IoT アプリケーション開発のためのメディア制御・処理フレームワーク

小牧大治郎，山口俊輔，篠原昌子，堀尾健一，村上雅彦，松井一樹

概要： IoT アプリケーション開発において，現場に設置されたカメラやマイクから取得される映像や音声などのメディアデータを取り扱うためには，コーデックやプロトコル，画像処理など多岐にわたる知識が必要なため，アプリケーション開発のハードルが高い．そこで本研究では，IoT アプリケーション開発でメディアデータを容易に扱うことができるフレームワークの開発を行った．提案フレームワークは，メディア入出力デバイスの抽象化，ゲートウェイとクラウド上でのメディアサービス分散実行，GUI によるメディアサービス定義を可能にする機能を備える．本稿では，提案フレームワークを用いて実際に IoT アプリケーションを試作し，提案フレームワークの有効性について評価，考察を行った結果を報告する．

A Multimedia Control and Processing Framework for IoT Application Development

Daijiro Komaki, Shunsuke Yamaguchi, Masako Shinohara, Kenichi Horio,
Masahiko Murakami, Kazuki Matsui

1. はじめに

コンピュータ等の情報機器だけでなく，家電や車など，身の回りのモノがインターネットに接続され，相互に通信することで，自動で認識や制御を行う IoT（Internet-of-Things）の時代が到来しつつある．IoT アプリケーション開発では，特定の IoT アプリケーションのために，センサやアクチュエータ等のデバイスを設置するだけではなく，すでに設置されたデバイスを，多種多様な IoT アプリケーションから利用可能にすることで，これまでにない利便性や価値が実現されることが期待される [12,15].

またこのような IoT アプリケーションでは，温度，加速度のようなセンサデータだけでなく，カメラ，マイク等のデバイスから取得される映像，音声といったメディアデータも重要である．例えば，「監視カメラから不審な動きの人物を検出する」など，メディアデータをセンサとして利用するケースや，「ライブでの観客の盛り上がりに応じて，遠隔地へ中継する映像にエフェクトをかける」など，センサデータを入力としてメディアデータを加工するケースなど，センサデータとメディアデータを組み合わせることで多様な IoT アプリケーションが提供可能になる．しかし，映像や音声などのメディアデータを取り扱うためには，コーデックやプロトコル，画像処理など多岐にわたる知識が必要なため，アプリケーション開発のハードルが高い．

一方，Kurento [5,14] や Skylink [9] など，メディアを使ったアプリケーション開発を容易にするフレームワークがこれまでに数多く提供されている．これらのフレームワークを用いることで，映像・音声のコーデックやフォーマットの差異を意識することなく，アプリケーションの開発が可能となる．例えば，通話機能や AR （Augmented Reality）機能などを持つアプリケーションを，フレームワークが提供する認識，加工等の処理部品を組み合わせるだけで容易に開発できる．しかし，これらのフレームワークを用いたとしても，IoT アプリケーション開発においては，開発者が，個々のデバイスの制御インタフェースや通信プロトコル等の仕様をあらかじめ知り，それに合わせた IoT アプリケーションを開発しなければならない．また，全てのメディアデータをクラウドへ集約する必要があるため，ネットワーク帯域を消費してしまう．

そこで筆者らは，上記の課題を解決し，メディアを使った IoT アプリケーションの開発を容易にするフレームワークの設計および実装を行った．提案フレームワークは以下の特長を備える．

- **メディア入出力デバイスの抽象化**
  提案フレームワークは，メディアストリームの送受信の開始や停止方法，デバイスとサーバ間のメディアセッションの確立方法など，デバイス毎に異なる仕様を隠蔽する機構を提供する．これにより，IoT アプリケーション開発者は，デバイスの詳細を意識しなくても開発でき，また開発したアプリケーションを異なるデバイスへ適用できる．
- **ゲートウェイとクラウド上での分散実行**
  現場に設置され汎用的な処理を実行可能な機器であるゲートウェイとクラウド，それぞれで独立して動作するメディアサーバを連携させ，一つのメディアストリームを分散して処理・制御する機構を提供する．こ

株式会社 富士通研究所

れにより，ネットワーク帯域の節約や現場への即座のレスポンスを可能とする IoT アプリケーションを簡単に開発できる．

- **GUI によるフロー定義で開発可能**

  入出力デバイスとメディア処理のブロックを接続するだけで，ゲートウェイとクラウドに分散したメディアサービスを直感的に定義できる GUI を提供する．これにより，一部のメディア処理ブロックの変更や追加などの試行錯誤が容易になる．

また本稿では，実装した提案フレームワークを用いて，筆者らが実際に行った PoC（Proof-of-Concept）事例を紹介し，IoT アプリケーション開発における複雑性，生産性，試行錯誤の容易性等の観点から評価・考察を行う．

2. 関連研究

2.1 メディアの扱いを容易にするフレームワーク技術

映像や音声などのメディアデータを扱うアプリケーションの開発を容易にするフレームワークとして, GStreamer [1] がある．GStreamer では，パイプラインアーキテクチャに従って，メディアの入出力や，コーデック，メディア処理等のエレメント群を組み合わせるだけで，メディアを利用したアプリケーションを開発できる．ただし GStreamer では，アプリケーション開発を始める時点で，利用するデバイスやプロトコル，メディアのコーデック等を決めておき，それに従ってパイプラインを決める必要がある．

Kurento Media Server は，GStreamer をベースにした WebRTC 対応のメディアサーバであり，提供されるライブラリを利用するだけでテレビ会議や AR などメディアストリーミング機能を持つ Web アプリケーションを容易に開発できる．Kurento Media Server はメディアのコーデックやフォーマットなどの差異を吸収する機構を持っており，メディア処理に詳しくない Web 開発者であっても，メディア処理部品同士を接続することで，簡単にメディアを使った Web アプリケーションを開発できる．また，コンピュータビジョンライブラリ OpenCV を使ってメディア処理部品を簡単に開発できる仕組みも提供しており，画像処理やコンピュータビジョンの技術者が，アプリケーションとは独立してメディア処理部品を追加できる．

これらのフレームワークでは，メディアの取り扱いが容易になるものの，個々のデバイスの制御インタフェースや通信プロトコル等の仕様に合わせてアプリケーションを開発する必要があるため，すでに現場に設置されたデバイスを活用する IoT アプリケーション開発に適しているとはいえない．

2.2 IoT アプリケーション開発基盤技術

現場に設置した複数のセンサやアクチュエータなどを組み合わせた IoT アプリケーションの開発を容易にするための基盤がいくつか提供されている．Kii Cloud [3] は，IoT アプリケーション開発のための BaaS (Backend-as-a-Service) であり，デバイスをクラウドで登録・管理・仮想化する機能を提供している．IoT アプリケーション開発者は，API を使って，クラウド上で仮想化されたデバイスの持つ機能を組み合わせ，新規 IoT アプリケーションを開発できるため, デバイス毎の通信方式の差異等を意識する必要がない．

IBM Bluemix [2] は，Kii Cloud と同様のデバイス管理機能に加え，GUI 上でデバイスの入出力と各種データ処理を組み合わせたフローを定義して IoT アプリケーションを簡単に開発できる．また Blackstock [13] らは，Bluemix 上でフロー定義のために用いられるソフトウェアである Node-RED [7] を拡張し, 複数マシン間で連携したデータ処理を定義できるように拡張している．Yahoo の提供する myThings [6]も同様に，グラフィカルなフロー定義による IoT アプリケーション開発が可能で, デバイスと様々な Web サービスとを簡単に接続できる．

これらの基盤を用いることで，複数のデバイスを連携させた IoT アプリケーションの開発が容易になるが，カメラやマイクから取得されるメディアデータと連携する IoT アプリケーションを開発するためには，別途，前述のようなメディアフレームワークと連携させる必要がある．

センサデータとメディアデータを組み合せた IoT アプリケーションの開発を容易にするものとして ThingStore [11] がある．ThingStore では，モノを「真偽を出力するもの」と抽象化することで，カメラや各種センサから出力されるイベントを組み合わせて IoT アプリケーションを開発・デプロイするための基盤機能を提案している．この仕組みにより，モノを提供する人と IoT アプリケーションを提供する人のコラボレーションが促進されることが期待できる．しかし，メディアストリームも真偽のデータストリームに抽象化されるため，本研究で対象としているデバイス間でメディアストリームそのものをやりとりするような IoT アプリケーションの開発には用いることができない．

2.3 提案フレームワークの位置づけ

本研究では, 2.2 節で記述したような IoT アプリケーション開発基盤において，デバイス間の制御情報だけでなく，デバイス間を流れるメディアストリームを容易に制御・処理できることを目指している．そのため，IoT アプリケーション開発向けに Kurento Media Server のメディアサーバ機能を拡張し，メディアを使った IoT アプリケーションの開発を容易にするフレームワークを提案する. 具体的には，提案フレームワークは，メディアストリームを入出力するデバイスとセンサ･アクチュエータを組み合わせることで，センサ系とメディア系が相互に作用するような IoT アプリケーションの開発を容易にすることを目的とする．

3. Kurento Media Server

提案フレームワークの実装のベースとした Kurento Media Server の概要について説明する. Kurento Media Server は WebRTC 対応のメディアサーバで，1 対 1 の通話機能やグループ会議機能に加え，サーバ上でメディアの変換，加工, 分析などの処理をする Web アプリケーションを容易に開発できる．Kurento Media Server は，表 1 に示す入出力のエンドポイントや，変換，加工，認識等の処理のフィルタといったモジュールを提供しており，サーバ上でこれらのブロックを組み合わせてパイプラインを作成するだけで，コーデックやフォーマットの違い等を意識することなく Web アプリケーションを開発できる．例えば，図 1 に示すようなパイプラインを組み立てると，WebRTC を用いてブラウザから取得したカメラ映像を受け取り，その映像中の顔の位置に任意の画像を重畳して表示し，ブラウザにループバックして表示しつつ，サーバで録画する処理になり，人物の顔を任意の画像に変換する Web アプリケーションが開発できる．

Kurento Media Server は Java 及び JavaScript のクライアントライブラリを提供しており，Servlet や Node.js で実装する Web サーバプログラム等から，Kurento Media Server の機能 (API) を利用できる. 図 1 の例を実際に作成する手順を図 2 のアーキテクチャ上に示す. まず Web ブラウザ側では，SDP [8]（接続情報，利用可能なコーデック，プロトコル等の情報を記述したもの）オファーを作成し，Web サーバに送信する．Web サーバは，アプリケーションロジックに従って, クライアントライブラリを使用し Kurento Media Server 上にパイプラインを作成する．その後，端末との端点となる WebRtcEndpoint に対して，受け取った SDP オファーを処理させ，SDP アンサーをクライアントである Web ブラウザへ返信する．最後に Web ブラウザ側で，受信した SDP アンサーをもとに WebRtcEndpoint との間のセッションを確立させ，WebRTC を使って映像を送り始めると，処理された映像が返ってくる．今回の提案フレームワークでも，クライアントライブラリを利用して Kurento Media Server の制御を行っている．

Kurento Media Server が提供するフィルタは，メソッドを持ち，Web アプリケーションがフィルタの持つパラメータの変更等を行える．また，各フィルタの発行するイベントを購読することもできる．例えば図 3 に示すように，Web アプリケーションが画像を重畳表示するフィルタの画像の種類を変更してフィルタの動作を制御することや，バーコード認識フィルタの認識結果を購読しておき，認識された文字列を受け取って使用することができる．提案フレームワークでは, このような Kurento Media Server の持つ特徴を活用し，IoT アプリケーション開発においてより使いやすくするための機能をいくつか実装した．

表 1: Kurento Media Server の提供モジュール群

エンドポイント（入出力）	
WebRtcEndpoint	WebRTC を用いた送受信
RtpEndpoint	RTP を用いた送受信
PlayerEndpoint	ローカルファイルや RTSP の再生
RecorderEndpoint	ローカルファイルに録画
フィルタ（変換・加工・認識等）	
FaceOverlayFilter	顔を認識し、任意の画像を重畳して表示
ZBarFilter	バーコード、QR コードを認識
GStreamerFilter	GStreamer のフィルタを適用

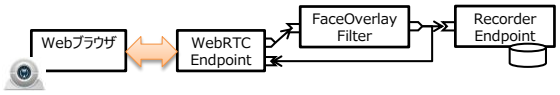

図 1: 入出力、処理のブロックの組合せによるアプリケーション開発

4. 機能要件

我々は，メディアを利用した IoT アプリケーションとして以下のようなユースケースを想定している．

1. あるカメラで撮影した映像を，あるディスプレイに投影
2. あるマイクから音声認識した結果のテキストを，あるカメラからの映像に重畳して近くのディスプレイに表示
3. あるカメラで撮影した映像から人数を数え，人数変化の時系列データからイベントを判定
4. 現場の温度が一定以上になったときだけ，あるデバイスからの映像を録画

このような IoT アプリケーションを容易に開発できるようにするため，フレームワークの備えるべき機能の要件を検討し，以下の 4 つの要件を抽出した．

4.1 メディア入出力デバイスの抽象化

上記のようなユースケースを考えたとき，特定の現場向けに IoT アプリケーションを開発するより，一度開発した IoT アプリケーションを複数の現場で実行できることが望ましい．しかし，現場には RTSP 対応のカメラ，独自インタフェースで SDP の交換を行うカメラ，USB 接続のカメラ等，様々な種類のメディア入出力デバイスが存在し，それぞれ WebRTC，RTP，HTTP など異なる通信プロトコルでメディアデータを送受信する可能性がある. 従来のように，Web アプリケーションにメディアを使った機能を実装する場合は，デバイスの制御方法，利用プロトコルが既知のため問題とならないが，IoT の世界では，設置されたデバイスを様々な IoT アプリケーションから利用することが想定される．したがって，このようなデバイスの仕様の差異を

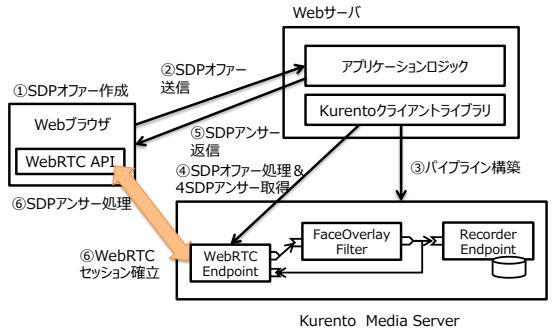

図 2: シグナリングとメディア処理が分離されたアーキテクチャ

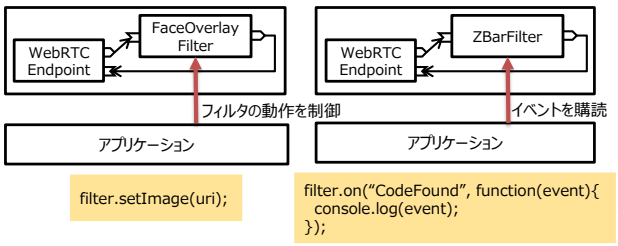

図 3: フィルタとのインタラクション

意識することなく IoT アプリケーションを開発できることが望ましい.

4.2 ゲートウェイとクラウドでの分散実行

メディアデータはセンサデータに比べて大容量であるため，全てのデータをクラウドに集約すると，ネットワーク帯域を大量に消費してしまう.ユースケース 2 のように，現場の映像を処理した結果を，折り返し現場で利用する場合や，ユースケース 3 のように，IoT アプリケーションがメディアデータから抽出されるメタデータだけを必要とする場合は，現場に近い場所で処理を行うことで，ネットワーク帯域を節約することや即座のレスポンスが可能となる.

このように，一つのメディアストリームに対する処理を，現場に設置されたゲートウェイとクラウド間で分散して実行することが有効だと考えられるが，そのためには，複数場所で起動しているメディアサーバ間でメディア送受信の口を開き，セッションの確立処理を行う必要があるなど，煩雑な処理が必要となる．したがって，開発者がこのようなことを意識しなくても複数場所でのメディア処理の分散実行可能になることが望ましい.

4.3 外部との連携

ユースケース 1 や 2 のように，デバイスからデバイスへ送信されるメディアストリームを制御・処理するだけでなく，メディアストリームから抽出されるメタデータの時系列変化からイベントを検出するユースケース 3 や，温度等のセンサデータの変化を起点にメディアストリームを制御するユースケース 4 を実現するためには，既に存在している時系列分析や複合イベント処理などの機能を備えた既存の IoT 系プラットフォームと容易に連携できることが望ましい.

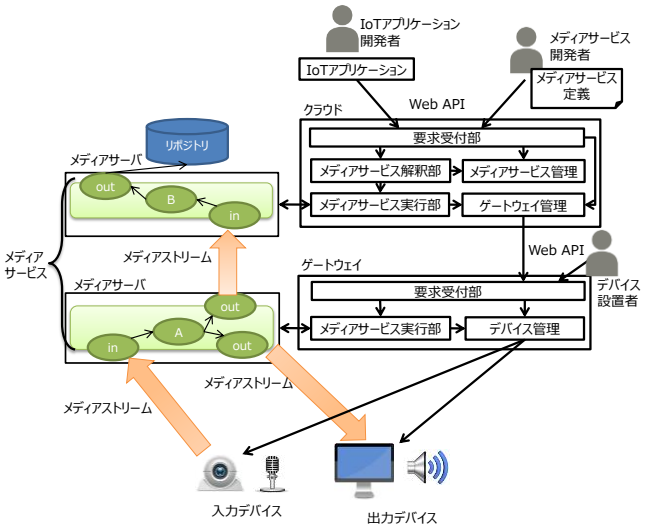

図 4: 提案フレームワークのアーキテクチャ

4.4 シンプルなメディアサービス定義の記述

Kurento Media Server のようなメディアサーバを使うと，複数の処理ブロックを組み合わせるだけで，メディアを使った Web アプリケーションを容易に開発でき，処理ブロックを入れ替えることで，試行錯誤や派生 Web アプリケーションの開発を容易に行える．この特徴を継承しつつ 4.1〜4.3 の要件を満たせるようなメディアサービスの定義を，シンプルに記述できるようになることが望ましい.

5. アーキテクチャの概要

4 章で述べた機能要件を満たすため，提案フレームワークのアーキテクチャを設計した（図 4）．ここで，メディアサービスとは，提案フレームワーク上で実行される一連のメディア処理の実体であり，IoT アプリケーションとは，提案フレームワークの提供する API を用いてメディアサービスを利用する実体である．提案フレームワークでは，ゲートウェイ（複数可）とクラウドで，それぞれ独立してメディアサーバを動作させ，メディアサーバ間を連携させる機能を持つモジュールを，各メディアサーバに配備する構成とした．各ゲートウェイ上に配備されたモジュールをクラウド上のモジュールで集約するスター型のトポロジを形成する．なお提案フレームワークは，メディアサーバとして Kurento Media Server を採用したが，他のメディアサーバであっても同等のアーキテクチャを実現することは可能である．以下では，各コンポーネントの持つ責務について記述する.

5.1 要求受付部

要求受付部は，クライアントから各種リクエストを受け付ける HTTP サーバを開いており，表 2 に示すリクエストを受け付ける．具体的には，デバイスの登録，ゲートウェイの登録，メディアサービスの登録，メディアサービスの実行・停止等の処理を受け付ける．デバイス設置者は，ゲ

表 2: 提案フレームワークが提供する Web API 一覧

URI	メソッド	説明	パラメータ
/service	GET	登録されたメディアサービスの一覧を取得	
/service	POST	新規メディアサービスの登録	id
			Service
/service/:id	GET	id で指定したメディアサービス定義を取得	id
/service/:id/create/:params	GET	id で指定したメディアサービス定義に引数を与え、メディアサービスの実体を作成	id
			params
/pipeline	GET	実行中のメディアサービスの一覧を取得	
/pipeline	POST	メディアサービスの実体を作成	Service:
/pipeline/:id	GET	id で指定したメディアサービスの詳細を取得	id
/pipeline/:id/:method	GET	id で指定したメディアサービスを操作(開始・一時停止・停止・解放)	id
			method
/gw	GET	登録されたゲートウェイの一覧を取得	
/gw	POST	新規ゲートウェイの登録	key
			uri
/device	GET	登録されたデバイスの一覧を取得	
/device	POST	新規デバイスの登録	key
			uri

ートウェイ側モジュールが公開する API を利用してデバイスを登録する．一方メディアサービス開発者，IoT アプリケーション開発者は，クラウド側モジュールが公開する API を利用して，メディアサービスの登録，実行を行う．

5.2 メディアサービス管理部

メディアサービス管理部では，JSON (JavaScript Object Notation) で記述されたメディアサービスの定義情報を, ID に紐づけて管理する．ゲートウェイ側には実行時に配備するため，メディアサービスの定義情報はクラウド側で一元管理するものとした．

5.3 メディアサービス解釈部

メディアサービス解釈部は，メディアサービス管理部で管理される JSON 形式で記述された定義情報を，メディアサービス実行部で実行可能な形式に変換する. 具体的には，クラウドで実行する処理部分とゲートウェイで実行する処理部分の切り分けを行う．

5.4 メディアサービス実行部

5.3.の解釈結果に基づいて，メディアサーバ上に，具体的なメディア処理部品の配備，制御を行う．この際，メディアサーバ上にメディア処理ブロックを配備するだけでなく，ゲートウェイ管理部，デバイス管理部と協調して動作し，デバイスや，ゲートウェイ上のメディアサービスとのセッションを確立する．具体的にはクラウド側モジュールのメディアサービス実行部では，ゲートウェイ管理部と協調して，指定したゲートウェイへのメディアサービスの配備，実行，停止などの操作，メディアセッションの確立処理などを実行する．一方，ゲートウェイ側モジュールのメディアサービス実行部は，デバイス管理部と協調して，デバイスとメディアサーバ上のエンドポイント間のセッション確立を行う．

5.5 ゲートウェイ管理部

各ゲートウェイの登録 ID（現場を識別する ID）と接続情報(URL)の対応関係を保持し，ゲートウェイを操作するインタフェースを提供する．メディアサービス実行部からの要求を受信すると，その命令をゲートウェイ側モジュールの要求受付部へ送信する．

5.6 デバイス管理部

各デバイスの登録 ID と接続情報（URL，ソケット等）の対応関係を保持し，デバイスとの SDP の交換，映像ストリームの送受信開始・停止などの操作を行うインタフェースを提供する．

5.7 利用の流れ

以下に, 実際に現場にゲートウェイやデバイスを設置し，メディアサービスを実行するまでの流れを説明する．ここでは，登場人物として，図 4 に示す現場管理者，メディアサービス開発者，IoT アプリケーション開発者の 3 人のプレーヤを考える．それぞれ別々の人物であっても同一の人物であってもよい．

① ゲートウェイの登録

現場管理者は，現場に設置したゲートウェイ機器上で，設定ファイル中の現場 ID を編集し，ゲートウェイ用プログラムを起動すると，自動でクラウド側へゲートウェイ情報の登録リクエストが送られる．

② デバイスの登録

現場管理者は，ゲートウェイ側プログラムが開く Web API を使って現場に設置するデバイスの登録を行う．このときデバイスの ID とデバイスの種類を指定する．デバイス ID はゲートウェイ毎に管理するので，ゲートウェイ内で一意に識別できる ID であればよい．

③ メディアサービスの登録

メディアサービス開発者は，図 5 のような JSON 形式でメディアサービスを記述し，クラウド側の Web API を使って登録を行う．

④ メディアサービスの実行

```
[
 {
  id: 0,
  //エンドポイント、フィルタのタイプを指定
  type: “DeviceEndpoint”,
  //現場に設置されたデバイスを特定するためのid
  key:“camera001”,
  //実行箇所がを指定(“cloud”か”gw”を明示的に指定)
  place: “gw”,
  //配備するGWを特定するためのid。$から始まる文字列は変数
  front_id: “$0”,
  out:[0]
 },
 {
  id: 1,
  type: "FaceOverlayFilter",
  //フィルタ固有のプロパティ。重畳する画像URLを指定
  img: “http://XXXX/hat.jpg”,
  place: "gw",
  in:[0],
  out:[1]
 },
 {
  id: 2,
  type: "RecorderEndpoint",
  place: "cloud",
  in:[1]
 }
]
```

図 5: メディアサービスの記述例

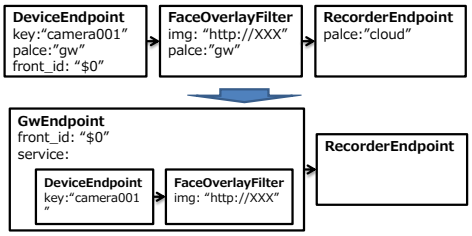

図 6: メディアサービス解釈の様子

IoT アプリケーション開発者はメディアサービス ID を使って，指定したメディアサービスの実行・操作を行う Web API を呼び出して，メディアサービスを IoT アプリケーションにつなぐ．

6. 機能の詳細

以下では，5 章で述べたアーキテクチャ上の各コンポーネントが持つ機能の詳細な動作について説明する．

6.1 メディアサービスの解釈・実行機能

メディアサービス開発者は，図 5 に示すような JSON 形式で記述して登録する．この例は，現場に設置されたカメラの映像を，ゲートウェイ上で顔領域に指定画像を重畳したものを，クラウド側で録画する．**type** でフィルタ・エンドポイントの種類を指定，**place** で実行場所を指定，**front_id** で適用する現場を指定，**in** と **out** でフィルタ・エンドポイント間の入出力関係を定義する．その他の要素は，フィルタ・エンドポイント固有のパラメータである．

メディアサービス解釈部では，JSON 配列で表現されたグラフ構造の定義情報をゲートウェイ側の処理部分と，クラウド側の処理部分に切り分け，ゲートウェイ側のグラフを内包した GwEndpoint を新たに作成する（図 6）．GwEndpoint は，ゲートウェイ側で実行される一連の処理と，実行されるゲートウェイの ID を保持する．この解釈後の情報をもとに，Kurento Media Server 上に，対応する処理ブロックを作成・初期化し，それらを接続する．

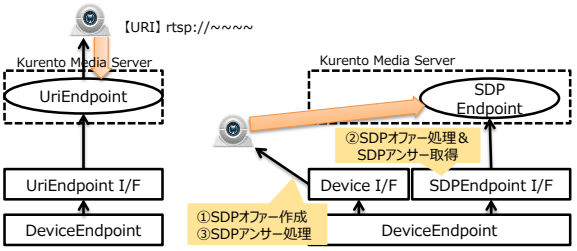

図 7: デバイス毎に異なる初期化処理を隠蔽

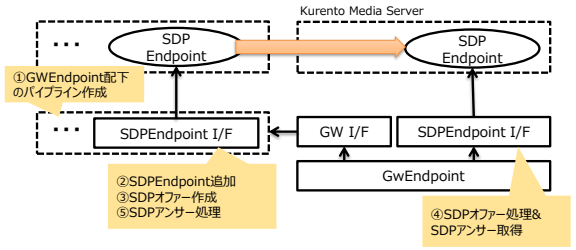

図 8: ゲートウェイ・クラウド間でのセッション確立

また，メディアサービス定義中に変数を定義可能にしており，図 5 の例では，メディアサービスを適用する現場 ID を変数にしておくことで，実行時に適用現場を決定できる．

6.2 デバイス抽象化機能

Kurento Media Server には，ネットワーク経由でのメディアの入出力に対応するエンドポイントとして，RTSP，WebRTC/SDP，RTP/SDP に対応したものがある．RTSP 対応のカメラの場合は，リソースの URL を指定するだけでセッションを確立でき，RTSP のプロトコル仕様に則って，再生・停止などの操作も行える．一方，WebRTC，RTP の場合は，デバイスとサーバ間で SDP の交換を行ってセッションを確立する必要がある．さらに，再生，停止などの処理は，デバイス毎に異なる操作インタフェースを提供している可能性があり，デバイス毎にセッション確立処理や操作方法が異なる．そこで，デバイスとのセッションを確立するための一連の処理を実装したクラスを用意した．5.7 節の利用手順でデバイス登録の際に指定したデバイスをもとに利用するクラスを変更するようにした．これにより，メディアサービス定義の記述の中では，デバイスの詳細は隠蔽され ID で指定するだけでよい．これにより，実行時に登録されたクラスに対応するセッション確立処理が行われる（図 7）．

6.3 ゲートウェイ・クラウド連携機能

一つのメディアストリームに対する処理パイプラインをゲートウェイとクラウドに分散して実行するためには，ゲートウェイとクラウドそれぞれで実行されるパイプライン間でセッションを確立する必要がある．図 8 に示すように，ゲートウェイ側で実行される一連の処理の末端に SDPEndpoint を自動で挿入し，ゲートウェイ側でパイプラ

インを生成する．その後，クラウド側に作成した SDPEndpoint との間にセッションを確立する．その後，クライアントからパイプライン操作命令が来れば，対応するゲートウェイ側のパイプライン，対応するデバイスへ，開始，停止などの命令を伝搬させる．

6.4 イベント管理機能

JSON 形式のメディアサービス定義の記述の中で，フィルタ間のイベントの購読を定義できるようにした．図 9 に示すように，購読対象フィルタ，対象イベント，イベント発生時のコールバック処理の組合せを定義しておけば，実行時にイベントの購読を行い，イベント発生時にコールバック関数が評価される．コールバック関数内部では，変数 **this** が購読するフィルタ自身を指すようにバインドする．

また図 10 に示すように，外部から指定 URL を呼び出すと，メディアサービス内で購読可能なイベントに変換する InputHttpEndpoint，メディアサービス内で起こったイベントをあらかじめ登録した URL にポストする OutputHttpEndpoint を利用可能にした．これらを使うことで，環境変化に応じてメディアストリームの加工や，認識結果の系列データを外部データベースに格納するようなメディアサービスの開発が可能となる．

6.5 GUI を用いたメディアサービス定義・実行

図 5 に示した JSON 形式の仕様に則ることで，コーディング不要でメディアサービスを定義可能であるが，処理ブロックの追加・編集などの試行錯誤をより簡単にするために，図 11，12，13 に示すような開発用クライアントを開発した．この開発用クライアントの利用方法を以下に記述する．

6.5.1 メディアサービスの定義

メディアサービス定義画面（図 11）は SVG ベースでインタラクティブなグラフ構造を表現できる JavaScript ライブラである JointJS [4] を用いて実装した．画面左のフィルタ，エンドポイント一覧からアイテムを選択すると，画面中央の領域にノードが現れ，ノードが保持する入力ポートと出力ポートをドラッグアンドドロップで接続できる．また，例えば顔を認識して任意の画像を重畳するようなフィルタの場合，フィルタの情報として画像の URL が必要であるが，そのようなパラメータは，画面右部の入力フォームで設定可能にした．また，6.4 節で説明したフィルタ間のイベント購読の定義もこの画面で追加できる．

6.5.2 メディアサービスの実行

図 12 に示す実行画面を操作することで，上述の定義したメディアサービスを，適用する現場（ゲートウェイ）ID を指定して実行できる．複数の現場を選択すると，同時に複数の現場へ同じメディアサービスを配備できる．実行中のメディアサービスはパイプライン一覧として表示されて，ボタン操作で開始・一時停止・停止・解放の操作を行える（図 13）．

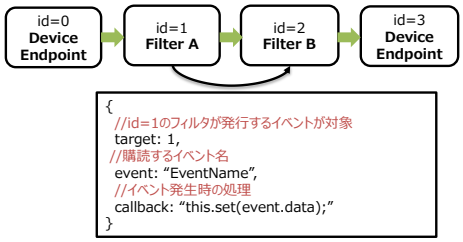

図 9: フィルタ間のイベント購読の定義

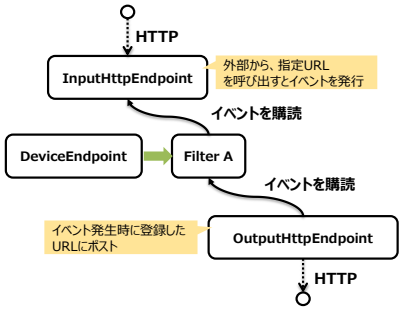

図 10: 外部と連携するための入出力エンドポイント

7. Proof-of-Concept 試作と考察

提案フレームワークの有効性を確認するために，以下の 3 つの IoT アプリケーションを実際に試作した．ここでは，これらの IoT アプリケーションについて説明し，各々の特徴について考察する．

7.1 試作 1: 講義支援

大学などの講義では，講師がパソコン内の講義資料をプロジェクタでスクリーンに投影し，説明することが多い．このような講義において，講義資料の特定部分を説明する場合には指示棒やレーザポインタを利用するが，大きな講義室などでは指定位置がわかりづらかったり見えなかったりする．そこで，スクリーン上で講師が指さした位置付近を拡大表示する講義支援アプリケーションを開発した．

映像の中からスクリーン領域を抽出し台形補正を行うフィルタ，映像の中から指先を検出し，その座標のイベントを発行する指検出フィルタ，指定座標付近を拡大表示する拡大フィルタを図 14 に示すように組み合わせることで実現した．教室にスクリーンを撮影するカメラを設置し，そのカメラから指の座標を取得する．一方で，講師の PC 画面をキャプチャした映像を送信するプログラムをデバイスとして登録しておき，PC 画面映像の指定座標周辺を拡大するフィルタを挿入し，それをスクリーンに投影する．これにより，講師がスクリーン上を指差すと，講義資料の一部が拡大表示され，聴講者は，講義コンテンツのどこに注目すればよいかわかりやすくなる．

メディアサービス実行者はクラウドにリクエストを送るだけで，実際の処理はデバイスを管理するゲートウェイ（教室に設置）で実行される．これにより，クラウドで実行す

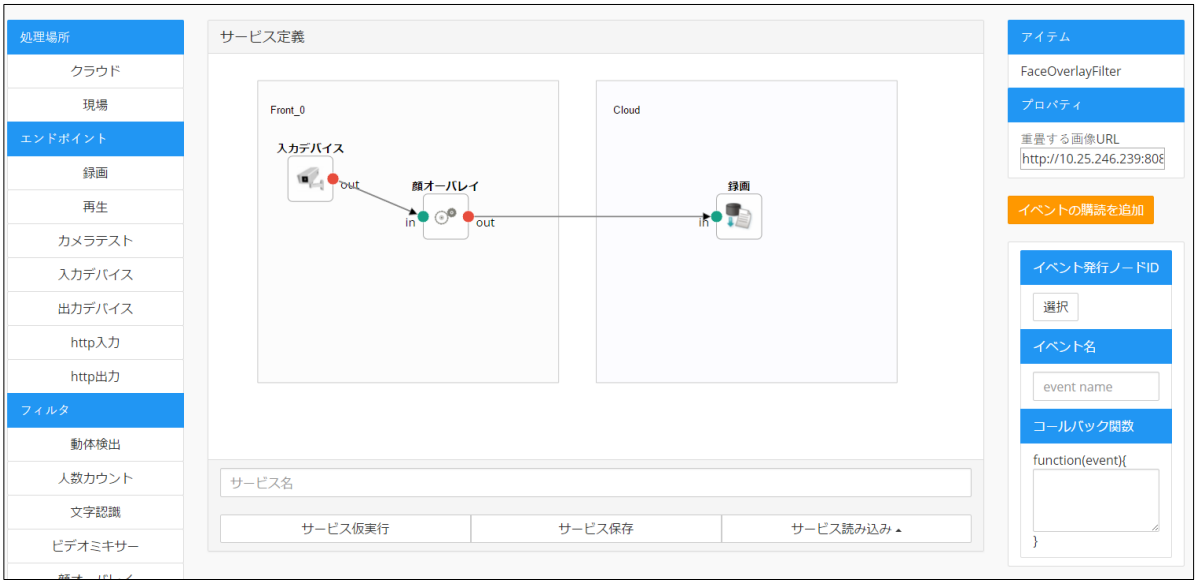

図 11: GUI によるメディアサービス定義

図 12：メディアサービス実行画面

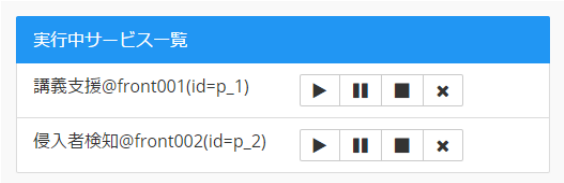

図 13：実行中メディアサービス一覧画面

る場合に比べて，手の動きに対する追従性が向上する．また，拡大フィルタの部分だけマーカー表示フィルタに置き換える等，クラウドで保持するメディアサービス定義の記述を修正するだけで，現場で実行される処理が変更される．

7.2 試作 2: 不審者監視

ネットワークカメラを使って，建物内の不審者侵入検知等を行う監視サービスが広く使われている．従来のクラウド型監視サービスは，映像データをネットワーク経由で全てクラウドに集約して蓄積するため，使用するネットワーク帯域，蓄積容量が大きいことが課題であった．そこで，現場で動きがあったときのみ，クラウドへ映像を送り録画する IoT アプリケーションを開発した．

背景差分法を使って映像から動体領域を検出する動体検出フィルタと，入力映像を出力するかどうかを制御できるスイッチフィルタを図 15 に示すように組み合わせることで実現した．ゲートウェイで動体検出処理をすることで，現場で動きがあったときの映像のみがクラウドへ送られるため，ネットワーク帯域，クラウドのストレージ容量を節約できる．

7.3 試作 3: 熱中症対策

夏場の屋外の作業現場では，作業者が高温多湿な環境下で肉体労働することで，熱中症を発症する危険性がある．熱中症を予防するため，WBGT 値 [10] と呼ばれる暑さ指数に応じた連続作業時間の制限をすることがあるが，監督者が作業現場の WBGT 値と全作業者の健康状態を常に把握することは難しい．そこで，作業現場の WBGT 値が一定以上の場合に映像を録画すると共に，作業員に動きがなければ異常と判断して監督者に通報する作業監視アプリケーションを開発した（図 16）．

ここでは，時系列データからリアルタイムにイベントを検出可能な既存の IoT プラットフォームを用いる．現場に設置したセンサから温度，湿度を定期的に取得し，WBGT 値を計算し，IoT プラットフォームにデータ登録する．その際，IoT プラットフォームの持つイベント機能を利用し，WBGT がある値以上になったとき，HTTP 入力エンドポイントが公開する API を呼ぶ．これにより，WBGT 値の変化に応じて，録画を開始したり，動きを検出したり，メディアサービスの動作を制御できる．また現場での動体検出結果を，HTTP 出力エンドポイントを使って IoT プラットフォームへ通知し，IoT プラットフォームがこのデータをもとに監視員や作業者にアラートをあげる．

このように，メディアサービス定義の中で，HTTP 入力，出力エンドポイントを用いることで，既存の IoT 系プラットフォームとの間でイベント情報をやりとりするような IoT アプリケーションも開発可能である．

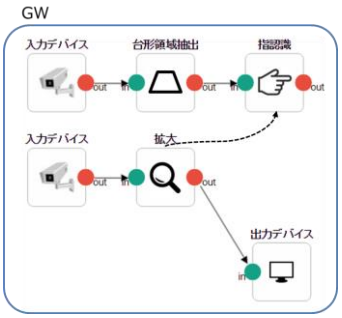

図 14：【試作 1】講義支援アプリケーション
におけるメディアサービス定義

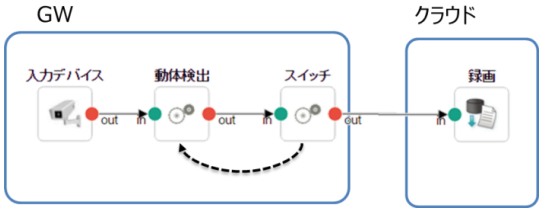

図 15：【試作 2】不審者監視アプリケーション
におけるメディアサービス定義

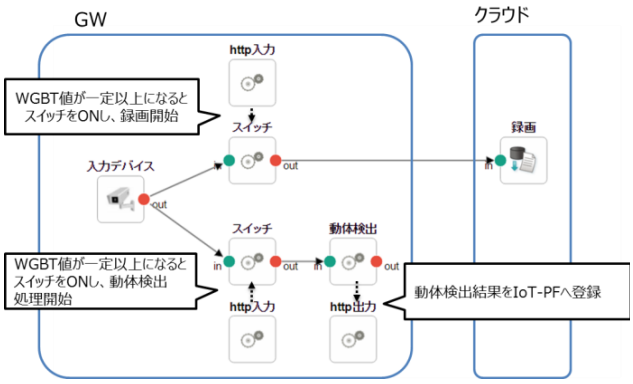

図 16：【試作 3】熱中症対策アプリケーション
におけるメディアサービス定義

表 3: 提案フレームワーク利用

	あり	なし
試作 1	38	50
試作 2	26	42
試作 3	50	48

8. 評価

7 章での IoT アプリケーション試作に基づき，提案フレームワークを以下の 5 つの観点での評価を行った．

8.1 複雑さの排除

一つのメディアストリームを複数場所で連携して処理するためには，本来必要なメディア処理以外に，RTP や WebRTC を用いて，処理場所間のメディアセッションを確立する必要がある．また，メディア入出力デバイス毎に異なるエンドポイントの用意と，異なるセッション確立処理が必要にある．今回の提案フレームワークを用いた場合，開発者はそのような煩わしさから解放され，現場に設置された入出力デバイスとメディア処理をつなぐという本質だけに専念してシンプルに IoT アプリケーションを開発できるようになる．

8.2 生産性の向上

上記の試作アプリケーションを，提案フレームワークを用いた場合と用いなかった場合でプログラム行数を比較した結果を表 3 に示す．提案フレームワークを用いた場合は JSON 形式で記述したもの，用いない場合は JavaScript プログラムで記述したもののうち，括弧などの除いた行数をカウントした．試作 2 で定義したようなメディアサービスの場合，ゲートウェイ・クラウド間のメディアセッション確立などを記述しなくてもよくなるため，プログラム行数が大きく削減できていることがわかる．

一方で試作 3 のように，センサ系のイベントに応じてメディアストリームの処理パイプラインが変化するような IoT アプリケーションの場合，提案フレームワークを用いた場合はスイッチ機能を持つフィルタを使い静的な一つのメディアサービスとして表現しているが，用いない場合は，IoT アプリケーション側のロジックとして条件を記述できるので，デバイスやゲートウェイとのセッション確立処理を加えても行数が少なくなっている．このことから，現在のメディアサービス定義のフォーマットは，動的にメディア処理パイプラインが変化するようなメディアサービスの記述に適しているとはいえない．今後，イベント駆動に適したメディアサービス定義の記述方法の検討が必要である．

8.3 試行錯誤の容易性

IoT アプリケーション開発においては試行錯誤が重要である．通常メディアを利用したアプリケーションは，プログラム上で，メディア入出力を制御する部分と，メディア処理自体が密に結合していることが多いため，処理内容に少し変更を加える場合，アプリケーションのプログラム自体を修正する必要がある．例えば試作 1 において，指の座標にマーカーを表示するような IoT アプリケーションに変更することを考えると，提案フレームワークを用いた場合は，メディアサービス定義中の拡大表示のブロックをマーカー表示のブロックに置き換えるだけでよく，IoT アプリケーション自体を配備し直す必要がない．

また，メディアサービスを配備するためには，クラウド側の存在しか意識する必要がないので，ゲートウェイとクラウドそれぞれでメディアサービスを更新して，両者のセッションを確立する作業を省けるため，試行錯誤のサイクルがより早く回せることが期待できる．

8.4 デバイス設置者と IoT アプリケーション開発者の分業

例えば，試作 1 のような IoT アプリケーションを，Kurento

Media Server だけを利用した場合，利用するデバイスの仕様に応じてアプリケーションを開発するため，適用する教室によってカメラの仕様が異なると，同じ IoT アプリケーションを適用することができない．一方，提案フレームワークを用いた場合，デバイス毎のセッション確立や操作の処理が抽象化されているため，同じ名前でカメラが登録されてさえいれば，全ての現場に適用できる．この仕組みによりデバイスの設置作業と独立して，IoT アプリケーションの開発が可能となる．

また，カメラやマイクだけでなく，抽象化したデバイスのインタフェースに則って実装したソフトウェアモジュールをメディア入出力デバイスとして登録可能なので，試作 1 のように，カメラ映像も PC 画面をキャプチャした映像もフレームワーク上では同等に扱える．

8.5 外部システムとの連携容易性

試作 3 のように，デバイスからデバイスへのメディアストリームを認識・加工を行うような IoT アプリケーションだけでなく，HTTP の入出力エンドポイントを定義可能にしていることで，メディアサービスを制御するための API を外部に公開することや，内部で起きたイベントを外部 URL へ通知することができる．今回の試作 3 では，既存の IoT プラットフォームとの連携を行ったが，他にも例えば，CEP（複合イベント処理）基盤などと連携し，映像からのイベントとセンサからのイベントを組み合わせた高度なイベント検出などに用いることもできる．

このように提案フレームワークは，既存の IoT アプリケーション開発のためのプラットフォームにとって代わるものではなく，既存の IoT プラットフォームで，メディアを利用しやすくするための拡張としても利用可能である．現段階では HTTP を用いた連携のみ定義可能となっているが，WebSocket や MQTT 等の様々なプロトコルの入出力に対応したエンドポイントを実装することで，センサストリーム系とメディアストリーム系を連携した IoT アプリケーションをより簡単に開発可能になる．

9. まとめ

IoT アプリケーション開発において，現場に設置されたカメラやマイクから取得される映像や音声などのメディアデータを扱いやすくするためのフレームワークの設計・実装を行った．提案フレームワークは以下の特長を持つ．

- メディア入出力デバイスの抽象化
- ゲートウェイとクラウドでの分散実行
- GUI によるフロー定義で開発可能

また，実装したフレームワークの有効性を検証するために，実際にフレームワークを利用して IoT アプリケーションの試作を行った結果，IoT アプリケーション開発にかかる生産性，試行錯誤の容易性などの観点での有効性が確認できた．

今後の課題として，イベント駆動なメディアサービス定義の記述方法の検討，外部システムとの連携を容易にするためのエンドポイントを追加することなどが挙げられる．また，提案フレームワークを用いることで，実際に試作や試行錯誤が行いやすくなるのかを客観的に実証するために，IoT アプリケーション開発ワークショップやハッカソンなどに提供して利用データを収集し検証を行う予定である．

参考文献

[1] GStreamer: Open Source Multimedia Framework: <https://gstreamer.freedesktop.org/>
[2] IBM Bluemix: <https://www.ibm.com/developerworks/cloud/bluemix/>
[3] IoT クラウドプラットフォーム Kii: <https://jp.kii.com/>
[4] JointJS - the HTML 5 JavaScript diagramming library: http://www.jointjs.com/
[5] Kurento: Open Source Software WebRTC Media Server: <https://www.kurento.org/>
[6] myThings - あなたの毎日が，組み合わせで便利になる: <http://mythings.yahoo.co.jp/>
[7] Node-RED: <http://nodered.org/>
[8] RFC 4566 - SDP: Session Description Protocol: http://tools.ietf.org/html/rfc4566.html
[9] Skylink - WebRTC peer introduction, audio, video, embeddable real-time communication: < http://skylink.io/>
[10] 環境省熱中症予防情報サイト: <http://www.wbgt.env.go.jp/wbgt.php>
[11] Akpinar, K., Hua, K. A., and Li, K., “ThingStore: A Platform of Internet-of-Things Application Development and Deployment,” ACM International Conference on Distributed Event-Based Systems (ACM DEBS 2015), pp.162-173, 2015.
[12] Alam, S., Chowdhury, M. M., and Noll, J., “SenaaS: An Event-Driven Sensor Virtualization Approach for Internet of Things Cloud,” IEEE International Conference on Networked Embedded Systems for Enterprise Applications (IEEE NESEA 2010), pp. 1-6, 2010.
[13] Blackstock M., and Lea, R., “Toward a Distributed Data Flow Platform for the Web of Things (Distributed Node-RED),” International Workshop on Web of Things (WoT 2014), pp.34-39, 2014.
[14] Fernandez, L., Diaz, M. P., Mejias, R. B., and Lopez, F. J., "Kurento: A Media Server Technology for Convergent WWW/Mobile Real-Time Multimedia Communications Supporting WebRTC," IEEE International Symposium and Workshops on World of Wireless, Mobile and Multimedia Networks (IEEE WoWMoM 2013), pp. 1-6, 2013.
[15] Munjin, D. and Morin, J. H., "Toward Internet of Things Application Markets." Proc. IEEE International Conference on Green Computing and Communication (IEEE GreenCom, 2012), pp.156-162, 2012.