複数のストリーム処理基盤を連携可能なサービス開発・実行システムの設計と実装

笠波昌昭† 小牧大治郎† 山口俊輔† 篠原昌子† 堀尾健一† 村上雅彦†

概要：デジタルビジネスではサービスの要件があらかじめ明確でないことが多く，サービス開発者には現場でのサービス開発・改良の迅速な試行錯誤が求められる．こうしたサービス開発者を支援するため，現場で発生するセンサデータやメディアのようなストリームデータを扱うサービスを容易に開発できるストリーム処理基盤が多数存在するが，個々のストリーム処理基盤だけでは扱える処理の種類に限りがあり，また複数のストリーム処理基盤を利用するには連携処理の開発が必要という課題があった．そこで我々は，複数の既存ストリーム処理基盤を自動的に連携することで多様な処理を組み合わせたサービスを開発，実行可能なサービス開発・実行システムを設計，実装した．これによりサービス開発者は，連携処理を独自開発することなく容易にサービスを作成できる．連携にあたっては既存ストリーム処理基盤がもつデータ入出力手段を流用するため改造が不要であり，処理基盤の新規追加や更新対応が容易になる．本稿では提案システムのプロトタイプ実装とサービスの試作を通じ，提案システムの利点と課題について考察を行った．

Design and Implementation of a Service Development and Execution System by Integrating Multiple Stream Processing Platforms

Masaaki Kasanami† Daijiro Komaki† Shunsuke Yamaguchi†
Masako Shinohara† Kenichi Horio† Masahiko Murakami†

1. はじめに

IoT などの ICT 技術によって既存ビジネスをデジタル化し，新たな価値を生み出すサービスを提供するデジタルビジネスへの注目が集まっている．従来のウォーターフォール型システム開発とは異なり，デジタルビジネスではサービスの要件があらかじめ明確でないため，実証実験などによりビジネスの現場で何度も試しながら，サービスが生み出す価値やサービスに必要な機能や性能などを探る必要がある．一方でビジネスの現場には人が存在し，現場のデバイスや発生するデータと作用し合うため，これらが協調してリアルタイムに動作するサービスを開発することが求められる．そのため，デジタルビジネスにおけるサービス開発者は現場の多様なデバイスから連続的に発生するセンサデータ，メディアなどのストリームデータをリアルタイムに処理するサービスを開発することと，サービスを検証，修正する試行錯誤のサイクルを現場で迅速に行うことが求められる．

このようなサービス開発者を支援するため，ストリームデータを扱うサービスを容易に作成・実行可能なストリーム処理基盤（以下，処理基盤）が多数存在している．これらの処理基盤では，センサやスマート照明などの IoT デバイスとの連携処理，映像や音声に対するメディア処理，分散実行による大規模データ処理など，様々なデータを多様に処理できる．

しかしこれらの処理基盤はそれぞれ扱える処理の種類に限りがあるため，単一の処理基盤では，様々なデータや処理を組み合わせて同時に利用するサービスを作成できない可能性が高い．一方，複数の処理基盤を用いて作成するには，利用する処理基盤の間でデータ受渡しを行う連携処理を独自開発する必要があり，迅速にサービス開発できない．例えば，ホテルの玄関に設置されたカメラで入館者を識別し，入館者の予約情報等を検索してフロントに速やかに伝達するサービスの場合，カメラ映像から入館者の顔をリアルタイムに識別するメディア処理，識別した入館者を顧客データベースに照会するデータ処理，さらに照会した予約情報をフロントのスピーカやディスプレイに通知して動作させるデバイス連携処理が必要になるが，この 3 処理を同時に扱える処理基盤を筆者らは見つけられなかった．

そこで我々は，複数の処理基盤を自動的に連携することで，様々なデータや処理を組み合わせたサービスを容易に開発，実行できるサービス開発・実行システムの設計と実装を行った．提案システムは既存の処理基盤が扱える処理を利用可能とし，サービスにおいて異なる処理基盤が扱う処理を接続する箇所に，データ受渡しを行う連携処理を自動挿入する．これによりサービス開発者は，連携処理を独自に開発せずとも複数の処理基盤を用いたサービスを開発できる．このとき，連携処理には各処理基盤が本来備えている，連携システム（システム間でのデータ受渡しを制御するシステム）へのデータ入出力手段を流用する．これにより提案システムでは，処理基盤に手を加えずに新規追加

† 株式会社富士通研究所
FUJITSU LABORATORIES LTD.

や更新対応が可能になると同時に，サービス開発者が新たな処理基盤で扱える処理や既存の処理基盤に追加された処理を利用できる機会が増すという利点ももたらす．また提案システムは，利用可能な処理を処理部品として配置，接続，設定可能な GUI の開発画面を提供することで，予備知識のないサービス開発者でも様々なデータや処理を同時に利用するサービスを容易に開発できる．

以下ではまず 2 章で関連研究を紹介し，3 章で提案システムのアーキテクチャについて説明する．4 章で実装したプロトタイプについて説明し，プロトタイプで試作したサービス例を紹介する．5 章ではプロトタイプを通じて判明した提案システムの利点と課題について考察する．最後に 6 章でまとめと今後の課題を述べる．

2. 関連研究

2.1 ストリーム処理基盤

処理の組合せでサービスを作成可能な Node-RED[1]はオープンソースのフローベース開発ツールであり，センサから出力されるストリームデータを扱う処理や，SNS や電子メール，データベースなど外部サービスに接続する処理などの様々な処理を「ノード」と呼ばれる処理部品として提供する．サービス開発者は Node-RED の開発画面上でノードを配置，パラメータ設定し，相互に接続することで，サービスを容易に作成，実行できる．また，有志により開発された約 2500 個の OSS ノードが公式サイトに登録されており，サービス開発者はこれら多種多様なノードを利用して幅広いサービスを作成できる．

Apache Nifi[2]はシステム間のデータの流れを定義・制御可能なオープンソースソフトウェアであり，Node-RED 同様，開発画面上でデータ処理を組み合わせたサービスを作成・実行できる他，サービスの分散実行や，データフローのバックプレッシャ制御，通信量の監視もできる．

筆者ら[3]は，映像や音声などのメディアに対する処理部品を多数提供し，開発画面上でこれらの処理部品を配置，接続するだけでサービスを容易に開発，実行できるメディア処理基盤を提供している．サービス開発者は GUI で処理部品を繋ぐだけでよく，コーデックやプロトコル，画像処理などの専門知識がなくてもメディアを活用するサービスを作成できる．加えてこの処理基盤は，ゲートウェイとクラウドでサービスを分散実行でき，ネットワーク帯域を節約できる．

このように既存の処理基盤ではデータ処理，分散処理，メディア処理などの多様なデータや処理を扱えるが，これらの処理を同時に扱えるものは存在しない．

2.2 処理基盤やサービスを連携する技術

鎌田ら[4]は，複数のライフログサービスの集約・連携による付加価値の高いサービスの実現を目指し，ライフログのための標準データモデルと，既存のライフログサービスを横断してデータ検索・取得可能なマッシュアップ API を提案している．提案 API を利用してサービスを開発した場合，既存サービスの個別 API を利用した場合に比べ，開発規模が減少することを明らかにした．既存のサービスを連携する際の開発工数削減を目指す点は本研究と類似しているが，サービス作成に GUI を用いない点，既存サービスの新規追加にかかる工数について議論していない点で異なる．

吉田ら[5]は，既存の Web サービスを組み合わせて新たな IT システムを構築可能なサービス連携プラットフォーム ΣServ を開発している．ΣServ は，主に企業システムとしての利用を想定し，GUI でのフロー記述エディタ，トランザクション制御，アプリケーション実行状態保存機能をもつ．ΣServ は XML や CSV などのデータを処理対象としており，本研究が扱うストリームデータは対象外である．

StreamAnalytix[6]は，様々な処理部品を GUI 上で配置，接続，設定することで複数の処理基盤を組み合わせたリアルタイムデータ解析サービスを作成可能なプラットフォームである．StreamAnalytix ではビッグデータ処理や分散処理を得意とする処理基盤しか利用できないため，メディア処理やデバイス連携処理などの多様な処理を組み合わせたサービスは作成できない．

3. アーキテクチャ

提案システムは，複数の既存の処理基盤を利用可能とし，これらが扱う処理部品を開発画面（サービスビルダ）上で組み合わせてサービスを定義可能な GUI を提供する．また，サービス定義において異なる処理基盤の処理部品を接続する箇所に，処理基盤間でデータを受渡しする連携システムを決定し，これを使用するための情報を格納した連携処理部品を挿入する連携制御機能と，サービス定義を各処理基盤ごとに分解，書式変換し，各処理基盤を制御する実行制御機能をもつ．

ここで，提案システムが扱う処理基盤は以下の 2 つの特徴をもつと規定する．

- 処理部品の組み合わせでサービスを作成する
- 外部へのデータ入出力手段を有する

また，このような既存処理基盤の多くは停止中，実行中の 2 状態をもち，一部は一時停止中もサポートしている．デジタルビジネスでは外部システムとの連携が多く，提案システムのメンテナンスや障害に備え，連携に関する内部パラメータを保持したまま動作を停止する一時停止中も重要と考えられるため，処理基盤は停止中，実行中，一時停止中の 3 状態をもつとする．一時停止中をもたない処理基盤に対しては，内部パラメータを保存し停止することで仮想的な一時停止中とする．

提案システムのアーキテクチャを図 1 に示す．以下では各コンポートネントの機能について述べる．

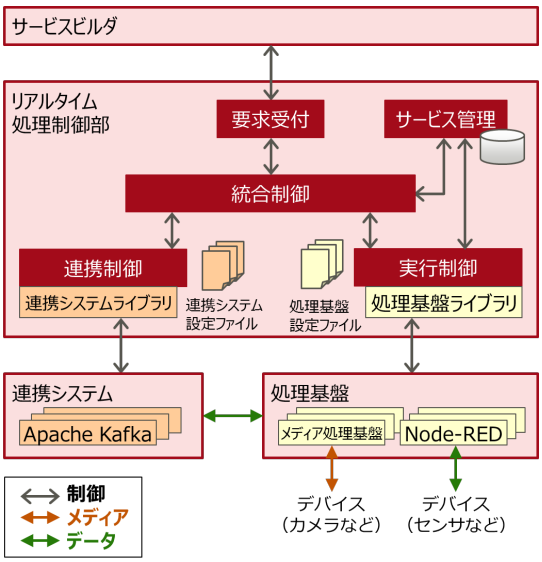

図 1　提案システムのアーキテクチャ

Figure 1　Architecture of proposed system.

3.1　サービスビルダ

サービスビルダでは，各処理基盤で利用可能な処理部品を統一的に配置，接続，設定可能とする GUI を提供し，サービス開発者が処理部品の処理順序を記述して，様々なサービスを定義可能にする．またサービスビルダでは，サービスに対する操作を受け付け，処理する．具体的にはサービスの登録では，作成したサービス定義を【登録要求】としてリアルタイム処理制御部の「要求受付」へ送り，要求成功時にはサービスの識別子（以下，サービス ID）を受け取る．またサービスの開始や停止など，サービスの動作に対する制御では，【登録要求】で取得したサービス ID と共に【制御要求】として「要求受付」へ送り，結果を受け取る．

3.2　リアルタイム処理制御部

- 要求受付

  「サービスビルダ」から受け付けた【登録要求】や【制御要求】を「統合制御」へ送り，「統合制御」から受け取った結果を「サービスビルダ」に返す．

- 統合制御

  【登録要求】を受け取った場合，サービス ID として一意の識別子を生成し，要求に含まれるサービス定義を「連携制御」に送って加工させる．サービス定義（加工前，加工後）をサービス ID と共に「サービス管理」に保存，サービス ID を「要求受付」に返す．

  【制御要求】を受け取った場合，要求に含まれるサービス ID とサービスに対する制御内容を「実行制御」に送って処理基盤を制御し，「実行制御」から受け取った結果を「要求受付」に返す．

- 連携制御

  処理の流れを図 2 に示す．まず「統合制御」からサービス定義を受け取ると，サービス定義内から異なる処理基盤の処理部品が接続された箇所（連携箇所）を抽出する．次に手順 1.から 4.を連携箇所ごとに行う．

  1. 連携箇所両端の処理基盤の「処理基盤設定ファイル」をそれぞれ参照し，両者が利用可能な連携システムを 1 つ選択する．
  2. 選択した連携システムの「連携システム設定ファイル」から連携システムを使用するために必要な情報（システムが稼動するサーバの URL など）を取得する．
  3. 「連携システムライブラリ」を用いて連携識別子を決定する．
  4. 上記の情報を格納した連携処理部品を 2 つ生成し，連携箇所を切断してそれぞれに連携処理部品を接続する．

  上記を全ての連携箇所に対して実行後，加工済サービス定義を「統合制御」に返却する．

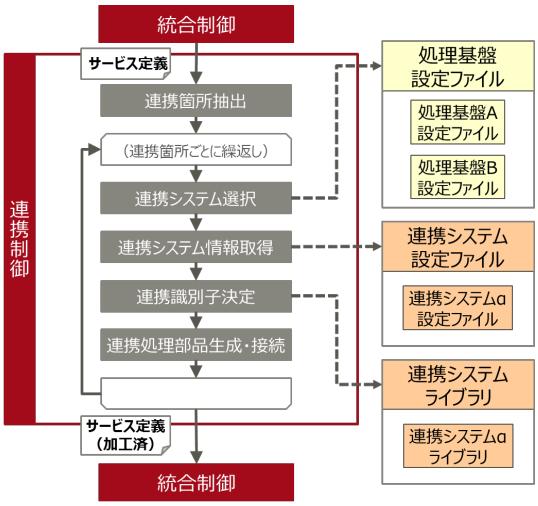

図 2　連携制御の流れ

Figure 2　Flow of platform integration.

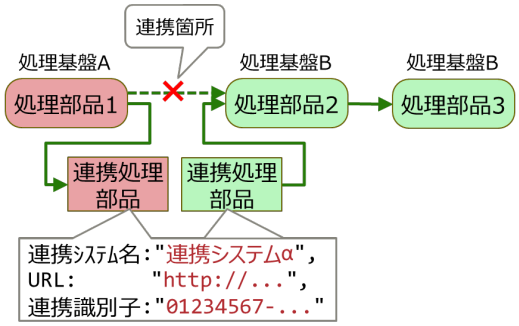

図 3　連携制御の例

Figure 3　Example of platform integration.

例えば図 3 では，処理基盤 A の処理部品 1 と処理基盤 B の処理部品 2 の間が連携箇所となる．そして処理基盤 A と B の処理基盤ライブラリを参照し，両者

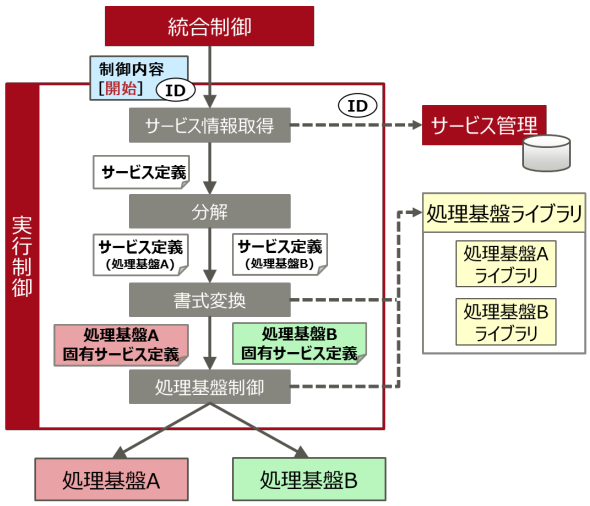

図 4　開始時の実行制御の流れ

Figure 4　Flow of platform control module when service starts.

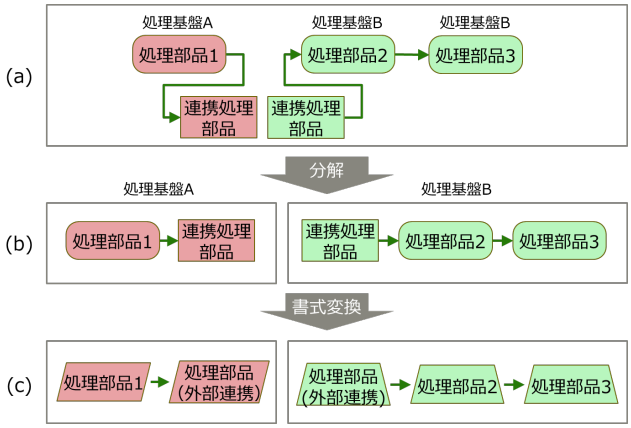

図 5　サービス定義の分解と変換

Figure 5　Division and translation of service definition.

が使用可能な連携システムαを選択する．次に，連携システムα設定ファイルからサーバ URL を取得，連携システムαライブラリを用いて連携識別子を決定する．最後に，これらの情報を格納した連携処理部品を 2 つ生成し，連携箇所を切断したのちそれぞれ接続する．

- 実行制御

「統合制御」から受け取ったサービス ID とサービスに対する制御内容に基づいて処理基盤を制御する．まず制御内容として開始を受け取った場合の手順を図 4 に示す．

1. 「サービス管理」からサービス ID に紐付いたサービス定義（「連携制御」で加工済）を取得する（図 5 (a)）．
2. サービス定義を処理基盤ごとに分解する（図 5 (b)）．
3. 分解したサービス定義を，「処理基盤ライブラリ」を用いて各処理基盤固有の書式（形式やパラメータ名）に変換する（図 5 (c)）．
4. 変換したサービス定義を「処理基盤ライブラリ」を用いて各処理基盤に登録して開始し，成功した場合は制御識別子を取得する．そして全ての処理基盤で開始が成功した場合のみ，成功の結果を「統合制御」に返却するとともに，各処理基盤の制御識別子と実行状態をサービス ID と紐づけて「サービス管理」に保存する．

また，制御内容として停止，一時停止，再開を受け取った場合，以下の手順を実行する．

1. 「サービス管理」からサービス ID に紐付いた処理基盤の制御識別子を取得する．
2. 「処理基盤ライブラリ」の制御内容に相当する関数に制御識別子を渡し，各処理基盤を制御させる．
3. 全ての処理基盤で制御が成功した場合のみ，成功の結果を「統合制御」に返却し，「サービス管理」の実行状態を変更する．

- サービス管理

サービス定義とサービス ID，および各処理基盤の制御用の情報をまとめて管理する．

3.3 処理基盤設定ファイル，ライブラリ

提案システムが利用する処理基盤ごとに存在する．処理基盤設定ファイルには，処理基盤の制御に必要な情報（処理基盤が稼動しているサーバの URL など）や利用可能な連携システムの一覧が記述されており，「連携制御」や「実行制御」が参照して必要な情報を取得する．

一方，処理基盤ライブラリにはサービス定義を処理基盤に固有の書式へ変換する処理と，処理基盤を制御する処理が記述されており，「実行制御」でのサービス定義の書式変換と処理基盤上の動作を制御する際に利用する．ここで，処理基盤の制御処理は，処理基盤がもつ 3 状態を遷移するため，開始，停止，一時停止，再開の 4 関数をもつ．

3.4 連携システム設定ファイル，ライブラリ

提案システムが利用する連携システムごとに存在する．連携システム設定ファイルには連携システムの制御に必要な情報（URL など）が，連携システムライブラリには連携システムを利用する際の処理（連携識別子の決定処理）がそれぞれ記述されており，「連携制御」が連携システムを制御する際に利用する．

4. 実装

3 章に基づいてサービス開発システムのプロトタイプを実装した．本プロトタイプでは，処理基盤に Node-RED[1]と筆者らが提案したメディア処理基盤[3]を，連携システムに Apache Kafka[7]を利用した．

4.1 サービスビルダ

プロトタイプでは Node-RED 開発画面を改修したものを

表 1　リアルタイム処理制御部の Web API

Table 1　Web API of the real time processing control module.

URI	メソッド	用途
/flows	GET	サービス一覧を取得
/flows	POST	サービスを登録し，サービス ID を取得
/flows	DELETE	サービスを消去
/flows/:id	GET	指定したサービス ID（id）のサービスを取得
/flows/newest	GET	最新のサービスを取得
/flows/:id/:command	PUT	指定した id のサービス実行状態を変更(“start”, “stop”, “pause”, “resume”)

表 2　処理基盤ライブラリが備える関数

Table 2　Functions provided in processing platform libraries.

関数	用途
convertFlow(flow)	サービス定義（flow）を処理基盤の固有書式に変換
start(flow, callback)	flow を処理基盤に登録し開始．callback で制御識別子を取得
stop(id, callback)	制御識別子（id）に該当するサービスを停止
pause(id, callback)	id に該当するサービスを一時停止
resume(id, callback)	id に該当するサービスを再開

サービスビルダとして利用した（図 6）．これは，Node-RED の使用方法が直観的でわかりやすく，またドキュメントやノウハウが数多く公開されているため，プログラミングや処理基盤の利用に習熟していない開発者でも扱いやすいからである．なお Node-RED 自体の変更を容易に取り込めるよう，改修点は以下の 5 点に留めた．

- サービス定義の送信先を Node-RED サーバからリアルタイム処理制御部に変更
- リアルタイム処理制御部から受け取るサービス ID をブラウザの Cookie に保存
- 保存したサービス ID を送信する要求に付加
- メディア処理基盤の処理部品表示名を日本語化
- サービスの停止（ストップボタン）を追加

また，サービスビルダ上でメディア処理基盤の処理部品を Node-RED の処理部品（ノード）と同様に表示するために，メディア処理基盤の処理部品と 1 対 1 に対応する「ダミーノード」を，Node-RED の自作ノード機能を利用して作成した．具体的には，ダミーノードは各処理部品と同一の名前，入出力端子数，パラメータ設定項目をもち，開発画面上で配置，接続，パラメータ設定が可能である．

4.2 リアルタイム制御部

プロトタイプでは, Express フレームワークを用いた Web サービスとして実装した. Node-RED 開発画面と Node-RED サーバ間の通信の一部を本制御部に横流しすることでサービスビルダとの通信を実現した．

- 要求受付
  サービス定義の送受信や，サービス実行状態の制御を受け付ける Web API を実装した. 一覧を表 1 に示す.
- サービス管理
  MongoDB を利用し，サービスごとに保存するデータスキームとして，サービス ID，サービス定義，実行状態，作成日時，処理基盤名とその制御識別子のペアを定義した．

4.3 処理基盤ライブラリ

処理基盤ライブラリが備えるべき関数を定義したテンプレートを作成し，これに基づいて Node-RED とメディア処理基盤の処理基盤ライブラリをそれぞれ実装した．その関数一覧を表 2 に示す．

- Node-RED ライブラリ
  サービスビルダに Node-RED 開発画面を利用したことで，サービス定義はそもそも Node-RED 固有書式であるため，convertFlow 関数には連携処理部品を OSS ノードの[node-red-contrib-kafka-node][8]へ変換する処理のみ実装した．また，Node-RED は開始に相当する機能しかもたないため，stop 関数には空のサービスを開始する処理を，pause，resume 関数には Node-RED のサービス定義に含まれる有効・無効フラグを切替えたのち再度開始する処理を実装した．
- メディア処理基盤ライブラリ
  convertFlow 関数には，サービス定義をメディア処理基盤の固有書式に変換する処理，サービス定義に含まれるダミーノードを対応するメディア処理基盤の処理部品に変換する処理，さらに連携処理部品をメディア処理基盤の「kafka からの入力」/「kafka からの出力」処理部品に変換する処理を実装した．また start，stop，pause，resume 関数には，各動作に該当するメディア処理基盤の API を呼び出す処理を実装した．

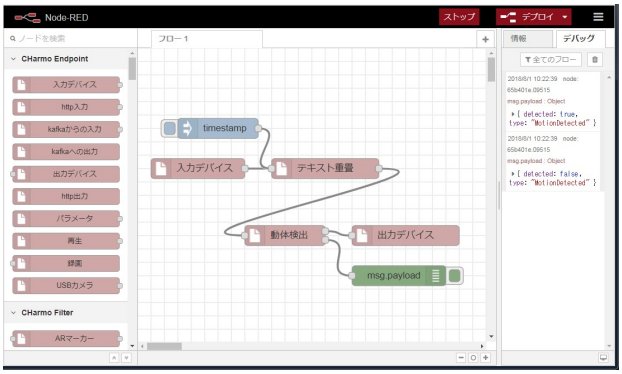

図 6　サービスビルダ
Figure 6　Service builder.

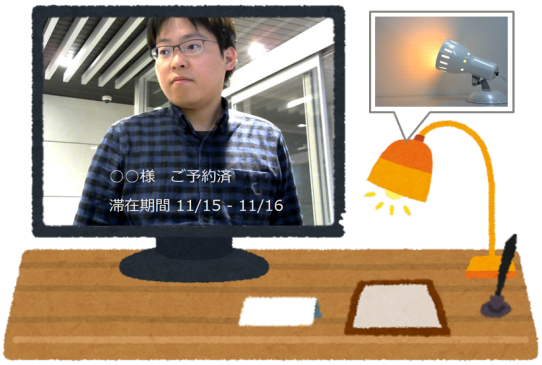

図 7　入館者通知サービスの動作イメージ
Figure 7　Operation image of visitor notification service.

4.4 連携システムライブラリ

連携システムライブラリが備えるべき関数を定義したテンプレートを作成し，これに基づいて Apache Kafka ライブラリを実装した．Apache Kafka ライブラリは連携識別子として Apache Kafka 内でのデータ区別に用いられる topic 名を決定する determineTopicId 関数をもつ. topic 名には UUID を利用した．

4.5 サービス例

プロトタイプを用いて試作した入館者通知サービスを紹介する．通常，ホテルへの入館者は予約の有無にかかわらずフロントにいる窓口担当者に対して自身の名前や予約内容等を伝えるが，予約済の入館者にとっては既に連絡した情報を再度伝える面倒がある. 入館者通知サービスでは，玄関に設置したカメラ映像から入館した人物をリアルタイムに識別し，照明を点滅させ入館者の存在を通知すると同時に，入館者情報を窓口担当者に先回りして伝える．このサービスにより，窓口担当者と入館者の不必要なやり取りを削減し，接客業務の品質向上や省力化が期待できる．サービスの動作イメージを図 7 に示す.

入館者通知サービスのサービス定義を図 8 に示す. 本サービスでは, デバイスとして, Web カメラとディスプレイ，ワイヤレスで制御可能なスマート照明 Hue[9]を使用する. またメディア処理基盤の処理部品として，Web カメラから映像を取り込む[Web カメラ]，映像から顔を識別し結果を出力する[顔認証]，映像に指定された文字を重畳する[テキスト重畳]，映像をディスプレイに表示する[ディスプレイ]を利用する．また Node-RED の処理部品として，Node-RED 標準ノードである[function]内にデータベース照会処理を記述した[DB 照会]，hue を制御する[hue 制御]を利用する.

本サービスを開始するとまず，[Web カメラ]が映像を[テキスト重畳]と[顔認証]へ送る．[顔認証]は受け取った映像内から入館者の顔の認識し，予約時に登録しておいた顔画像とマッチングした場合，入館者の ID を[DB 照会]へと送る．[DB 照会]は ID から入館者情報（氏名，予約情報）を取得して[テキスト重畳]と[hue 制御]へ送信する. [テキスト重畳]は受け取った映像上に DB から送られた入館者情報を重畳表示し[ディスプレイ]へ送信，ディスプレイ上に表示する．[hue 制御]は，入館者情報を受け取ったタイミングで hue を点滅させる.

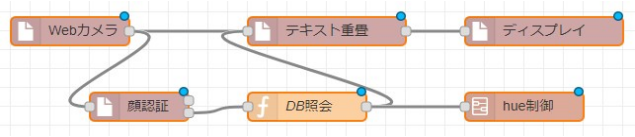

図 8　入館者通知サービスのサービス定義
Figure 8　Service Definition of visitor notification service.

5. 考察

実装したプロトタイプを通じて判明した提案システムの利点と課題について考察する.

5.1 利点

4.5 節で述べた通り, サービスビルダ上で Node-RED とメディア処理基盤を組み合わせたサービスを開発できた．従来，このようなサービスの作成には各処理基盤の開発画面を何度も往復しなければならなかったが，本プロトタイプを用いることでサービス開発者は多様なデータや処理を組み合わせたサービスを統一的な GUI で全体の見通しよく容易に開発できる.

また，従来は処理基盤間でデータ受渡しする連携処理を連携箇所の数だけ開発しなければならなかったが，本プロトタイプでは処理基盤間を自動的に連携するため開発不要である．実際にサービス例は処理部品の配置，接続と，最小限のコード記述（DB 照会処理と重畳テキストの定義）のみで試作でき，処理基盤間の連携処理は一切開発していない．このように，サービス開発者は連携に煩わされることなくサービスの開発に集中できる.

さらに，処理基盤の制御を処理基盤ライブラリによって抽象化しているため，提案システムへの処理基盤の新規追加，更新対応が容易になる．具体的には，新規追加の際は処理基盤ライブラリとして 5 つの関数さえ実装すればよく，

更新の際は必要に応じて処理基盤ライブラリを修正すれば対応できる．これによりシステム管理の作業負担が軽減され，サービス開発者が新たな機能を利用できる機会が増加すると考えられる．

5.2 課題

プロトタイプではサービスビルダに Node-RED 開発画面を利用したため，Node-RED の機能はそのまま使用できたが他の処理基盤がもつ機能が制限された．例えば，メディア処理基盤の開発画面が有する「処理部品の実行場所（クラウドかゲートウェイ）を指定できる機能」が利用できない．同様に Apache Nifi を導入した場合は「開発画面上で処理部品間の通信量を表示する機能」も利用できない．この問題を解決するため，連携する処理基盤の全機能を盛り込んだサービスビルダを開発する場合，操作が複雑化してサービス開発者に高い知識やノウハウが要求されるおそれや，提案システムへの処理基盤新規追加・更新作業の手間が増加して新機能の素早い提供が困難になるおそれがある．また複数の開発画面を利用可能にし，サービス開発者が自身の習熟度合いや活用する処理基盤の機能に応じて適したものを選択する場合，複数の開発画面のそれぞれに依存する機能を同時に活用できない問題がある．扱いやすさを維持しつつサービス開発者が利用できる機能数をできる限り減らさないよう，サービスビルダがどのような機能を有すべきか検討する必要がある．

また，本プロトタイプでは処理基盤が停止中，実行中，一時停止中の 3 状態をもつとして，処理基盤に対する制御を開始，停止，一時停止，再開の 4 関数に抽象化した．3 状態を遷移する関数としてこの 4 関数が適しているか，さらには処理基盤がもつ状態としてこの 3 状態が適しているかについて検討する必要がある．加えて，この 4 関数による抽象化では一部機能（Apache Nifi のバックプレッシャ制御や分散処理など）が利用できないため，新たに関数を加える必要があるかについても検討が必要である．

6. まとめ

本研究ではデジタルビジネスに携わるサービス開発者を支援するために，複数の既存ストリーム処理基盤を自動的に連携することで多様なデータや処理を組み合わせたサービスを GUI 上で容易に開発，実行できるサービス開発・実行システムを設計し，プロトタイプを実装した．

またプロタイプを用いたサービス試作を通じ，複数処理基盤を組み合わせたサービスを，連携処理を開発せず容易に作成できることを確認した．しかし，連携によって処理基盤固有の機能が一部利用不可能になる課題も判明した．

今後の課題として，プロトタイプに処理基盤を新規追加し，それに伴って生じる新たな課題を抽出すると同時に，サービスビルダが有すべき機能や，より適した処理基盤の抽象化方法の検討を予定している．

参考文献

[1] “Node-RED: Flow-based programming for the Internet of Things”, https://gstreamer.freedesktop.org/.
[2] “Apache Nifi: An easy to use, powerful, and reliable system to process and distribute data”, https://nifi.apache.org/.
[3]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: IoT アプリケーション開発のためのメディア処理・制御フレームワーク，マルチメディア，分散協調とモバイルシンポジウム 2016 論文集, vol. 2006, pp. 79-88 (2006).
[4]鎌田早織，坂本寛幸，井垣 宏，中村匡秀: マッシュアップ API を用いた異なるライフログサービスの連携，電子情報通信学会技術研究報告. LOIS ライフインテリジェンスとオフィス情報システム, vol. 109, No. 450, pp. 91-96 (2010).
[5]吉田英嗣，横山和俊，松田栄之: サービス連携プラットフォーム Σ Serv, NTT 技術ジャーナル, Vol. 15, No. 7, pp. 46-49 (2003).
[6] “StreamAnalytix: The Visual Big Data Analytics Platform for Your Real-time Enterprise”, https://www.streamanalytix.com/product/streamanalytix/.
[7]”Apache Kafka: A distributed streaming platform”, https://kafka.apache.org/
[8]”node-red-contrib-kafka-node: Node-RED nodes of HighLevel Kafka Producer and Consumer”, https://flows.nodered.org/node/node-red-contrib-kafka-node
[9]”Philips: Hue”, https://www2.meethue.com/ja-jp