異種のIoTデバイスとWebサービスの相互運用のためのIoTサービス基盤*

大川　楠人[†a)]　林　冬惠[†b)]　村上　陽平[††c)]　中口　孝雄[†††d)]

IoT Service Platform for Interoperability among Heterogeneous IoT Devices and Web Services*

Nanto OKAWA[†a)], Donghui LIN[†b)], Yohei MURAKAMI[††c)], and Takao NAKAGUCHI[†††d)]

あらまし　Internet of Things（IoT）の発展によって，物理世界の多種多様なデバイスとインターネット上のWebサービスを接続し，高度なIoTアプリケーションを構築することが可能になっている．しかし，IoTデバイスやWebサービスの仕様が異なる場合，相互運用ができないことが多いため，開発者は膨大なバリエーションに対応する必要がある．本研究では，IoTアプリケーションを容易に構築するため，異種のIoTデバイスとWebサービスを相互運用可能なIoTサービス基盤を実現した．具体的には，IoTサービスの機能を軸にIoTデバイスとWebサービスを標準化し，イベント処理に基づく基盤を提案し，IoTアプリケーションにおける能動型及び受動型のイベント処理を両方実現した．また，提案した基盤の応用例として，IoT環境における対話エージェントを実装した．更に，IoTアプリケーション開発者のコスト削減に関する分析及び対話エージェントアプリケーションの応答速度に関する評価を行うことで，提案したIoTサービス基盤の有効性と実用可能性を示した．

キーワード　Internet of Things，相互運用性，Webサービス，IoTデバイス，IoTサービス基盤

1. まえがき

近年，Internet of Things（IoT）の発展によって，膨大なIoTデバイスによって生まれる大規模なデータを人工知能の技術を用いて処理し分析するというニーズが高まっている．スマートホームやヘルスケアなどの領域において様々な物理的なデバイスがインターネットに繋がり，クラウド上にあるWebサービスや各種アプリケーションと接続することで，高度なIoTアプリケーションを実現できるようになってきている[1]．

しかしながら，IoTアプリケーションの実装方法や通信プロトコル，各種IoTデバイスの規格はメーカーや領域ごとに異なっているため，異種のIoTデバイスの相互運用が困難な状況にある[2]．例えば，A社の温度センサーは5秒ごとに温度データを取得し，そのストリーミングデータをA社独自のアプリによってCSVファイルに格納するが，B社の温度センサーは30秒ごとにデータを取得し，JSON形式でデータを送信する仕組みを利用している．更に，利用者のリクエストによって温度データを取得する従来の天気予報Webサービスもインターネット上に存在している．本研究では，IoTデバイスであるセンサーやアクチュエータ，更にWeb APIなどのWebサービスを総称して**IoTサービス**と呼ぶ．これらのIoTサービスを利用するために，IoTアプリケーション開発者は独立した3種類のシステムを実装する必要がある．また，一つのIoTアプリケーションに複数種類のIoTサービスを利用する場合，これらのサービスの組み合わせによる実装のバリエーションが爆発的に増加する．そこで本

† 京都大学大学院情報学研究科社会情報学専攻，京都市
Department of Social Informaics, Kyoto University, Yoshida-honmachi, Sakyo-ku, Kyoto-shi, 606–8501 Japan
†† 立命館大学情報理工学部，草津市
College of Information Science and Engineering, Ritsumeikan University, 1–1–1 Noji-higashi, Kusatsu-shi, Shiga, 525–8577 Japan
††† 京都情報大学院大学，京都市
The Kyoto College of Graduate Studies for Informatics, 7 Tanakamonzen-cho, Sakyo-ku, Kyoto-shi, 606–8225 Japan
a) E-mail: nanto.070503@gmail.com
b) E-mail: lindh@i.kyoto-u.ac.jp
c) E-mail: yohei@fc.ritsumei.ac.jp
d) E-mail: ta_nakaguchi@kcg.ac.jp
* 本論文は，システム開発論文である．
DOI:10.14923/transinfj.2021SKP0023

研究は，IoT アプリケーションを容易に構築するため，インターネット上の Web サービスと物理世界の各種デバイスが混在する IoT 環境において，相互運用性を保証する IoT サービス基盤を実現することを目的とする．具体的には，以下の二つの課題に取り組む．

IoT サービスの相互運用の実現：IoT デバイスと Web サービス間の異質性に対応する必要がある．そのため，IoT サービスの機能を軸に，Web サービスと IoT デバイスを横断する IoT サービスインタフェースを標準化する．従来，IoT デバイスを用いるアプリケーションでは，センサーデータによるアクチュエータの駆動を実現する必要があるため，イベントベースの手法が多く利用されている [3], [4]．本研究では，Web サービスと IoT デバイスを一元的に管理するため，イベントベースの手法のためのサービスインタフェースを実現し，イベント処理に基づくアーキテクチャを提案する．IoT サービスは，イベントの発生源（センサーやデータの収集を目的とする Web サービス）であり，イベント処理後に駆動される機能（アクチュエータや機能の駆動を目的とする Web サービス）でもある．イベントは単体の IoT サービスによる原子イベント及び複数の原子イベントからなる複合イベント [5] を含む．

受動型処理と能動型処理の実現：IoT サービス基盤が相互運用性を保証する上で，IoT アプリケーションからサービスを呼び出す受動型イベント処理だけではなく，一定の条件において自動的に発火する能動型イベント処理も対応する必要がある．具体的には，センサーなど複数の IoT サービスから収集されるデータに基づきイベントを生成し，複合イベントの検出と処理を行い，自動的に IoT アプリケーションに情報を送信する，またはアクチュエータなどの IoT サービスを駆動する．本研究では，IoT サービスからデータを収集する機構と，IoT サービスを駆動する機構，ルールに基づいて複合イベントを処理する機構を別々に設計することで，受動型のイベント処理と能動型のイベント処理を実現する．

本研究で提案する IoT サービス基盤の応用例として，IoT 環境における対話エージェントを実際に実装する．また，IoT アプリケーション開発のコスト削減に関する分析を行うことで基盤の有効性を示す．更に，対話エージェントアプリケーションの応答性能を評価することで，基盤の実用可能性を検証する．

2. 関 連 研 究

従来のサービスコンピューティング分野では，Web サービスの相互運用性が最も重要な課題の一つである．Murakami らは，Web サービスの相互運用性を高めるため，サービスインタフェースの標準化に基づくサービス基盤を提案し，サービスグリッドソフトウェアとして実装している [6]．Ishida らは，サービスグリッドソフトウェアを言語サービスのドメインで展開し，「言語グリッド」といった言語サービス基盤を開発している [7]．本研究で提案する IoT サービス基盤の設計思想はサービスグリッドと共通する部分があるが，Web サービスと IoT デバイスが混在する IoT 環境における相互運用性問題を扱う点で大きく異なる．また，従来の Web サービス基盤では，利用者が Web サービスを呼び出すという受動型サービスに対応すれば良いが，IoT 環境では能動型サービスの実現を考慮する必要がある．一方，我々は，これまでに Web サービス基盤と IoT 基盤の統合に関する初期的なコンセプトを提案しているが，システムの詳細設計と実装にまで至っていない [8]．本研究は IoT サービス基盤だけではなく，基盤を用いた IoT アプリケーションの実装を通して，提案する基盤の有効性を示す．

また，IoT の分野においても，相互運用性問題が最も重要な問題として挙げられている [2]．一般的に，IoT の相互運用性に対応するためには，異種の IoT デバイスの機能を標準に基づいて統一的な記述方法で記述する必要がある．W3C や ISO などの標準化組織が 10 年以上に渡り，IoT の相互運用性を向上するための標準を策定している．特に，W3C は 2020 年 4 月に Web of Things (WoT) Thing Description (TD) 及び Web of Things (WoT) Architecture を正式に W3C 勧告として発表している(注1)．しかし，WoT は特定の IoT デバイスからのデータを，メタデータにマッピングするためのドメイン固有の語彙を提供していない．Novo と Francesco は，物流という具体的な IoT 利用シナリオを与え，WoT を拡張したアーキテクチャを提案している [9]．本研究は，IoT サービス基盤の構築が主な目的であり，標準インタフェースの定義を WoT 準拠とすることを前提としない．一方，特定の応用領域における Web サービスと IoT デバイスの標準インタフェースを定義する際に，W3C などの標準化組織や外部の

（注1）：https://www.w3.org/WoT/

IoT オントロジーが提供する定義を活用することが可能である．

3. IoT サービス基盤の実装と運用

3.1 設計思想

IoT サービス基盤は IoT アプリケーションと IoT サービス（センサー，アクチュエータ，Web サービス）を繋ぐものである．IoT デバイスと Web サービス間の異質性を対応するため，IoT サービスの機能を軸に，Web サービスと IoT デバイスを横断する IoT サービスインタフェースを標準化する必要がある．従来，IoT アプリケーションでは，センサーからデータを収集しアクチュエータを駆動することを想定するため，イベントベースの手法 [3], [4] が多く利用されている．本研究では，IoT デバイスと Web サービスの相互運用のため，Web サービスをデータを収集するタイプと機能を駆動するタイプに分けて，それぞれセンサーとアクチュエータと同様なインタフェースに従って IoT サービス基盤に登録することで，センサーとアクチュエータのようにイベントベースの手法で管理することができる．例えば，**1.** で示した例では，天気予報サービス C が Web サービスであるため，従来の Web サービス基盤においては，利用者のリクエストを受けてサービスの実行を行い，利用者に実行結果をレスポンスする．一方，本研究では，天気予報サービス C が温度センサー A と温度センサー B と同様な機能をもつため，温度センサー A，B と同じインタフェースに従って，定期的なリクエストを自動的に生成しレスポンス結果をデータベースに蓄積するという Web サービスに対してポーリングする方式で実装され，IoT サービス基盤に登録される．IoT アプリケーション開発者から見ると，同じ機能をもつ温度サービスインタフェースを利用することで，IoT サービス（A，B，C）を識別できる ID を切り替えるだけで同じプログラムで IoT アプリケーションを実装可能である．

単体の IoT サービスによるイベントは原子イベントと呼ぶ．また，複数の原子イベントからなるイベントは複合イベントと呼ぶ．IoT アプリケーション開発者が IoT サービス基盤上に登録されている IoT サービスによる原子イベントまたはルールの記述による複合イベントを利用することで，IoT サービスの元の仕様やデータフォーマットを意識せずに，IoT アプリケーションを容易に構築可能である．図 1 に IoT サービス基盤の設計思想を示す．

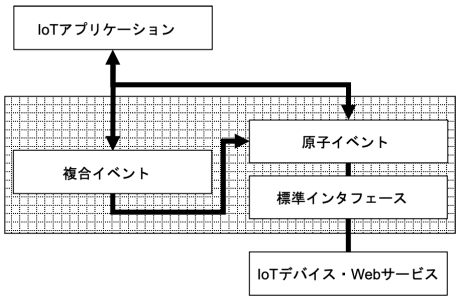

図 1　IoT サービス基盤（網掛け部分）の設計思想
Fig. 1　Design concept of the IoT service platform (shaded part).

IoT サービスの相互運用を実現する上で，IoT サービス基盤のもう一つ重要な要件として，受動型イベント処理だけではなく，一定の条件において自動的に発火する能動型イベント処理も必要である．図 1 で示すように，IoT アプリケーションからイベントを呼び出すことも，イベントから IoT アプリケーションへデータを送信することも可能である．そのためには，IoT サービス（Web サービスやセンサー）からデータを収集する機構と，IoT サービス（アクチュエータや Web サービス）を駆動する機構，及びルールによる複合イベントを処理する機構を別々に設計する必要がある．

3.2 アーキテクチャ

図 2 に IoT サービス基盤のアーキテクチャを示す．センサー，アクチュエータ，Web サービスは IoT サービスである．また，Web サービスは機能の種類によって，センサーのようにデータを提供する場合もあればアクチュエータのように駆動が必要な場合もある．アダプターは，異なる仕様をもつ IoT サービスそれぞれに対して標準インタフェースに従って実装される．デバイス・サービスマネジャーは，IoT サービスの登録管理を行う．コレクターは，Web サービスやセンサーごとのデータフォーマットや仕様の違いをアダプターで吸収してから，標準インタフェースを用いてデータを取得し他のモジュールに渡す．パブリッシャーは，データを IoT アプリケーションへ配信する．インボーカーは，駆動リクエストをアクチュエータや Web サービスに送信する．CEP（Complex Event Processing，複合イベント処理）プロセッサは，イベントデータベースを用いて，ルールの実行，複合イベントの検出，及びアクチュエータの駆動や Web サービスの実行を行う．イベントマネジャーは，原子イベントと複合イベントの配備管理を行う．イベントマネジャーの設計に

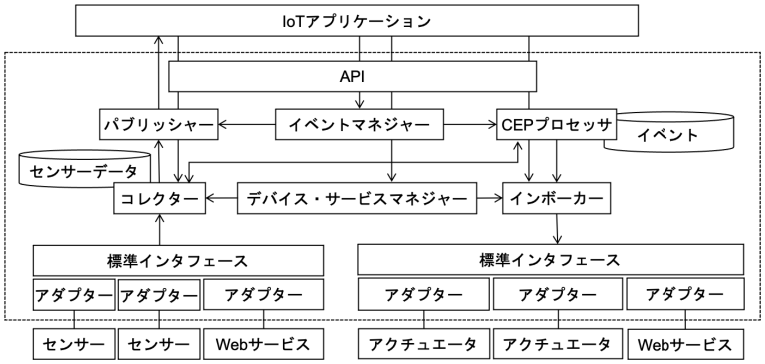

図 2　IoT サービス基盤のアーキテクチャ
Fig. 2　Architecture of the IoT service platform.

ついて，IoT アプリケーションの実装例を用いて **4.2** で詳細に説明する．また，マネジャー，コレクター，インボーカーの API 機能が提供される．このアーキテクチャに基づいて，我々は IoT サービス基盤ソフトウェアを実装した(注2)．

IoT サービス基盤ソフトウェアは，IoT サービスの管理機構，IoT サービスと外部とのインタラクションを制御する機構，イベント処理の管理・実行機構を提供するが，特定の IoT アプリケーションの応用領域に依存しない．IoT サービス基盤を運用するには，以下の 4 種類のステークホルダーがいる．**IoT サービス基盤運営者**は，実世界の特定の応用領域のニーズに応じて，IoT サービスの標準インタフェースを実装することで，様々な応用領域の IoT サービス基盤を構築し公開する．**IoT サービス提供者**は，IoT サービス基盤運営者が提供する基盤における IoT サービスの標準インタフェースに従って，様々な IoT サービスのアダプターを実装し，IoT サービスの登録と，原子イベントの定義，複合イベントの記述を行う．**IoT アプリケーション開発者**は，IoT サービス基盤上の IoT サービスを用いて，一般の **IoT アプリケーション利用者**向けに様々な IoT アプリケーションを開発する．実際に，運用上の都合で，一つの組織が複数のステークホルダーの役割を担う場合がある．

4. 対話エージェントへの応用

本章では，対話エージェントの実現を具体例として，**3.2** で提案した IoT サービス基盤を特定の応用領域においてどのように構築するかを説明する．また，IoT サービス基盤に関わる技術的な部分を中心に説明するため，対話理解や対応生成など既存のエージェントモデルを利用する実装部分の説明を省略する．以降，標準インタフェースの実装及び対話エージェントからの IoT サービスの利用，受動型と能動型の複合イベント処理の実現を具体的に説明する．

4.1 標準インタフェースと IoT サービスの実装

特定の応用領域の IoT サービス基盤を IoT アプリケーション開発者に提供するため，まずその領域のための標準インタフェースの実装が必要である．今回実装する対話エージェントアプリケーションの目的として，利用者の生活環境における IoT サービス（環境センサーや環境情報を取得する Web サービス，スマートホーム環境のアクチュエータ，スマートフォンから利用者の情報を取得するセンサーなど）を用いて，状況に応じた対話を実現することである．スマートホーム環境を中心に標準インタフェースの設計や IoT サービスの実装を行う．本実装では，IoT サービス基盤上の標準インタフェースを階層的に設計する．

まず，データを取得する IoT サービスの標準インタフェースとして，上位抽象インタフェースである `Sensor` を設計する．次に，スマートホームにおける対話エージェントを想定し，`Sensor` インタフェースを継承する形で，温度や湿度，騒音などを扱う `TemperatureSensor`，`HumiditySensor`，`NoiseSensor` を実装している．更に，温度や湿度，騒音などの標準インタフェースに従って，データフォーマットや仕様が異なる IoT サービスのアダプターをそれぞれ実装する．例えば，`TemperatureSensor` インタフェースの具体的な IoT サービス用のアダプターとして，Web サービスである Open-

(注2)：https://github.com/nantooo/iostbase

WeatherMap(注3)から変換された WeatherMapSensor や，CSV や JSON などの異なるデータフォーマットを対応する CSVTemperatureSensorService，JSONTemperatureSensorService が実装されている．TemperatureSensor には，倍精度浮動小数点型の戻り値をもつ getTemperature メソッドのテンプレートが実装され，CSVTemperatureSensorService と WeatherMapSensor には，それぞれ CSV ファイルから温度値を取得する getTemperature の実装と Web サービスから温度値を取得する getTemperature の実装を行う．IoT サービス基盤上のコレクターは，温度を取得する IoT サービスごとのデータフォーマットの違いをアダプターで吸収してから，TemperatureSensor 標準インタフェースを用いて，倍精度浮動小数点型に統一された温度値を取得する．

同様に，機能を駆動する IoT サービスの標準インタフェースとして，上位抽象インタフェースである Actuator を実装し，また，スマートホームにおける対話エージェントからの利用を想定し，Actuator 抽象インタフェースを継承する形で，電灯やエアコン，ドア，音量などの操作を扱う ToggleSwitchActuator，VolumeSwitchActuator，OnOffSwitchActuator を実装し，それらの標準インタフェースに従って IoT サービスのアダプターを実装している．例えば，OnOffSwitchActuator 標準インタフェースに従って，HumidifierOnOffSwitchActuatorService や LightOnOffSwitchActuatorService などが実装されている．

このように，スマートホーム環境に関わる標準インタフェースを設計し，合計 20 種類以上の具体的な IoT サービスを実装し，相互運用可能な形で IoT サービス基盤に登録することで，対話エージェントアプリケーションから容易に利用するために提供する．

4.2 対話エージェントからの IoT サービスの利用

本研究で実装した対話エージェントは主に図 2 に示すイベントマネジャーの部分を通じて基盤上の IoT サービスを利用する．対話エージェントから見ると，**4.1** で実装したデータを収集する IoT サービスはイベントの発生源である．また，機能を駆動する IoT サービスはイベント処理によって実行される．そこで，原子イベントの通知と取得（notifyEvent，getEvents など），複合イベントのルールの管理（updateRule, activateRule, deactivateRule）といった機能をもつイベントマネジャー（表 1）を設計する．

(注3)：https://openweathermap.org/

```
# CSVTemperatureSensorServiceのサービス情報
"type": "service.iot.CSVTemperatureSensorService",
"deviceId": "KyotoWeatherCSV",
"latLng": {"latitude": 45.0116, "longitude": 145.7681},
"placeTag": "myhome",
"dataType": "temperature",
"file": "sensorrunner/kyoto_weather_20210101.csv",
"headerRows": 4,
"valueColumn": 4,

# WeatherMapSensorのサービス情報
"type": "service.iot.WeatherMapSensor",
"deviceId": "OpenWeatherMap",
"latLng": {"latitude": 35.0116, "longitude": 135.7681},
"placeTag": "outside",
"dataType": "temperature",
"host": "community-open-weather-map.p.rapidapi.com",
"key": "SERVICE_KEY",
```

図 3　IoT サービス基盤上のサービス登録情報の例
Fig. 3　Examples of service profiles in the IoT service platform.

データを収集する IoT サービスは，イベントマネジャーの notifyEvent メソッドを用いて，データを更新する際にイベントとしてデータベースに記録する．例えば，図 3 に TemperatureSensor を継承する 2 種類の IoT サービスの登録情報を示す．CSVTemperatureSensorService のサービス情報には，アダプターの種類（type），イベントの構成要素（deviceId，latLng，placeTag，dataType）が含まれている．また，file（CSV ファイルの場所），headerRows（温度値を格納する開始行），valueColumn（温度値を格納する列）の情報をパラメーターの値として，CSVTemperatureSensorService の getTemperature メソッドを用いて温度値を取得し，イベントの要素である value に値を渡し，notifyEvent メソッドを用いてイベントデータベースに記録する．同様に，WeatherMapSensor のサービス情報の場合，host（Web サービスのエンドポイント）と key（Web サービスにアクセスための認証キー）の情報をパラメーターの値として，WeatherMapSensor の getTemperature メソッドを用いて温度値を取得しイベントデータベースに記録する．また，機能を駆動する IoT サービス（例えば，電灯操作サービス）は，インボーカーが定期的に getEvents を用いてイベントデータベースからイベントを取得しその結果をもとにサービスを駆動する．

本研究では，実際に getEvents メソッドのバリエー

表 1　IoT サービス基盤上のイベントマネジャーの設計
Table 1　Design of the event manager in the IoT sevice platform.

メソッド定義	引数と動作の説明
void notifyEvent(Event event)	**引数 (Event event)**: 通知するイベント イベントは deviceId（サービス ID），dataType（データ種類），placeTag（場所タグ），latLng[latitude,longitude]（緯度・経度），value（値），created（発生時間）から構成される． **動作**: イベントを通知する．渡されたイベントはイベントデータベースに記録される．
Event[] getEvents(Date lastEventTime,long timeoutMillis)	**引数 (Date lastEventTime)**: イベントを取得する際に基準となる時間 **引数 (long timeoutMillis)**: タイムアウト **動作**: イベントデータベースに記録されたイベントを取得する．lastEventTime に指定された時間より後のイベントが返される．返すイベントがない場合，最長 timeoutMillis に指定された時間待機する．
Event[] getEventsOfDevice(String deviceId, Date lastEventTime,long timeoutMillis)	**引数 (String deviceId)**: イベントを取得する IoT サービスの ID **引数 (Date lastEventTime)**: イベントを取得する際に基準となる時間 **引数 (long timeoutMillis)**: タイムアウト **動作**: イベントデータベースに記録されたイベントを取得する．deviceId に指定された IoT サービスに対して，lastEventTime に指定された時間より後のイベントが返される．返すイベントがない場合，最長 timeoutMillis に指定された時間待機する．
Event[] listEvents(int page, int size)	**引数 (int page)**: 取得するページ **引数 (int size)**: 1 ページに含まれる件数 **動作**: イベントデータベースに記録されたイベントのリストを取得する．取得するイベントのリストは，page 及び size 引数で指定する．
void updateRule(String ruleId, String body)	**引数 (String ruleId)**: 更新するルールの ID **引数 (String body)**: 更新するルールの内容 **動作**: ルールを更新する．ルールは Drools の文法に従って記述する．
void activateRule(String ruleId)	**引数 (String ruleId)**: ルール ID **動作**: ルールを有効化する．notifyEvent によりイベントが追加されると，有効化されたルールにイベントが挿入される．
void deactivateRule(String ruleId)	**引数 (String ruleId)**: ルール ID **動作**: ルールを無効化する．notifyEvent により追加されるイベントは，無効化されたルールには挿入されない．

ションとして，表 1 に示す `getEventsOfDevice` 以外に，`getLatestEventsByPlaceTagAndDataType` や `getLatestEventsByLatLngAndDataType` などイベントの各要素の組み合わせによりイベントを取得するメソッドも複数実装されている．また，`listEvents` を用いて，イベントデータベースに記録されるイベントのリストを取得できる．このように，対話エージェントを実装する際に，イベントマネジャーを利用することで IoT サービスの利用と操作を行うことができる．

4.3　受動型と能動型の複合イベント処理の実現

4.1 で実装した IoT サービスに基づき，対話エージェントのための複合イベントを実装する．複合イベントは，イベントマネジャーにより管理され，CEP プロセッサにより処理・実行される．また，複合イベントを実行するため，事前に原子 IoT サービスの実行に対応する複数の原子イベントを用いてルールを記述する必要がある．本研究の実装では，ルールの構成として Drools(注4) を利用し，実際に受動型と能動型を含む 4 種類の複合イベントを実装している．

図 4 に，利用者の情報から取得した位置情報（`placeTag`）に基づき寒暖差を取得する受動型複合イベントのルールを示す．具体的には，利用者から対話エージェントに寒暖差の情報のリクエスト（`req`）があると，利用者のいる場所の温度情報（`temp1`）と利用者のいる場所の外の温度情報（`temp2`）を取得し，寒暖差をイベントデータベースに追加する．対話エージェントはこのイベントを受け取り，利用者にイベントの内容を応答する．

図 5 に，利用者が出かける際に天気予報に基づき傘をもつよう提示する能動型複合イベントのルールを示す．具体的には，利用者のカレンダーに記録されている予定の内容，開始/終了時刻，場所の情報を対話エージェントが定期的に取得し，予定の場所と自宅場所をもとに経路探索 API で想定時間を算出し，想定時間 ±15 分程度の範囲のときに，傘提案ルールをアクティベートする．人感センサーやドアセンサーのデータに基づき利用者が出かけるという状況を検出し，天気予

(注4)：https://www.drools.org/

```
rule "寒暖差"
when
  $time: TimerEvent()
  $req:Event(deviceId.contains("request:寒暖差") &&
    ( $time.getTime()- created.getTime()) <  10 * 1000)
  $temp1:Event(dataType == "temperature" &&
    placeTag == $req.getplaceTag())
  $temp2:Event(dataType == "temperature" &&
    placeTag == "outside")
  not Event(dataType == "寒暖差" &&
    ($time.getTime()-created.getTime()) <  60 * 1000)
then
  eventStore.insert(new Event("complex_event",
    "寒暖差", $req.getplaceTag(), null, null,
    Math.abs(Double.valueOf($temp1.getValue().toString())
    - Double.valueOf($temp2.getValue().toString()))));
end
```

図 4　受動型複合イベントの例：寒暖差（Drools による実装）
Fig. 4　An example of passive complex event: temperature difference (implemented using Drools).

```
rule "傘提案"
when
  $time: TimerEvent()
  Event(deviceId.contains("calendar:user:taro") &&
    (created.getTime() - $time.getTime()) <
    45 * 60 * 1000)
  Event(deviceId.contains("placedetector:user:taro") &&
    value == "home.entrance")
  Event(deviceId.contains("weatherforecast") &&
    value == "rainy")
  not Event(deviceId == "rules:eventdetector" &&
    dataType == "event" && value == "rain:entrance" &&
    $time.getTime() - created.getTime() < 45 * 60 * 1000)
then
  eventStore.insert(new Event("rules:eventdetector",
  "event", "rain:entrance"));
end
```

図 5　能動型複合イベントの例：傘提案（Drools による実装）
Fig. 5　An example of active complex event: umbrella suggestion (implemented using Drools).

報サービスから取得する値が雨であれば，傘をもつよう対話エージェントを通して利用者に提示する．

このように，複数の IoT サービスによる原子イベントの組み合わせにより，実環境において様々かつ複雑な利用シーンに対応する複合イベントを実装可能である．また，複合イベントのルールでは，原子 IoT サービスインタフェースは引数という形式で記述する．そのため，同じ種類のインタフェースを用いる IoT サービスを利用する場合，新しいルールを作成する必要がなく，対話エージェント側でサービスの ID を手動的に指定する，またはサービスの登録情報に基づき自動的にマッチングすることによって実現可能である．

以上より，本研究で提案した IoT サービス基盤ソフトウェアを用いて実装された対話エージェントアプリケーションは，対話エージェントのための IoT サービスの相互運用性を実現し，受動型処理と能動型処理の両方に対応可能である．実際に，本研究の対話エージェントアプリケーションを AI アシスタントである Amazon Alexa のスキルとして実現している．

5. 評　　価

本研究で実装した基盤により IoT サービスの相互運用性が実現されるため，IoT アプリケーションの開発コストの削減が期待される．そこで，まず本研究の基盤を用いる場合の IoT サービスのバリエーション数と IoT アプリケーションのコーディング量について分析する．次に，本研究で IoT サービス基盤を用いて実装した対話エージェントの対話応答速度が，実用に耐えるものかを評価するため，IoT サービス基盤を用いない場合と比較する．

5.1　サービスのバリエーション削減の分析

本研究では，IoT サービス基盤上に標準化されたインタフェースを設けることで，異種の IoT デバイスと Web サービスの相互運用性を実現した．相互運用性の提供を目的とするサービス基盤では，どの程度アプリケーション開発者の開発コストを削減できるかについて，サービスのバリエーションの削減に関する分析が一つの重要な手段である [10]．本節では，この IoT サービス，特に複合イベントにおけるバリエーション削減効果について分析する．

本研究では，対話エージェントが原子 IoT サービスから直接得た原子イベントだけでなく，それらを組み合わせて取得する複合イベントも呼び出せるようにシステムを設計した．複合イベントのルールは事前に記述されており，原子イベントが入れ子になる形で実行される．複合イベントは対話エージェントからの引数をもとにした，イベント通知及びそのイベントの状態による特定のルールの実行により生成される．ある複合イベントのルールの構成原子イベント呼び出し数を EI，それぞれの原子イベント i に関連する IoT サービスの種類数を S_i とすると，ある複合イベントのバリエーション数 $V = \prod_{i=1}^{EI} S_i$ で表される．そのため，複合イベントのバリエーション数は容易に組合せ爆発を起こすことがわかる．例えば，実際に「建物から出るときに，その場所の体感温度を通知する」という複合

イベントがあった場合，それを構成する原子イベントは玄関での人検知イベント，その場所の温度取得イベント，その場所の湿度取得イベント，その場所の風速取得イベントに関する四つの原子 IoT サービスが必要である．それぞれの関連する原子 IoT サービスが 10 種類ずつあった場合，単純に計算すると，この複合イベントは 10,000 個のバリエーションをもつことがわかる．全てのバリエーションが常に使われるとは限らないが，ある IoT サービスを確認するたびにルールとして記述する必要がある．この場合，本研究手法では，複合イベントのルールを一つだけ実装する必要があるため，バリエーション数は大きく削減され，IoT アプリケーション開発者にとって，同じ機能をもつ IoT サービスの仕様の違いを意識せずに開発のコストを削減できる．

5.2 IoT アプリケーションのコーディング量の分析

本研究で提案した IoT サービス基盤を用いる場合，IoT デバイスと Web サービスの相互運用性を向上するため，サービスのバリエーションを削減できる．しかしながら，IoT サービス基盤を用いて IoT アプリケーションを開発するためには，IoT サービスの標準インタフェースとアダプターの実装が必要である．一方，IoT サービス基盤を用いない場合は，標準インタフェースとアダプターの開発が不要である．例えば，本研究で実装した対話エージェントにおいて，一つの DHT22 温度センサーを用いて温度値を取得する IoT アプリケーションを開発する．IoT サービス基盤を用いない場合は，温度センサーから発生する値を取得するコーディングが必要であり，合計 60 行のコードで実装される(注5)．IoT サービス基盤を用いる場合は，合計 139 行のコード（インタフェースの実装：68 行，アダプターの実装：66 行，サービスを呼び出すための実装：5 行）が必要であり，IoT サービス基盤を用いない場合より 2 倍以上の開発コストがかかる．次に，同じ DHT22 温度センサーを用いて新たに二つの IoT アプリケーションを独立に開発することを考える．IoT サービス基盤を用いない場合は，それぞれ新しい 60 行のコードが必要である．一方，IoT サービス基盤を用いる場合は，既にインタフェースとアダプターが実装されているため，それぞれサービスを呼び出すための 5 行のコードで新しい IoT アプリケーションを実装可能であり，IoT サービスの再利用が増えることによってコーディング量を削減できる．

IoT サービス基盤を用いる場合のコーディング量の削減効果を示すため，100 個の IoT アプリケーションを時系列に開発するシミュレーションを用いて，IoT 基盤を用いない場合と比較した．コード行数など各種パラメーターの値は本研究で実装した対話エージェントアプリケーションの実際値の範囲に従ってランダムに生成される．IoT 基盤を用いる場合の一つ IoT アプリケーションのコーディング量は IoT サービスのインタフェース（20 行〜90 行の乱数）とアダプター（50 行〜100 行の乱数），ルール（複合イベントを用いる場合，15 行〜30 行の乱数），サービスを呼び出すためのコード行数（5 行〜10 行の乱数）の合計である．また，新しく開発する IoT アプリケーションが既に開発した IoT サービスのインタフェースとアダプター，ルール（複合イベントを用いる場合）を確率的に再利用し，その再利用確率はアプリケーションの数が増えることによって増加する．IoT 基盤を用いない場合は，100 個の IoT アプリケーションがそれぞれ独立し，一つの IoT アプリケーションのコーディング量は，アプリケーションの主要機能（IoT サービス基盤を用いる場合のアダプターのコード量の 50%〜100% の乱数）とルール（複合イベントを利用する場合，IoT サービス基盤を用いる場合のルールのコード量の 80%〜120% の乱数）のコードで計算される．図 6 は各 IoT アプリケーションが 1 個の原子イベントのみを利用する際のコーディング量の累積値（50 回シミュレーションの平均）を示す．IoT サービス基盤を用いる場合のインタフェースの再利用確率はアプリケーションごとに 10% で増加し最大 90% であり，アダプターの再利用確率は 5% で増加し最大 80% である．図 7 は各 IoT アプリケーションが 2 個の原子イベントによる複合イベントを利用する際のコーディング量の累積値（50 回シミュレーションの平均）を示す．IoT サービス基盤を用いる場合のインタフェースとアダプターの再利用確率は図 6 と同様であり，ルールの再利用確率は 2 種類の比率で増加し（$\mathtt{rr} = 10\%, 20\%$）最大 80% である．

図 6，7 に示すように，IoT サービス基盤を用いない場合，コーディング量の累積値がアプリケーション数に対してほぼ線形に増える．IoT 基盤を用いる場合，導入時に IoT アプリケーション用のインタフェースとアダプターのコーディングコストがかかる．そのためアプリケーションの数が少ないと，累積コード行数が

(注5)：本研究の分析では，IoT アプリケーションのコーディング量は主要機能の部分のみカウントされる．

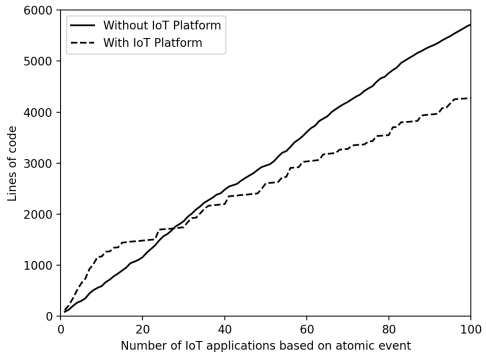

図 6　原子イベントを用いた IoT アプリケーション開発のコーディング量

Fig. 6　Lines of code for developing IoT applications based on atomic event.

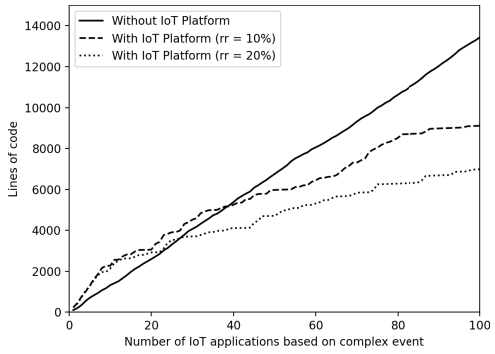

図 7　2 個の原子イベントによる複合イベントを用いた IoT アプリケーション開発のコーディング量

Fig. 7　Lines of code for developing IoT applications based on complex event (each consisting of two atomic events).

IoT 基盤を用いない場合と比べて多くなる．一方，IoT アプリケーション数が増えることによって，実装されたインタフェースとアダプター，ルール（複合イベントを用いる場合）の再利用率が増加するため，コード行数の累積値の増加が緩やかになり，あるアプリケーション数に達したときに，累積コード行数が IoT 基盤を用いない場合と比べて少なくなる．また，図 7 の場合，複合イベントのルールの再利用確率の増加が 10% から 20% に変化する際に，IoT サービス基盤を用いる場合のコーディング量の削減効果がより高まる．この結果から，IoT サービスの再利用が見込まれる場合，IoT サービス基盤を用いることによって，IoT アプリケーション開発のコーディング量を削減できる事がわかる．

5.3　IoT サービス基盤の応答速度の評価

本研究では，IoT アプリケーションとして対話エージェントを実装し，従来のモデルに IoT サービス基盤を搭載する形で構築した．IoT デバイスを直接呼び出す手法と比べ，標準インタフェースを通してイベント処理を行うため，オーバーヘッドが発生する．ただし，その実行時間の増加分はインタフェースを経由するという単純な処理であるため，実行速度に対する影響が小さいはずである．また，Web サービスを呼び出す場合に関しては，IoT サービス基盤側が定期的に Web サービスに対するポーリングの結果をデータベースに記録し，Web サービスの呼び出しをイベント処理として扱うため，直接 Web サービスにリクエストを送る場合よりもデータ取得が速くなることが予想される．それらの対話応答速度が対話応答全体に与える影響を評価するため，対話応答時間を計測し，IoT 基盤を通さずにサービスを実行する対話エージェントと比較した．比較用に作成した IoT サービス基盤を搭載しないシステムのイベント処理サーバは，本研究で実装したシステムと全く同様の環境である．

対話応答時間の計測においては，八つのシナリオを用いて実験を行った．各シナリオで用いた IoT サービスを表 2 に示す．シナリオ A〜D は Web サービスを利用し，E〜H はセンサーを利用している．また，D（寒暖差），G（不快指数），H（絶対湿度）はそれぞれ 2 個の原子イベントからなる複合イベントを処理するシナリオである．IoT サービス基盤上の標準インタフェースを 3 種類（温度，湿度，降水確率），登録されている IoT サービスを 5 種類（OpenWeatherMap の温度 API と湿度 API, Lifesocket(注6) の降水確率 API, DHT22 の温度センサーと湿度センサー）利用している．各シナリオを 30 回実行した平均の対話応答時間の比較結果を図 8（対話理解と対話生成の時間を含む），図 9（対話理解と対話生成を除く）に示す．

図 8，9 の（a）に示すように，Web サービスを利用する場合，本研究で提案した基盤を用いたシステムの方が応答時間が短いことがわかる．これは IoT サービス基盤では Web サービスもイベントベースのインタフェースに従って，定期的に基盤からリクエストを送り，イベントデータベースにその Web サービスからの応答データを保存しているからである．そのため，リクエストごとに Web サービスを呼び出す IoT サービス基盤なしのシステムの場合と比較して，標準インタフェースを通してもレスポンスが速くなり，かつ安定

（注6）：https://lab.life-socket.jp/

表 2　応答速度を評価するシナリオ（Case ID: A〜H）
Table 2　Scenarios for evaluating response time (Case ID: A-H).

ID	eventType	interface	deviceID	category
A	原子	温度	OpenWeatherMap	Web サービス
B	原子	湿度	OpenWeatherMap	Web サービス
C	原子	降水確率	Lifesocket	Web サービス
D	複合	温度	OpenWeatherMap	Web サービス
		温度	OpenWeatherMap	Web サービス
E	原子	温度	DHT22	センサー
F	原子	湿度	DHT22	センサー
G	複合	温度	DHT22	センサー
		湿度	DHT22	センサー
H	複合	温度	DHT22	センサー
		湿度	DHT22	センサー

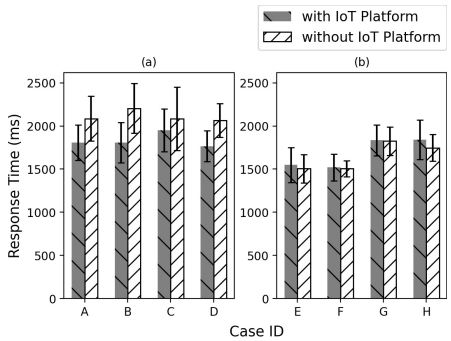

図 8　対話理解と対話生成を含む応答時間（N = 30）：(a) Web サービスの利用，(b) センサーの利用
Fig. 8　Response time including dialogue understanding and generation (N = 30): (a) scenarios using Web services, (b) scenarios using sensors.

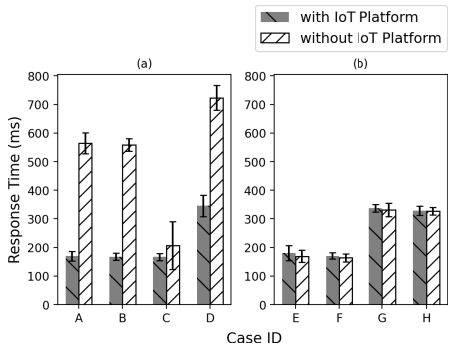

図 9　対話理解と対話生成を除く応答時間（N = 30）：(a) Web サービスの利用，(b) センサーの利用
Fig. 9　Response time excluding dialogue understanding and generation (N = 30): (a) scenarios using Web services, (b) scenarios using sensors.

する傾向が読み取れる．

一方，図 8，9 の（b）に示すように，センサーを利用する場合，IoT 基盤を利用しない対話エージェントシステムのレスポンスが速い．これは，Web サービスを呼び出すシステムでは，IoT サービス基盤を用いるシステムと用いないシステムで別々のデータの取得方式であったのに対して，今回は両方ともデータベースに保存してあるデータを取得することに起因する．すなわち，同じ方式でデータを取得する場合，IoT サービス基盤を用いた対話エージェントシステムは標準インタフェースを通してデータベースからデータを取得するため，オーバーヘッドが発生する．ただし，応答時間の差は小さく，図 8 の場合，平均のオーバーヘッドは，原子イベントのシナリオ E，F では 1.90%，複合イベントのシナリオ G，H では 2.90% であった．また図 9 の場合，原子イベントのシナリオ E，F では 5.30%，複合イベントのシナリオ G，H では 1.14% であった．この結果から，本手法が応答速度に与えるオーバーヘッドは限定的である事がわかる．

また，IoT サービス基盤を利用したシステムの複合イベント (Case：D，G，H) と原子イベント (Case：A，B，C，E，F) の応答時間の比較では，複合イベントが基盤へのリクエスト回数が原子イベントよりも多いため，応答時間が約 2 倍に増えている．しかし，いずれのシナリオにおいて IoT サービス基盤が応答速度に与える影響がわずかであり，実用上問題ないと考える．

一方，本研究では応答速度に関する評価は小規模で実施されているため，IoT サービス数の増加や IoT アプリケーションからのアクセス集中，Web サービスに対するポーリングによる IoT サービス基盤への負荷の影響が少ないが，IoT サービス基盤を大規模に運用する際に，これらの課題を解決する必要がある．例えば，Web サービスの利用が少ないときに，ポーリングにより大量な無駄なトラヒックが発生し，IoT サービス基盤の性能低下につながる．そのため，Web サービスに対するポーリングの頻度を基盤の負荷と利用の状況に応じて自動的に調整する仕組みを IoT サービス基盤上に設計する必要がある．また，IoT サービス数や利用者数の増加によって，IoT アプリケーションから大量なアクセスが発生する際に，IoT サービス基盤のサーバがボトルネックになる可能性が多いため，今後，分散型 IoT サービス基盤アーキテクチャを設計し，IoT サービスの実行・管理とイベントデータベース，センサーデータを蓄積するデータベースの機能を分散するメカニズムを実現する必要がある．

6. むすび

本研究は，異種の IoT デバイスと Web サービスが

混在する環境において，アプリケーションを容易に構築するため，相互運用性を保証する IoT サービス基盤を実現した．まず，IoT デバイスと Web サービスを横断するサービスインタフェースを標準化し，イベント処理に基づくアーキテクチャを提案した．次に，サービスからデータを収集する機構とサービスを駆動する機構，複合イベントを処理する機構を設計することによって，能動型及び受動型のイベント処理を実現した．また，提案した IoT サービス基盤の応用例として，対話エージェントを実装した．更に，サービスのバリエーションとコーディング量の観点から IoT アプリケーション開発のコスト削減を分析し，対話エージェントアプリケーションの応答速度を評価することで提案した基盤の有効性と実用可能性を検証した．

謝辞　本研究は，日本学術振興会科学研究費基盤研究 (B)（21H03556, 2021 年度～2023 年度；21H03561, 2021 年度～2024 年度）及び挑戦的研究（萌芽）（20K21833, 2020 年度～2022 年度）の補助を受けた．

文　　献

[1] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, “Internet of Things: A survey on enabling technologies, protocols, and applications,” IEEE Communications Surveys & Tutorials, vol.17, no.4, pp.2347–2376, 2015.

[2] D. Bandyopadhyay and J. Sen, “Internet of Things: Applications and challenges in technology and standardization,” Wireless Personal Communications, vol.58, no.1, pp.49–69, 2011.

[3] Y. Zhang, L. Duan, and J.L. Chen, “Event-driven SOA for IoT services,” 2014 IEEE Int. Conf. Services Computing, pp.629–636, 2014.

[4] B. Cheng, D. Zhu, S. Zhao, and J. Chen, “Situation-aware IoT service coordination using the event-driven SOA paradigm,” IEEE Trans. Network Service Management, vol.13, no.2, pp.349–361, 2016.

[5] E. Wu, Y. Diao, and S. Rizvi, “High-performance complex event processing over streams,” Proc. 2006 ACM SIGMOD International Conference on Management of Data, pp.407–418, 2006.

[6] Y. Murakami, D. Lin, M. Tanaka, T. Nakaguchi, and T. Ishida, “Service Grid architecture,” The Language Grid, pp.19–34, Springer, 2011.

[7] T. Ishida, Y. Murakami, D. Lin, T. Nakaguchi, and M. Otani, “Language service infrastructure on the Web: the Language Grid,” Computer, vol.51, no.6, pp.72–81, 2018.

[8] D. Lin, Y. Murakami, and T. Ishida, “Integrating Internet of Services and Internet of Things from a multiagent perspective,” Massively Multi-Agent Systems II, eds. by D. Lin, T. Ishida, F. Zambonelli, and I. Noda, pp.36–49, Springer, 2019.

[9] O. Novo and M.D. Francesco, “Semantic interoperability in the IoT: Extending the Web of Things architecture,” ACM Trans. Internet of Things, vol.1, no.1, pp.1–25, 2020.

[10] 中口孝雄，村上陽平，林　冬惠，石田　亨，“高階関数を導入した階層的サービス合成記述，” 信学論（B），vol.99, no.10, pp.834–842, Oct. 2016.

（2021 年 3 月 22 日受付，7 月 24 日再受付，
9 月 3 日早期公開）

大川　楠人

2021 京都大学大学院社会情報学専攻修士課程了．サービスコンピューティング，Internet of Things，対話エージェントに興味をもつ．

林　　冬惠　（正員）

2008 京都大学大学院社会情報学専攻博士課程了．博士（情報学）．現在同専攻特定准教授．サービスコンピューティング，マルチエージェントシステム，Internet of Things，異文化コラボレーションの研究に従事．

村上　陽平　（正員）

2006 京都大学大学院社会情報学専攻博士課程了．博士（情報学）．現在，立命館大学情報理工学研究科准教授．電子情報通信学会サービスコンピューティング研究専門委員会を立ち上げるなどサービスコンピューティングの研究に従事．異文化コラボレーションのための多言語サービス基盤「言語グリッド」の研究開発を推進．

中口　孝雄　（正員）

2017 京都大学大学院社会情報学専攻博士課程了．博士（情報学）．現在，京都情報大学院大学准教授，株式会社コネクションズ代表取締役．サービスコンピューティングの研究，社会実装に従事．