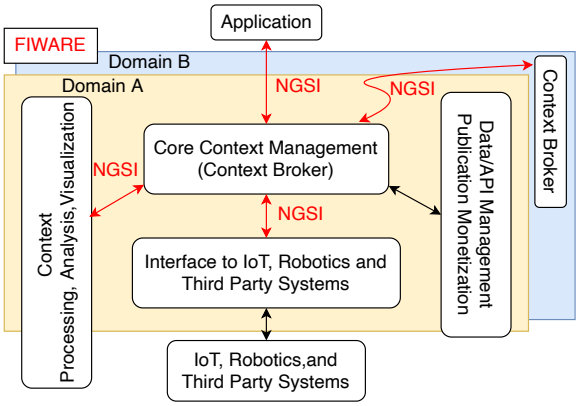

図 2: FIWARE のアーキテクチャ [7]

2.4 IoT システム間連携

文献 [8, 9] では IoT システム間の連携に焦点を当てている．これらは IoT システムが収集したセンサデータを注釈付けし，コンテキスト情報の表現が異なる IoT システムでのデータ利用を可能にしている．この手法では IoT システムをまたぐと，センサデータの表現に曖昧性が生じる．ある IoT システムでのデータ表現を他の IoT システムでは正確に解釈できず，また同じ表現名でも IoT システムが異なると，その名前の意味が異なる場合がある．また IoT システム間の I/F は統一されていないため，IoT システム 間での連携は困難である．さらに IoTSP のような事業者は未考慮であり，IoTSP 間の情報共有手法が未定義である．

FedIoT では全 IoTSP 共通の位置情報に基づくオントロジである OntoFedIoT と IoTSP 間の I/F を簡素化した API を定義し，これらをスキーマ化することで IoTSP 間で曖昧性を排除し連携を容易にする．コンテキスト情報は IoTSP に負荷をかけず，またポリシに配慮した共有をする．

3. FedIoT の提案

FedIoT は自律分散的に IoTSP 間連携を実現する機構である．ユーザはデータを取得するために，IoT におけるデータ検索に必要な要素である地理的領域，データの種類，時間範囲 [10] を契約している IoTSP に指定する．例えば，ユーザは東京都内のある位置（e.g., 大崎駅周辺）のある日時（e.g., 2021 年 1 月 1 日の 10 時からの 5 分間）における，ある種類のデータ（e.g., 気温）を検索したいとする．ユーザは地理的領域として大崎駅を含む矩形領域，データの種類として気温，時間範囲として 2021 年 1 月 1 日 10 時からの 5 分間を IoTSP に指定することで，FedIoT により連携している IoTSP から希望のデータを取得できる．このとき FedIoT 内では，IoTSP 間が連携する上で情報保護に配慮する必要がある．

そこで FedIoT では，全 IoTSP 共通の位置情報に基づくオントロジ OntoFedIoT を定義する．IoTSP 間の I/F は，できるだけ簡素な Primitive API を定義することで接続を簡素化する．OntoFedIoT と Primitive API をスキーマ化することで IoTSP ごとの曖昧性を排除する．また OntoFedIoT で情報共有ポリシを定義し，ポリシを使用した IoTSP 間のコンテキスト情報の伝搬を可能にする．

3.1 FedIoT アーキテクチャ

図 3 に FedIoT のアーキテクチャを示す．IoTSP1 は内部の詳細を示しており，他の IoTSP の内部は省略している．この接続は論理的なものであり，安全な通信路で接続されているとする．図 3 では IoTSP 間は論理的にメッシュ接続しており，センサデータは直接やり取りされる．さらに各 IoTSP は公開鍵暗号系における秘密鍵と公開鍵を所有しているとする．秘密鍵は各 IoTSP が秘匿し，公開鍵は他のすべての IoTSP が共有しているとする．

User は IoTSP と契約を結んでいるユーザである．この例では，User1 は IoTSP1 と契約を結んでおり，ユーザ用のアプリケーションである IoT App1 を利用して IoTSP1 からセンサデータ等を得る．

App API はアプリケーション用 API であり，アプリケーションからの要求を Primitive API に変換し，Query Server にアクセスする．この例では，App API1 は IoT App1 からの要求を受け付けて Primitive API に変換し，Query Server1 にアクセスする．さらに App API1 は Primitive API により IoTSP 間連携を実現する．この例では，App API1 は IoTSP2 内の App API2 と連携し，Primitive API により IoTSP1 と IoTSP2 間の連携を実現する．

Admin は IoTSP の管理者である．この例では，Admin1 は 管理用 API である Admin API を介して IoTSP1 を管理・運用する．

Graph DB は OntoFedIoT のインスタンス情報を保存するデータベースである．この例では，Graph DB1 は IoTSP1 内の Graph DB である．また Admin API を通して Graph DB に OntoFedIoT のインスタンスの追加，変更を実行する．

Sensing DB はセンサの測定値の時系列データを保存するデータベースである．この例では，Sensing DB1 は IoTSP1 内の Sensing DB である．

Query Server は App API から Primitive API を受け付け，Graph DB や Sensing DB にアクセスする．この例では Query Server1 は App API1 からの Primitive API を受け付け，Sensing DB1 や Graph DB1 にアクセスする．

AAA（Authentication, Authorization and Accounting）Server はユーザ，管理者，IoTSP，Thing Provider，Mobile Node の認証認可を行うモジュールである．この例では，AAA Server1 は User1，Admin1，Mobile Node13 の認証認可を行う．図 3 では省略したが，App API2 内の AAA Server2 は IoTSP1 等の認証認可を行う．

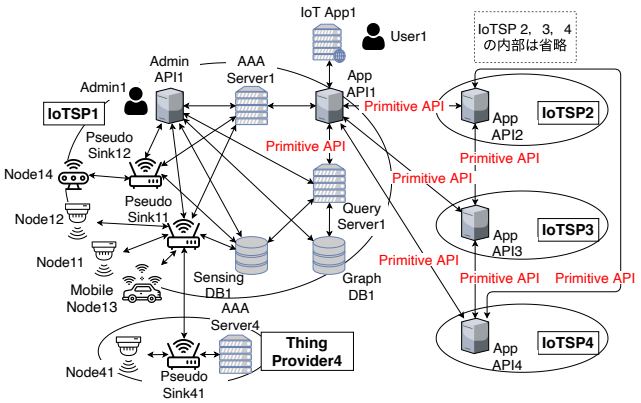

図 3: FedIoT アーキテクチャ

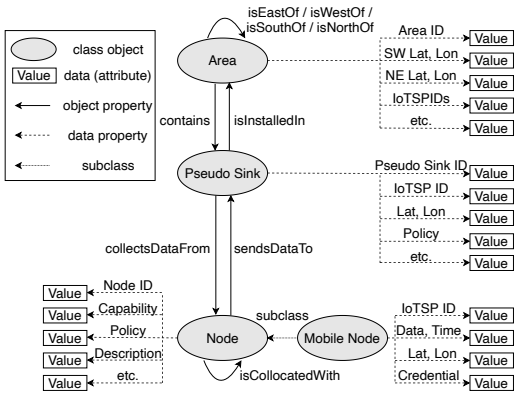

図 4: OntoFedIoT におけるクラス関係

Node や Mobile Node はセンサを搭載した物理的モジュールである．

Pseudo Sink は Node からセンサデータを収集するモジュールである．Node は基本的に無線で Pseudo Sink に接続し，接続手法は範囲外とする．Pseudo Sink は無線基地局がその役割を果たしてもよいし，Pseudo Sink としての専用機器を設置してもよい．この例では Node11 等が Pseudo Sink11 に接続し，さらに Mobile Node13 が Pseudo Sink11 の無線通信範囲に入ったため，一時的に Mobile Node13 は Pseudo Sink11 に接続している．そして Pseudo Sink11 は収集したデータを Sensing DB1 に送信する．

Thing Provider4 では Node41 が Pseudo Sink41 に接続し，Pseudo Sink41 は Pseudo Sink11 に接続している．

3.2 OntoFedIoT: FedIoT におけるオントロジ表現

FedIoT では，センサを位置情報に基づいたオントロジ OntoFedIoT により表現する．オントロジでは物事の関係を主語，述語，目的語の 3 つ組みで表現する．物事の型をクラスと呼び，具体的な物事は該当するクラスのインスタンスとして表現する．クラス間やインスタンス間の関係を表す述語をオブジェクトプロパティ（object property）と呼び，クラスやインスタンスとその属性を表す述語をデータプロパティ（data property）と呼ぶ．

3.2.1 OntoFedIoT のクラス

OntoFedIoT のクラス関係を 図 4 に示す．地理的な領域を緯線経線で囲まれた 0.1 分四方の矩形に分割した領域を `Area` クラスと定義する．図 5 に Area インスタンスの例を示す．緯度経度の 0.1 分は日本付近では約 185m となるため，センサ配置の密度やユーザの探索粒度の面から適当と判断した．Area はデータプロパティとして Area ID，矩形の南西隅（SW）の緯度経度，矩形の北東隅（NE）の緯度経度，Area 内にセンサを設置している IoTSP の ID（IoTSPIDs）等を持つ．Area 同士は東西南北の関係を表すオブジェクトプロパティ（isEastOf 等）で接続している．

センサを有線ネットワークに接続するためのモジュール

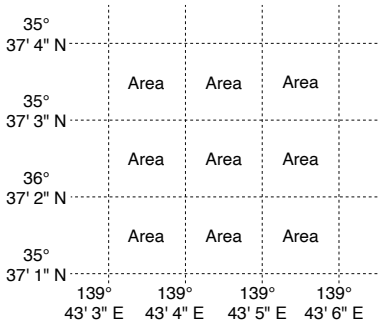

図 5: Area インスタンスの例

を `Pseudo Sink` クラスと定義する．Pseudo Sink はデータプロパティとして Pseudo Sink ID，緯度経度，所有している IoTSP の ID，情報を開示する IoTSP を示す属性である Policy 等を持つ．Policy については 3.2.2 項にて後述する．Pseudo Sink は，自身が設置されている Area とオブジェクトプロパティ（isInstalledIn 等）で接続している．

センサを搭載したモジュールを `Node` クラスと定義する．Node はデータプロパティとして Node ID，センサの種類を示す Capability，Node の注釈を示す Description，Policy 等を持つ．Node は接続する Pseudo Sink とオブジェクトプロパティ（sendsDataTo 等）で接続し，これにより Node の管理や検索を容易にする．

Node クラスのサブクラスとして `Mobile Node` クラスを定義する．Mobile Node は Node が持つデータプロパティに加え，Mobile Node が契約する IoTSP ID，日付と時刻を示す Date, Time，緯度経度，認証情報を示す Credential 等を持つ．

3.2.2 OntoFedIoT のインスタンス

OntoFedIoT のクラスの IoTSP1 におけるインスタンス例を図 6 に示す．地理的領域を分割した Area インスタンスが存在し，Area インスタンス間は東西南北の関係を表す isEastOf 等のオブジェクトプロパティで接続している．

図 6 では Area11 に Pseudo Sink11 が設置されている．Pseudo Sink11 は `IdIoTSP_1` という ID を持つ IoTSP1 が所有する．またこのとき，IoTSP1 は Pseudo Sink11 の情

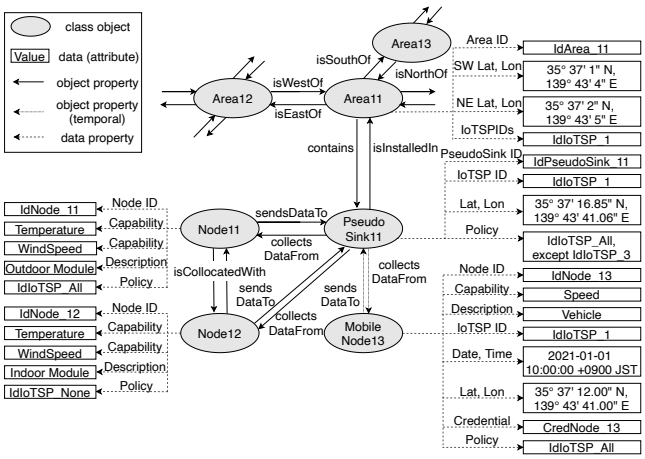

図 6: OntoFedIoT におけるクラスのインスタンス例

報を IdIoTSP_3 という ID を持つ IoTSP 以外への共有を許可するといった Policy を示すことができる．

Pseudo Sink11 に気温データを収集する Node インスタンスである Node11 と Node12 が接続し，Pseudo Sink11 が Node11 と Node12 を管理している．IoTSP1 は Node11 を室外，Node12 を室内に設置した．このとき Policy として，室外に設置された Node11 は全 IoTSP への共有を許可し，室内に設置された Node12 は IoTSP1 以外の IoTSP へ共有しないといった Policy の定義が可能である．

この例では，Mobile Node13 は移動中に Pseudo Sink11 の無線通信圏内に入ったため，一時的に Pseudo Sink11 と接続している．Mobile Node13 の通信時における認証認可には，認証情報である CredNode_13 を使用し，認証が成功した場合にのみデータの送信が可能になる．

以上のインスタンス情報は IoTSP1 内の Graph DB1 に保存される．しかし，Mobile Node インスタンスと Pseudo Sink インスタンスの接続関係は一時的であるので，Graph DB1 には格納しない．

OntoFedIoT により，Area インスタンスを指定することでそこに設置されている Node インスタンスの検索が可能になる．さらに OntoFedIoT 内で定義された Policy により，IoTSP 間で情報の保護が可能になる．

3.3 IoTSP 間でのコンテキスト情報の共有

全ての IoTSP は OntoFedIoT をサポートしていると仮定する．Primitive API を実行するためには，OntoFedIoT 内のコンテキスト情報をある程度 IoTSP 間で共有する必要がある．全くコンテキスト情報を共有しない場合，Primitive API 実行の度に該当する IoTSP にデータ検索するため，遅延やネットワークの負荷が増大する．逆に全コンテキスト情報を共有する場合，Graph DB の負荷が増大し，さらに事業者としてのポリシを考慮するとこれは非現実的である．

そこで IoTSP 間では，各 IoTSP が Pseudo Sink を設置した Area インスタンスの ID（Area ID）を共有する．Area ID を Policy を適用して共有することで，ポリシも考慮した Primitive API が実行可能になる．Area ID の情報共有は IoTSP 間の情報広告により行われる．

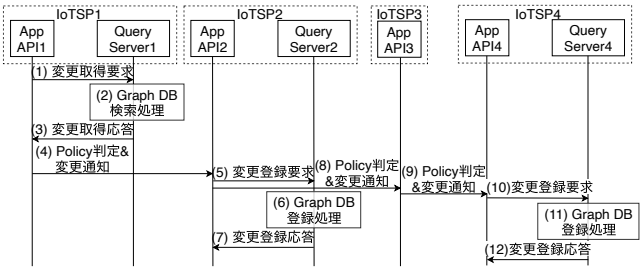

図 7: Policy によるコンテキスト情報共有の例

図 7 に Policy によるコンテキスト情報の共有例を示す．IoTSP 間のコンテキスト情報共有の転送経路はあらかじめ決まっているものとする．前回のコンテキスト情報共有以降，IoTSP1 は Area11 に Pseudo Sink11 を設置したと想定する．この構成変更を IoTSP 間で共有する．

まず，App API1 は IoTSP1 内の構成変更を知るため Query Server1 に変更取得要求を送信する．変更取得要求メッセージには日時を指定し，その日時以降に更新された情報を要求する．Query Server1 は指定された日時以降に変更された構成情報を Graph DB1 に問い合わせ，変更された構成情報を App API1 に送信する．この例では，Pseudo Sink11 が設置された Area11 の ID である IdArea_11 と Pseudo Sink11 の Policy フィールドが変更取得応答として送られる．変更取得応答を受信した App API1 は，あらかじめ決められた転送先である App API2 に変更通知を送信する．変更通知には IdArea_11，IoTSP1 の ID である IdIoTSP_1，Pseudo Sink11 の Policy フィールドが含まれ，IoTSP2 の公開鍵で暗号化されている．すなわち，App API1 は図 6 に示す Pseudo Sink11 の Policy によって公開先，非公開先を判定している．

App API2 は自身の秘密鍵を使用して変更通知を復号し，Query Server2 に IdIoTSP_1 と IdArea_11 を含む変更登録要求を送信する．Query Server2 は Graph DB2 にアクセスし，IdIoTSP_1 を Area11 の IoTSPIDs フィールドに登録する．Query Server2 は変更登録応答を App API2 に送信し，App API2 は App API3 に変更通知を送信する．App API2 は Policy で IoTSP3 には Pseudo Sink11 の情報公開が禁止されていると知り，その次の転送先である IoTSP4 の公開鍵で変更通知を暗号化し送信する．

App API3 は変更通知を受信するが，自身の秘密鍵で復号できないため，次の転送先である IoTSP4 へ送信する．

このような流れでコンテキスト情報の共有が実行される．

3.4 データ取得手順

図 8 に Primitive API の実行例を示す．3.3 節で述べた

ように，IoTSP 間では Pseudo Sink が設置されている Area ID が共有されている．ユーザからデータ取得要求を受信した IoTSP は Primitive API によるデータ検索を実行する．**図 9** にデータ取得要求でユーザが IoTSP に指定する地理的領域の例を示す．またユーザは Capability として気温，時間範囲として 2021 年 1 月 1 日 10 時からの 5 分間を IoTSP に指定しているとする．さらに IoTSP2 が，Area21 に全 IoTSP に公開する Pseudo Sink21 と，全 IoTSP に公開し Capability が気温の Node21 を設置しているとする．

まず，データ取得を要求したユーザに対し AAA Server が認証認可を行う．認証認可による確認が成功すると，IoTSP はユーザが指定した地理的領域内において Pseudo Sink が設置されている Area インスタンスを検索する．これを Pseudo Sink 解決と呼ぶ．この例では，Area11 の ID である `IdArea_11` と Area11 に Pseudo Sink を設置した IoTSP1 の ID である `IdIoTSP_1` の組，Area21 の ID である `IdArea_21` と Area21 に Pseudo Sink を設置した IoTSP2 の ID である `IdIoTSP_2` の組が返る．このときコンテキスト情報の共有により自身の IoTSP 内で検索を解決する．

次に Pseudo Sink 解決で得られた Area インスタンスに Pseudo Sink を設置している IoTSP に対し，該当する Pseudo Sink と接続している Node 内でユーザが指定する Capability を持つ Node インスタンスを検索する．これを Node 解決と呼ぶ．この例では，ユーザは Capability に気温を指定したため IoTSP1 から Node11 の ID である `IdNode_11` と Node12 の ID である `IdNode_12`，IoTSP2 から Node21 の ID である `IdNode_21` が返る．

最後に Node 解決で得られた Node インスタンスを所有する IoTSP に対し，ユーザが指定した日時におけるデータ検索を行う．これを SensorData 解決と呼ぶ．この例では，IoTSP1 から Node11 と Node12 が 2021 年 1 月 1 日 10 時からの 5 分間に収集したセンサデータ，IoTSP2 から Node21 が 2021 年 1 月 1 日 10 時からの 5 分間に収集したセンサデータが返る．

この Primitive API によりユーザは広範囲かつ多種多様なデータ検索が可能で，また必要最低限の API のみを定義することで IoTSP 間の接続が簡素化される．さらにコンテキスト情報を共有することでネットワークの負荷が軽減されている．

3.5 GraphQL によるスキーマ定義

GraphQL [11] は API 向けのクエリ言語であり，データと API におけるクエリ/レスポンスをスキーマ定義による型付けが可能である．そこで FedIoT では，GraphQL により OntoFedIoT と Primitive API のスキーマを定義することでスキーマ化を行う．OntoFedIoT と Primitive API をスキーマ化することで，IoTSP ごとのコンテキスト情報と

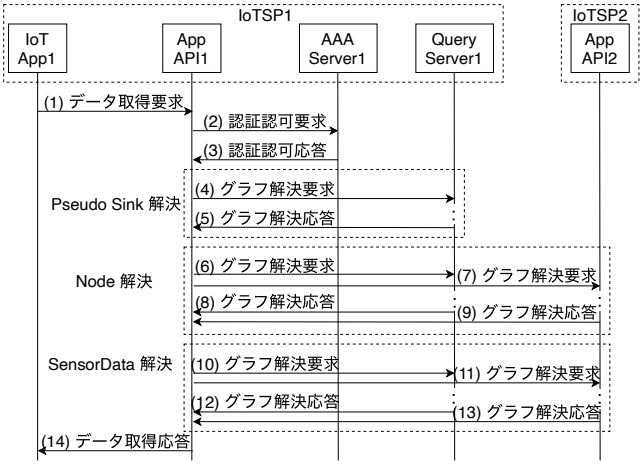

図 8: Primitive API の実行例

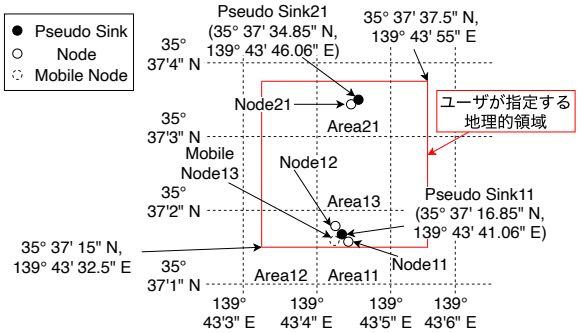

図 9: ユーザの検索範囲の例

```
type Area {
  # data property
  AreaID: ID!
  SW: SquarePoint!
  NE: SquarePoint!
  IoTSPIDs: [ID!]

  # object property
  isEastOf: Area!
  isWestOf: Area!
  isNorthOf: Area!
  isSouthOf: Area!
  contains: [PseudoSink!]
}
```

図 10: Area のスキーマ定義

```
type PseudoSink {
  # data property
  PseudoSinkID: ID!
  IoTSPID: ID!
  LatLon: SquarePoint!
  Policy: [IoTSPID!]

  # object property
  isInstalledIn: Area!
  collectsDataFrom: [Node!]
}
```

図 11: Pseudo Sink のスキーマ定義

```
type Node {
  # data property
  NodeID: ID!
  Capability: [Capability!]!
  Description: String
  Policy: [ID!]

  # object property
  sendsDataTo: PseudoSink!
  isCollocatedWith: [Node!]
}
```

図 12: Node のスキーマ定義

IoTSP 間の I/F に曖昧性がなくなり連携が容易になる．

3.5.1 OntoFedIoT のスキーマ定義

GraphQL による OntoFedIoT のスキーマ定義を**図 10**, **図 11**, **図 12** に示す．GraphQL では，OntoFedIoT におけるクラスのスキーマを “変数名: 型” で定義することができる．例えば Area クラスにおける `AreaID` は `ID` 型である，とスキーマ化することで全ての IoTSP では `ID` 以外の型による `AreaID` の保存は行われない．このスキーマ定義を全 IoTSP がサポートすることで，IoTSP ごとの OntoFedIoT における曖昧性を排除する．

3.5.2 Primitive API のスキーマ定義

GraphQL による Primitive API のスキーマ定義を**図 13**

```
type Query {
  pseudoSinks(SW: PointInput!, NE: PointInput!): [AreaID!]
  nodes(AreaIDs: [AreaID!], Capabilities: [Capability!]): [NodeID!]
  sensorData(NodeID: NodeID!, Capability: Capability!, Period: PeriodInput!):
    [SensorData!]
}
```

図 13: Primitive API のスキーマ定義

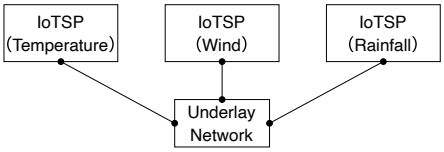

図 14: 実験環境

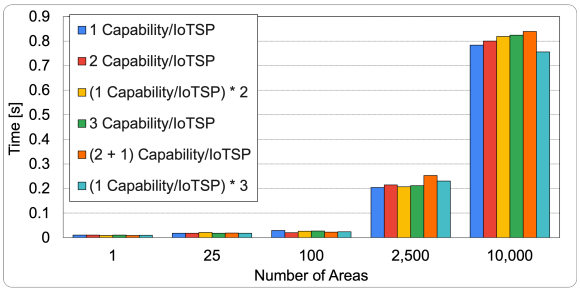

図 15: Pseudo Sink 解決の応答時間

に示す．GraphQL では API のスキーマを “API 名（変数: 引数）: 返り値” で定義することができる．例えば Pseudo Sink 解決において，入力値を地理的領域の矩形における南西隅の緯度経度，北東隅の緯度経度，出力を `AreaID` の配列とスキーマ化することで，全 IoTSP の Pseudo Sink 解決におけるこれら以外の入力や出力を防ぐ．このスキーマ定義を全 IoTSP がサポートすることで，IoTSP 間の Primitive API における曖昧性を排除する．

4. PoC 実装

本稿では FedIoT を PoC 実装した．図 3 に示したモジュールのうち，App API と Query Server を実装した．

App API は Go 言語で IoT App と通信するモジュール，Query Server と通信するモジュール，App API と通信するモジュールを実装した．各通信は HTTP（Hypertext Transfer Protocol）で実行され，Node 解決と SensorData 解決における複数の IoTSP と通信する処理は並行に実行する．

Query Server は GraphQL Server を実装するライブラリである gqlgen[*1]を使用し，Go 言語で実装した．App API と通信するモジュール，Graph DB と Sensing DB のそれぞれへクエリするモジュールを作成した．

5. 評価

5.1 評価環境

図 14 に実験環境を示す．1 台の物理マシン上に VM（Virtual Machine）を 4 台稼働し，3 台は IoTSP，1 台は IoTSP 間を接続する Underlay Network とした．物理マシンの OS は VMware（R）ESXi（TM），CPU は Intel（R）Xeon（R） Gold 6248 2.50GHz×40，RAM は 400GB である．IoTSP の OS は Ubuntu Server 20.04 LTS，仮想 CPU は Intel（R） Xeon（R） Gold 6248 2.50 GHz×4，RAM は 32GB である．Underlay Network の OS と仮想 CPU は IoTSP と同様で，RAM は 4GB である．VM の各 I/F には tc コマンドにより遅延を 2.5ms 加えた．

Graph DB は Neo4j [12]，Sensing DB は SQLite [13] を採用した．Area インスタンスは東京都の島を除いた領域として 232,932 個を保存した．センサのデータセットは，気象業務支援センター [14] が提供する「地上アメダス 1 分値・10 秒値データ」を使用した．このデータは日本国内に設置されている気象台とアメダスが収集した気象データを含む．気象データの中で気温，風，雨に関するデータをそれぞれ別々の IoTSP が収集するとした．気象台等は，島を除く東京都内の適当な位置に存在する Pseudo Sink インスタンスと Node インスタンスとし，それぞれ 1,441 個のインスタンスを各 IoTSP に保存した．各 IoTSP の Node インスタンスは，3 から 6 種類の Capability を持つ．センサデータは，各 Node インスタンスに該当する気象台等が 1 分ごとに収集した半年間分のデータを保存した．データ取得要求は IoTSP 内の VM から curl コマンドにより送信する．

[*1] https://github.com/99designs/gqlgen

5.2 評価結果

IoTSP に対してユーザが指定する地理的領域と Capability を変更したときの Primitive API の応答時間を評価する．日時は適当な日付の 5 分間を指定した．地理的領域は正方形の領域となるように指定した．Capability は最大 3 種類指定し，それぞれのデータが保存されている IoTSP のパターンを変更しながら実行した．例えば 1 Capability/IoTSP は気温のみを指定し，（1 Capability/IoTSP）*3 は気温, 風向, 降水量を指定する．（2+1）Capability/IoTSP では気温，前 1 分間の最高気温，風向を指定する．

図 15 に Pseudo Sink 解決の応答時間を示す．地理的領域が大きくなると応答時間は増加している．また Pseudo Sink 解決では入力値の地理的領域のみに影響されるため，指定した地理的領域が同じときの Capability 数の違いは応答時間に影響していない．約 1.8km × 1.8km の大きさである 100 個の Area を指定したとき，探索時間は約 30ms となった．約 18km × 18km の大きさである 10,000 個の Area を指定したとき，探索時間は約 800ms となった．

図 16 に Node 解決の応答時間を示す．Node 解決は全ての場合で応答時間が 40ms に収まった．Area 数が 1 個

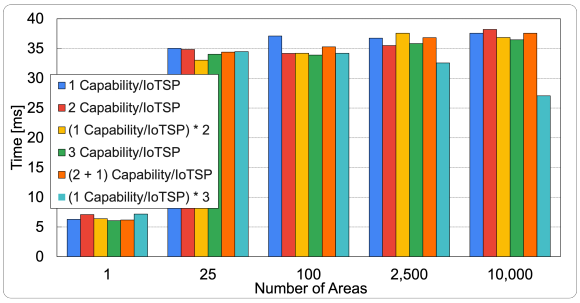

図 16: Node 解決の応答時間

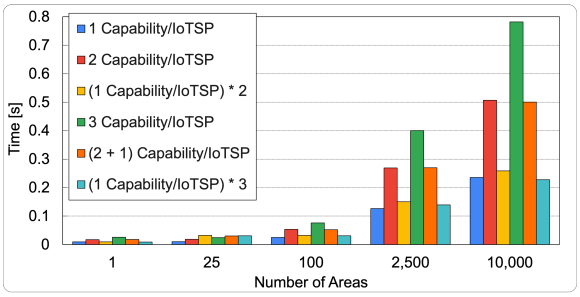

図 17: SensorData 解決にかかる探索時間

から 25 個になると応答時間が急に増加してるのは，他の IoTSP が設置した Pseudo Sink が検索範囲内に存在し，他の IoTSP との通信が発生しているためだと考えられる．Area 数が 25 個から増加したとき，検索範囲内に存在する Pseudo Sink は増加しているが，応答時間はほとんど変化していない．また Area 数が 100 個以上のとき，Capability が（1 Capability/IoTSP）*3 のときの応答時間が非常に小さくなっている．これは複数の IoTSP へ並行にリクエストし，処理が分散したためだと考えられる．

図 17 に SensorData 解決の応答時間を示す．指定した Capability が同じとき，Area 数が増加するとセンサデータを取得する Node の数が増加するため応答時間は増加している．3 Capability/IoTSP のときの応答時間が最も大きくなっている．これは 1 つの IoTSP のみにグラフ解決要求が集中するためだと考えられる．逆にグラフ解決要求が分散する（1 Capability/IoTSP）*3 では，応答時間が 1 Capability/IoTSP と変わらない．Area 数が 100 個のとき，応答時間は全ての場合において 100ms を下回っている．Area 数が 10,000 個で指定した Capability が（1 Capability/IoTSP）*3 のとき，センサデータを取得する Node の数は合計 73 個であったが，応答時間は約 250ms に収まった．

6. おわりに

従来の IoT システムは目的が明確かつ closed に構築されている．IoT システムを利用するユーザは広範囲に設置したセンサから多種多様なデータを得たいという需要がある．

本稿では，IoT システムで得たデータをユーザに提供する IoTSP が自律分散的に連携する機構である FedIoT を提案した．IoT システムはそれぞれで異なった表現により情報を保存しており，IoT システム間のデータ検索は困難である．また IoTSP 間の I/F が複雑だと連携は容易ではない．さらに IoTSP 間で連携するためには，お互いの情報を保護しながら情報共有をする必要がある．

そこで FedIoT はオントロジ OntoFedIoT と Primitive API を定義し，これらをスキーマ化することで IoTSP 間の連携を容易にした．またオントロジにポリシを付加することで，プライバシに配慮した情報共有を可能にした．本稿は FedIoT を PoC 実装し，基本性能を評価した．約 1.8km × 1.8km を探索する 3 種類の Primitive API の応答時間は，それぞれ 100ms 以下になることを確認した．

参考文献

[1] Nambi, A., Sarkar, C., Prasad, R. V. and Rahim, A.: A Unified Semantic Knowledge Base for IoT, *In Proc. of 2014 IEEE WF-IoT*, pp. 575–580 (2014).
[2] Xie, C., Yu, B., Zeng, Z., Yang, Y. and Liu, Q.: Multilayer Internet-of-Things Middleware Based on Knowledge Graph, *IEEE Internet of Things Journal*, Vol. 8, No. 4, pp. 2635–2648 (2021).
[3] Swetina, J., Lu, G., Jacobs, P., Ennesser, F. and Song, J.: Toward a standardized common M2M service layer platform: Introduction to oneM2M, *IEEE Wireless Communications*, Vol. 21, No. 3, pp. 20–26 (2014).
[4] oneM2M: oneM2M Technical Specification: Functional Architecture, TS-0001-V4.11.0 (2021).
[5] Castrucci, M., Cecchi, M., Priscoli, F. D., Fogliati, L., Garino, P. and Suraci, V.: Key Concepts for the Future Internet Architecture, *In Proc. of 2011 Future Network Mobile Summit*, pp. 1–10 (2011).
[6] Fonseca, J. M. C., Márquez, F. G. and Jacobs, T.: FIWARE-NGSI v2 Specification, `http://fiware.github.io/specifications/ngsiv2/stable/`.
[7] FIWARE Foundation: FIWARE/catalogue, `https://github.com/FIWARE/catalogue`.
[8] An, J., Gall, F. L., Kim, J., Yun, J., Hwang, J., Bauer, M., Zhao, M. and Song, J.: Toward Global IoT-Enabled Smart Cities Interworking Using Adaptive Semantic Adapter, *IEEE Internet of Things Journal*, Vol. 6, No. 3, pp. 5753–5765 (2019).
[9] Agarwal, R., Fernandez, D. G., Elsaleh, T., Gyrard, A., Lanza, J., Sanchez, L., Georgantas, N. and Issarny, V.: Unified IoT Ontology to Enable Interoperability and Federation of Testbeds, *In Proc. of 2016 IEEE WF-IoT*, pp. 70–75 (2016).
[10] Barnaghi, P. and Sheth, A.: On Searching the Internet of Things: Requirements and Challenges, *IEEE Intelligent Systems*, Vol. 31, No. 6, pp. 71–75 (2016).
[11] GraphQL contributors: GraphQL specification (draft), `https://facebook.github.io/graphql/draft/`.
[12] Neo4j, Inc.: Neo4j: Graph Database Platform, `https://neo4j.com/`.
[13] SQLite Consortium: SQLite Home Page, `https://www.sqlite.org/index.html`.
[14] 一般財団法人　気象業務支援センター：気象業務支援センター，`http://www.jmbsc.or.jp/jp/`.