MCMA(Media Cloud and Microservice Architecture) 標準化活動の紹介

日本放送協会　放送技術研究所 上級研究員　佐野（さの）　雅規（まさのり）

1. はじめに

インターネット技術の進化とスマートフォン（スマホ）など携帯端末の普及により、マルチメディアコンテンツの消費形態が大きく変わってきている。かつて、日常生活において映像を視聴する場といえば、家の中のテレビの前であったが、インターネット上で映像を流すことができるようになると、スマホやタブレットなど携帯端末を介して、いつでもどこでも映像コンテンツを楽しむことができるという状況が当たり前となりつつある。また、現在では、多くの人がスマホなどの端末を常時携帯しているため、アクセスする最初の情報はインターネット上の情報であることが普通になってきている。このような環境の変化に伴い、放送局でも、従来の番組制作に加え、ネットサービスに向けた様々なコンテンツを制作する必要が出てきており、今後その制作環境も大きく変わっていくものと考える。

近年の放送業界に起きた大きな技術の変化を挙げると、放送波がアナログからデジタルへ移行し、制作過程の多くの部分がファイルベースに移行し、現在はライブ信号のIP伝送化が徐々に始まっている。また、設備の面から考察すると、これまで専用ハードウェアで処理していた部分が、ソフトウェアにより置き換わってきている部分が多い。また、その処理を行う基盤は、自社でシステムを構築して運用するという、いわゆるオンプレミスという形態から、必要な時に必要なだけリソースを借用するクラウドを利用した基盤への移行が始まっているといえる。このように制作を支える技術基盤も緩やかに変化をしており、今後の制作環境を考える上では考慮しなければならない。

さらに、コンテンツの制作作業について考えてみる。今後は、多様なデバイスに対応するため、1つのコンテンツから複数のコンテンツを制作するなど、ワンソース・マルチアウトプットの考え方は基本になるであろう。さらに、ホームページなどウェブの世界へのコンテンツ以外にも、AR/VR用のコンテンツや、日常生活に徐々に入り込んできているスマートデバイスとの連携など、多種多様なコンテンツを、要求に応じて迅速に制作していくことが求められるであろう。これらの制作要求に対し、従来のように制作コンテンツごとに個別の専用制作システムを開発していくと、開発にコストと時間がかかる上、組織全体として俯瞰すると、重複するデータや処理が点在することになる。さらに、各システムが複雑に相互連携をすると、一部分の改修やメンテナンスであっても、その影響範囲の把握が難しくなり、総合的には運用コストが大幅に増大してしまう。一方、コンテンツ制作の作業を小さく分解してみると、素材は映像、音声、テキスト、その他のデータであり、これらを操作・加工する部分は、ある程度共通で利用できる部分が多い。そこで、今後のコンテンツ制作システムは、これら汎用的なメディア処理の組合せで実現するという、昔の表現であればSOA（Service Oriented Architecture）、最近の言葉ではMicroservice Architectureを採用した制作共通基盤が必須になると考える。

本稿では、この汎用的なメディア処理基盤を構築するための標準化の1つとして、現在、EBU（European Broadcasting Union）[1]の技術プロジェクトの1つとして進められているMCMAについて、その活動内容を紹介する。

2. 制作ワークフローに関する共通化

コンテンツ制作のワークフローを、様々なメディア処理を接続して構築するという考え方は古くからある。実はMCMAには前身の活動団体があり、それがFIMS（Framework for Interoperable Media Services）[2]にあたる。MCMAは、FIMSのコンセプトを引き継ぎ、かつ近年の技術とその環境に対応する方向で活動を進めている。ここでは、MCMAの目的を理解するために、FIMSのコンセプトと、MCMAへの移行の背景について述べる。

2.1　FIMSのスコープ

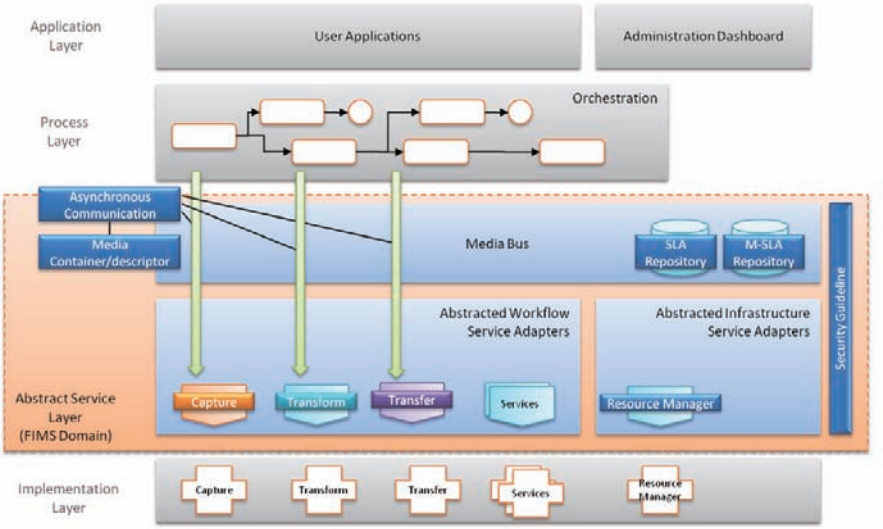
FIMSは、AMWA（Advanced Media Workflow Association）[3]とEBUが、柔軟で拡張性のある映像制作システムの実現を目指して、2009年12月に共同で開始したプロジェクトである（FIMS立ち上げまでの経緯については参考資料［4］参照）。それまでの制作システムの開発と導入について振り返ると、まずはじめに映像制作のワークフローがあり、これをサポートする形でシステムが開発される。通常、いろいろな制約があり、初めから一気通貫のシステ

ム構築は難しく、その時点でシステム化による効率の向上が見込まれる部分が、順次開発されシステムに置き換わっていく。また、新たな要求が生じると、既存システムにコンポーネントを追加する形で、次々と機能が追加され、各コンポーネント間は、独自の仕様でデータ連携を行う。このような状況が長い間経過すると、組織全体のシステムとしては、データやメディア処理の重複が点在し、かつ同じ情報であるが同期がとれていなかったり、あるコンポーネントを改修する場合に、その影響する範囲の把握が難しく、予想外の問題を引き起こすリスクが高くなるなど、全体としての運用コストが大幅に増加することになる。そこで、FIMSでは、SOAの考えを取り入れて、これらの問題を解決しようとした。

FIMSでは、メディア制作に必要な機能を提供する再利用可能な「メディアサービス」と、各種メディアサービスを組み合わせて制御する「メディアオーケストレーションシステム」により制作システムを構築するというコンセプトの元、このメディアサービスとオーケストレーションシステムの間のインタフェースについて標準化を行った。図1は、FIMSのアーキテクチャを示しており、上から順に、アプリケーションレイヤー、処理レイヤー、抽象化サービスレイヤー、実装レイヤーの4つのレイヤーで構成される。上述したメディアサービスは、図中の実装レイヤーにあたり、実際には各種メディア処理を行うソフトウェアプログラムである。また、メディアオーケストレーションは、図中の処理レイヤーにあたり、実際にはユーザーアプリケーションの中の一機能となる。この中で、抽象化サービスレイヤーにおける、各種メディアサービスの機能及びインタフェース仕様がFIMSの標準化部分となる。

2.2　FIMSの技術仕様

FIMSに関する情報は、AMWAが管理するウェブページ[2]に掲載があり、FIMSの解説をはじめ、FIMSに参加していた企業リストや、企業別のFIMS対応の度合い、またFIMS準拠の製品を利用しているユーザ側企業などの情報を取得することができる。ただし、技術的な仕様については、全てGithub[5]で公開されている。FIMSにはバージョンが存在し、表にそのバージョンとサポートしているメディ



■図1.　FIMSの参照アーキテクチャ

■表.　FIMS仕様のバージョンとメディアサービス

バージョン (策定年)	サポートするメディアサービス
1.0 (2012)	Capture, Transfer, Transform
1.1 (2014)	Capture, Transfer, Transform, Repository
1.2 (2015)	Capture, Transfer, Transform, Repository, Quality Analysis
1.3 (2017)	Capture, Transfer, Transform, Repository, Quality Analysis, Automatic Metadata Extraction

アサービスの種類を示す。FIMS仕様の初版では、Capture（収録）、Transfer（素材伝送）、Transform（変換）の3つのサービスについて、そのインタフェースなどを規定した。その後、メディアデータや関連データを蓄積管理するためのRepository（レポジトリ）、メディアデータの品質をチェックするためのQuality Analysis（QA）、メディアデータなどからメタデータを自動抽出するためのAutomatic Metadata Extraction（AME）サービスが追加され、最終バージョンは、Ver.1.3.1となっている。

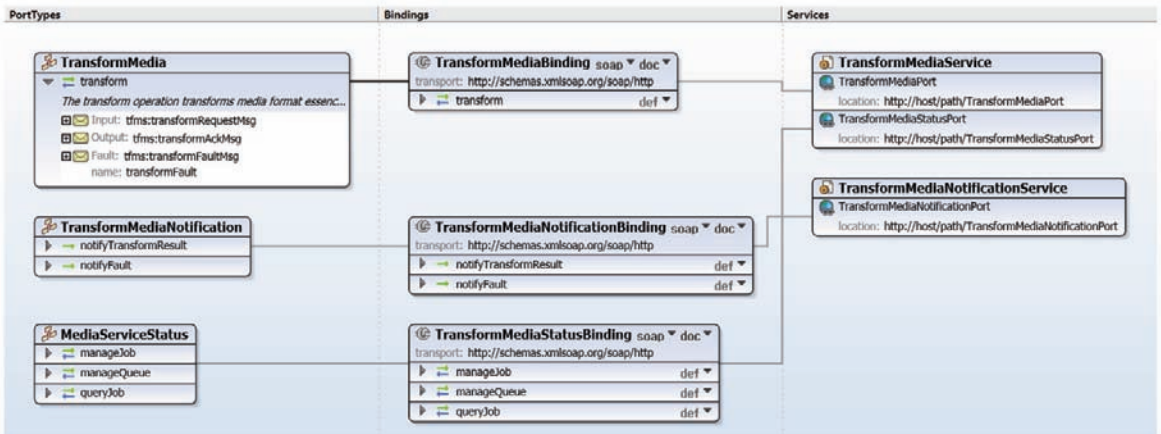
次に、FIMS Ver.1.3.1で規定されている項目について説明する。技術的なインタフェースは、上述したGithub[5]内の「WSDL-REST-XSD」というフォルダ内に、WSDL[6]とXSD[7]ファイルにより規定されている。WSDLファイルは、対象とするWebサービスを、人間もシステムも両方が理解できるようにXML形式[8]で記述したものであり、XSDファイルは、対象とするXMLデータの構造を、XMLスキーマ[9]を用いて記述したものである。Ver.1.3.1のインタフェースは、6つのWSDLファイルと24のXSDファイルで表現されている。したがって、この6つのWSDLファイルが、Ver.1.3でサポートされている6つのメディアサービスのインタフェースをそれぞれ規定している。24のXSDファイルは、そのインタフェースを介してやり取りする際のメッセージのデータ構造が定義されている。

図2に、Transformメディアサービスの WSDLを視覚化したものを記す。図中左側には、そのWebサービスを利用するための窓口となる「Port」と、そのPortを介してやり取りされるメッセージのフォーマットが抽象的に定義されている。まん中の列はそのPortと具体的な通信プロトコル及びネットワークアドレスを結びつける「Binding」が定義されており、右側には関連するPortが、必要があればまとめて「Service」として定義されている。理解促進のため、Java言語によるプログラミングと対比して説明すると、図中左上のTransformMediaというPort名は、Javaでいうところの Interface名に相当する。そのInterfaceの中に定義されたメソッドが、図中では、transformにあたり、そのメソッドのパラメータは、WSDLではメッセージと呼び、図中では、transformRequestMsg、transformActMsg、transformFaultMsgとなっている。これらメッセージの具体的なXML構造は、上述のXSDファイルに記述されている。そして、図中一番右側には、このTransformというFIMSのメディアサービスは、TransformMediaServiceとTransformMediaNotificationServiceの2つのサービスとして定義されていることがわかる。さらに、前者のサービスは2つのポートを、後者のサービスは1つのポートを、それぞれのURLで提供するよう定義している。

このようにWSDLには、サービスを提供するエンドポイントから、その中でやり取りするメッセージの構造まで、全てが明確に定義されているため、このWSDLを開発ツールに読み込ませることで、そのWebサービスを呼び出すためのモジュールコードが自動生成できるようになっている。FIMSのインタフェースは、このようなWSDLとXSDにより規定されており、これはFIMSのインタフェースが比較的素早く実装が行えるようになっていることを意味する。

2.3　MCMAの誕生

先にも述べたように、FIMSの策定は2009年末から始まっており、既に10年近くの時間がたっている。この間、いろいろなWeb技術の進歩と、その技術による環境の変化が起



■図2.　Transformメディアサービスの WSDL

こり、FIMS仕様もその影響を無視できなくなってきた。**図3**に、FIMSに関する年表を記す。FIMSは、2012年に、その仕様のVer.1.0を公表するとともに、IBC（International Broadcasting Convention）2012において展示を行い、Innovation Awardを受賞する。これを機に多くの企業が活動に参加し、FIMS仕様のアップデートが継続された。この頃より、ライブ映像信号のIP伝送に関する活動が始まり、多くのベンダーが徐々にそちらの方にリソースを移動させていったと聞いている。FIMS仕様のアップデートは少し期間をおいて、2017年7月に、Ver.1.3としてAMEサービスを追加し、それを利用したデモシステムをIBC2017で展示し、それを最後に活動は停止している状態である。

一方で、データを処理する基盤として、クラウドサービスを利用するという流れが徐々に広がりはじめ、メディア処理についてもクラウド上に移行する動きが出てきた。さらに、Webサービスについては、SOAPというプロトコルでなく、RESTと呼ばれるスタイルでの実装が主流となり、FIMSのSOAP仕様は利用されなくなってきていた。これらの変化に対応するため、2016年頃から、クラウド環境に向けた新しいFIMSの仕様を検討することが開始された。当時、この動きは、非公式にFIMS Ver.2.0と呼ばれ、2017年から2018年頃にかけて、AMWAがFIMSの活動から脱退するのを機に、MCMAと改名し、新しくEBU単独の技術プロジェクトとしてスタートした。これが現在のMCMAの成り立ちである。

2.4　MCMAのスコープと現状

現在のMCMAの活動は、EBUの技術セクション（TECHNOLOGY & INNOVATION）で策定した、Productionという戦略プログラム内の1つの技術トピック[10]として進められている。これまでのFIMSの活動を通して得たノウハウの上に、クラウド環境、サーバーレスアーキテクチャ、マイクロサービス、AIツールをキーワードに掲げ、最小限のパラメータ記述をもつ、メディア処理用の簡略化されたREST-APIセットの策定を目指している。また、REST-API仕様を策定するとともに、各種クラウドと共通化するAPIの間を実装するための共通ライブラリも開発・提供することを目的としている。MCMAのモットーは「サンプルによる実装例」となっており、技術仕様書のようなドキュメントは現在存在せず、コードから読み取ってほしい、としている。しかしながら、コードの存在だけでは不十分という声が多く、近い将来クラウドでのマイクロサービスアーキテクチャの実装について、ガイドラインを発行する予定としている。

実際のMCMAの活動は、毎週木曜日に開催されるWebexにより進められている。議長はFIMS時代からこの活動をリードしてきた、Bloomberg mediaのLoic BARBOU氏である。現在、このWebexへの参加者はあまり多くなく、Bloomberg、Triskel、Glookast、Cube-Tec、EBU、NHKに加え、時々、AWSやAzureのエンジニアなどが加わる程度である。先にも述べたように、まずはマルチクラウド上に実際のシステムを構築し、動作など検証を行った後に共通ルールを決める予定となっている。現在は、AWSのクラウドを基盤に、AWSとAzureのAIツールなどをワークフローにして呼び出すサンプルコードなどを開発し、Githubから公開している。EBUは、Githubに組織アカウント「EBU」を保持しており、本稿執筆時点で89のレポジトリを保有し、MCMAに関するレポジトリは2つ存在する。1つはmcma-libraries[11]、もう1つはmcma-projects[12]である。いずれにしても、現時点でのMCMAのアウトプットは、Githubから公開しているこれらコードが全てとなるため、以降ではGithub内のコードについて解説する。

■図3．FIMSとMCMAの活動

3. MCMAのGithubのコード

3.1 mcma-librariesとmcma-projectsレポジトリ

mcma-librariesレポジトリは、MCMAフレームワークの実装を支援するための共通ライブラリの提供を目的としている。現在、masterブランチには、ライセンスとREADMEの2つのファイルしか存在しないが、developブランチに切り替えると、現在検討しているライブラリ群が現れる。ライブラリはプログラミング言語別に提供するため、それぞれの言語名をつけたフォルダにより管理されている。現時点では、.NETとNode.jsの2つのフォルダが存在する。本レポジトリで提供するライブラリは、大きく2つのカテゴリに分けられ、それぞれcore librariesと、cloud provider specific librariesとなっている。core librariesは、データのモデルとして、classes、types、message contentsなどを定義しているほか、必要なメッセージを正確に作成し、それを利用した通信を行うための各種ユーティリティーを提供している。一方、cloud provider specific librariesは、AWS、Azure、Google Cloudなどの特定クラウドの特定機能を利用するためのコードを抽象化して実装したものである。例えば、AWSを利用する場合には、ある処理を行うLambda functionを、API Gatewayを利用してREST-APIとして設置し、そのLambda functionの中では、RESTリクエストを処理してDynamoDBにその結果を格納するようなことを行う。AWS用のライブラリは、この時のRESTリクエストのハンドリングやDyanmoDBへのアクセスなどを簡単に行えるような抽象化機能を提供するものである。現時点では、AWS用のライブラリのみ提供されている。

mcma-projectsレポジトリは、MCMAフレームワークを用いたメディア処理ワークフローの実装例の提供を目的としている。すなわち、上述したmcma-librariesを使った実装例が提供されており、ライブラリの利用方法などの情報を取得することができる。本リポジトリでは、実装例ごとにトップディレクトリにフォルダを作ることにしており、現在提供されている実装例は、multi-cloud-ai-workflowの1つのみとなっている。また、本レポジトリには、現時点で3つのブランチが存在する。masterは安定して動作するコード、developは機能の拡張やバグFIXなどを行っている途中のコード、そして3つ目のlaunch-controlでは、現在新しい取組みが進められている。

3.2 サンプルアプリケーション構築の準備と手順

MCMAフレームワークの仕組みについて、実際に動作するアプリケーション例として、2019年1月末時点のmulti-cloud-ai-workflow（masterブランチ）を用いて説明する。このアプリケーションは、IBC2018のEBUブースにて、MCMAのデモとして展示したシステムと同等である。次に述べる手順に従うことで、各ユーザのAWS環境上にMCMAベースのサンプルアプリケーションが構築され、動作確認をすることが可能となる。

はじめに、各ユーザのローカルマシンに、次に示す事前準備を行う。

- Node.js v8.10.0のインストール[13]
- Terraformの最新バージョンをインストール[14]
- Java JRE/JDK 1.8以上をインストール[15]
- Gradleの最新バージョンをインストール[16]
- AWSのアカウント[17]
- Azure video indexerのフリーアカウント[18]

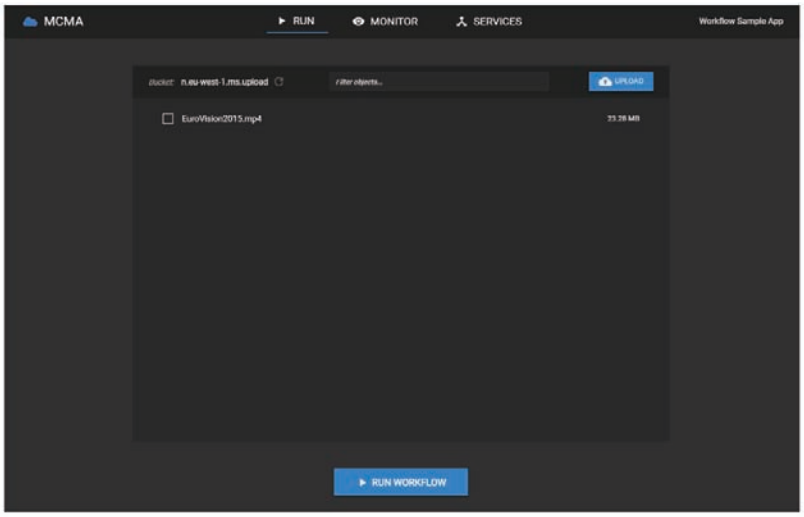
ここで、上記作業について簡単な解説と注意点を記す。本サンプルアプリケーションは、基本的にNode.jsで開発されている。AWSのLambda functionの利用ではNode.js v8.10.0を指定して開発しているため、必ずこのバージョンをインストールする必要がある。Terraformは、インフラストラクチャ構築ツールの1つで、コードを書いて動作させると、例えばAWS上に各種リソース（EC2、Lambda function、DynamoDBなど）を自動構築することができる。Terraformは、AWS、Azure、Google Cloudをはじめ、多くのサービスプロバイダーをサポートしており、機能のアップデートも頻繁に行われている。また今回、ビルド及び展開ツールとして、Gradleを利用する。このGradleはJava環境で動作するため、Java環境を整えた上で、Gradleをインストールする。アカウント関係では、AWSのアカウントのほか、Azureのvideo indexerのアカウントが必要となる。ただし、このアカウントはフリーのものでないと動作しないので、注意が必要である。手順の詳細は、該当するURLを参照されたし。

次に、サンプルアプリケーションのAWS上へのデプロイ手順を以下に記す。

①mcma-projectsレポジトリーを、ローカルマシンにクローンする
②multi-cloud-ai-workflowフォルダに移動する
③gradle.propertiesという名前のファイルを作成する
④**図4**のコードを上記ファイルに追記し、アカウント情報を更新する
⑤gradle.propertiesを保存する
⑥ターミナルを開き、カレントディレクトリをmulti-cloud-ai-

```
# Mandatory settings

environmentName = <YOUR ENVIRONMENT NAME – Whatever you like>
environmentType = <YOUR ENVIRONMENT TYPE – Whatever you like>

awsAccountId = <YOUR_AWS_ACCOUNT_ID>
awsAccessKey = <YOUR_AWS_ACCESS_KEY>
awsSecretKey = <YOUR_AWS_SECRET_KEY>
awsRegion = <YOUR_AWS_REGION>

# Optional settings, though without configuration some features may not work

awsInstanceType = <EC2_TRANSFORM_SERVICE_INSTANCE_TYPE – DEFAULTS TO "t2.micro">
awsInstanceCount = <EC2_TRANSFORM_SERVICE_INSTANCE_COUNT – DEFAULTS TO "1">

AzureLocation =  <YOUR AZURE REGION – USE "trial" FOR TESTING>
AzureAccountID = <YOUR AZURE Video Indexer Account ID>
AzureSubscriptionKey = <YOUR AZURE SUBSCRIPTION KEY>
AzureApiUrl = <AZURE VIDEO API ENDPOINT – DEFAULT IS: https://api.videoindexer.ai>
```

■図4．gradle. propertiesファイル



■図5．RUN画面

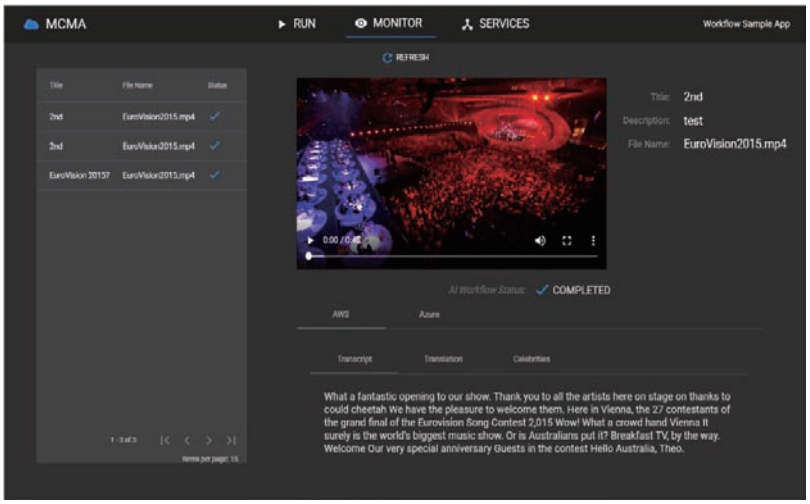
workflowにする

⑦gradle deployとタイプしてリターンを押す

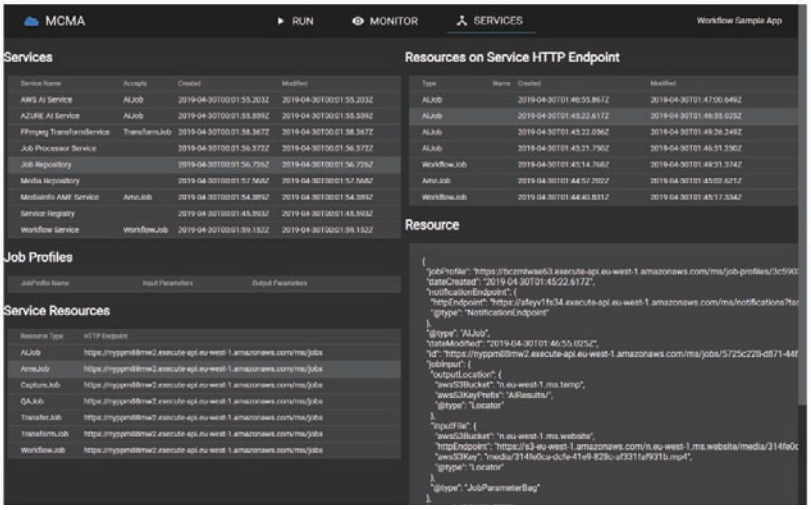
初回は、ビルドしてAWS上にリソースを構築するため、端末にもよるが5-20分ほどかかる。最後までエラーがなければ、ターミナルには、Apply complete! という文字とともに、Terraformの出力として、Outputs：の後に、作成した主なリソースのURLなどの情報などが出力される。この中のwebsite_urlで示されるURLにアクセスすると、AWS上に自動構築されたMCMAベースのサンプルアプリケーションが現れる。

3.3　サンプルアプリケーションの機能

自動構築されたサンプルアプリケーション（以降MCMAアプリと略す）の機能について説明する。MCMAアプリの画面は、RUN、MONITOR、SERVICESの3つのタブ画面から構成されており、それぞれ図5、6、7に示す。RUN画面では、ローカルマシンにある映像ファイルをアップロードし、ワークフローを起動（RUN）する画面となっている。画面右上の「Upload」ボタンをクリックすると、ファイル選択画面になるので、映像ファイル（動作テストには、英語音声の1分ほどの映像クリップが好ましい）を選択してOKボタンを押すことで、AWSのS3上へのアップロードが開始される。アップロード中は、画面下側にプログレスバーが表示され、処理状況が把握できるようになっている。アップロード終了後、図5に示すような、ファイル一覧にファイル名が現れるので、ファイル名横のチェックボックスにチェッ



■図6. MONITOR画面



■図7. SERVICES画面

クを入れて、画面下部の「RUN WORKFLOW」をクリックする。「Add Metadata」というダイアログが現れるので、タイトルと概要を入力して「RUN」ボタンを押す。なお、このタイトルと概要には、それぞれ必ず何か文字列が入力されていないと、途中でエラーとして扱うようになっているので、必ず入力する必要がある。これにより、本アプリにあらかじめ実装された固定のメディア処理ワークフローが開始される。すぐに、S3のBucketに戻るか、Workflowに移動するかを選択するダイアログが出現するので、Workflowの方を選択すると、図6に遷移する。

MONITOR画面（図6）では、左側にワークフローを適用した（している）ファイル名の一覧が現れ、ファイル名をクリックすることで、その映像ファイルにワークフローを適用させた（させている）結果が右側に表示される。アップロードした映像からは、プロキシーファイルが作成されるため、MONITOR画面では、そのプロキシーを再生することができるようになっている。また、本MCMAアプリに固定で実装したメディア処理である、AWSの英語音声認識、英日翻訳、有名人検出と、Azureの英語音声認識、有名人検出の結果が、それぞれタブをクリックして切り替えることで確認することができるようになっている。

SERVICES画面（図7）は、開発者のための確認ツールという位置付けで、後述するMCMAフレームワーク内でやり取りされる各種リソースやメッセージについて、実際のワークフロー処理で発生した内容が確認できるようになっている。

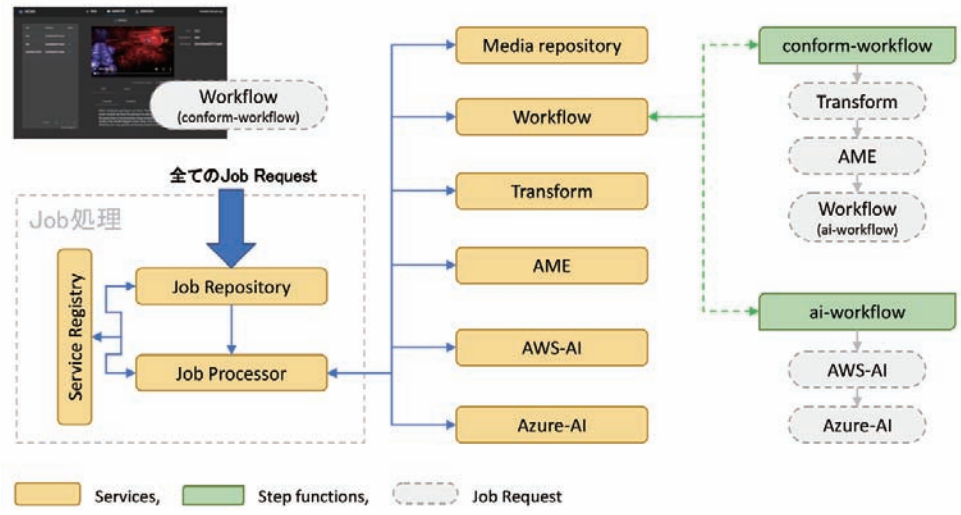
3.4　システムの概要

MCMAアプリを実現しているシステムの概要を図8に示す。基本的には、黄色で示される「サービス」群で形成されており、これらを適宜呼び出して、様々なメディア処理を行う。このサービスは、MCMAフレームワークにおけるメディアサービスであり、他の一般的なサービスと混同を避けるため、以降「MCMAサービス」と呼ぶ。

メディア処理の依頼は、Jobというメッセージで表現され、図中の灰色の丸点線で囲まれた「Job Request」がそのメッセージを表している。各MCMAサービスは、それぞれあるメディア処理を提供する役目を有するが、中でも、MCMAサービスの種類や利用するためのエンドポイントなどの情報を保有する「Service Registry」と、Jobの登録・管理をする「Job Repository」、登録されたJobについて、実際の処理を担当するMCMAサービスに依頼を投げる「Job Processor」の3つは、それぞれ連携して、MCMAフレームワークで扱う全てのJob処理を行う。一方、MCMAサービスを組み合わせてワークフローを形成する機能は、図中の緑色で示した「Step Functions」を利用している。Step Functionsは、AWSが提供するサービス名の1つであり、フローチャートのようなワークフローを形成することができる。本アプリケーションでは、conform-workflowとai-workflowの2つのStep Functionsを、Terraformにより自動作成している。conform-workflowの方は、必要に応じてプロキシーを作成するためのTransformサービスを呼び出し、その次にファイルのメタデータを抽出するAMEサービスを呼び出し、最後にもう1つのai-workflowを呼び出すためにWorkflowサービスを呼び出すように作られている。一方、ai-workflowの方は、AWS-AIサービスとAzure-AIサービスを呼び出すようになっている。

実際の動作について、図8を使って説明する。MCMAアプリケーションサイトにおいて、映像ファイルをアップロードしてワークフローを開始すると、conform-workflowを呼び出すためのWorkflowサービスを要求するメッセージがJob Repositoryに送られる。そのリクエストは、Job Processorに渡り、Job Processorは、Service Registryの中に、その処理を行ってくれるサービスがあるかを探し、この場合、Workflowサービスに要求を送る。Workflowサービスでは、Jobの中にconform-workflowを起動するよう記述されているので、conform-workflowを起動させる。conform-workflowの最初の処理は、プロキシーを作成するためのTransformサービスを要求するメッセージをJob Repositoryに送り、以降同様にJob Processor、Transformサービスにより処理される。これが終了すると、conform-workflowの2番目の処理として、アップロードした映像ファイルのメタデータを抽出するAMEサービスを要求するメッセージをJob Repositoryに送り、以下同様である。conform-workflowの最後の処理は、もう1つのai-workflowを起動するために、Workflowサービスを要求するメッセージをJob Repositoryに送って終了する。次のai-workflowの動作もこれまで説明したものと同様であり、全てのMCMAサービス起動の要求は、Job Repositoryを通して行われる。

このように、Jobを扱う部分を共通化しているため、新しいMCMAサービスを構築した際には、Service Regisitry



■図8.　MCMAアプリのシステム概要

にその情報を追加することで、容易に利用が可能となることが分かる。

3.5　MCMAサービスの構築パターン

AWS上におけるMCMAサービスの構築について**図9**に示す。RESTインタフェースは、AWSのAPI Gatewayというサービスを利用して構築する。その後段に、API HandlerとWorkerと名付けたLambda関数を用意し、必要なデータを保持するためのDynamoDBを1つ用意する。基本的にはこの4つのAWSコンポーネントでMCMAサービスのインタフェースを構築している。2つのLambda関数を用意している理由は、現在のLambda関数は、処理時間に15分以内という制限があり、それ以上長い処理を行うと、全てのデータを失う仕様となっている。一般的にメディア処理は時間がかかることも多いため、API Handlerは、リクエストを受け付けたことを、すぐリクエスト元に通知するとともに、次のWorkerというLambda関数に実際のメディア処理の管理を任せる仕組みをとっている。

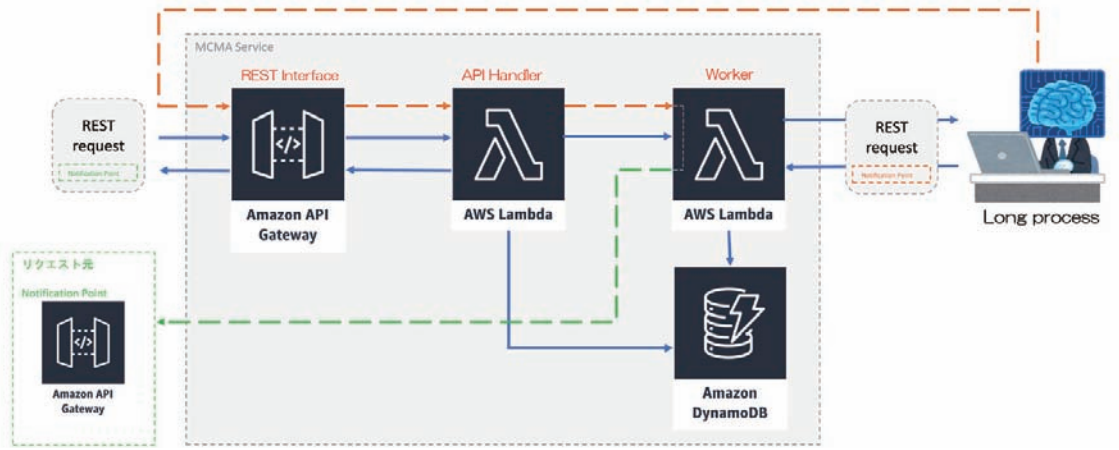
実際のメッセージの流れについて説明する。図中左から、RESTリクエストが届くと、API Gatewayは、そのリクエストをAPI Handlerに転送する。API Handlerは、必要な情報（例えば、Jobを受け付けてJobIDを作成した）をDynamoDBに書き込み、API Gatewayを介して、受付完了（JobID）をリクエスト元に返す。同時に、Workerを起動して、メディア処理を開始する。このメディア処理については、Worker内で15分以内に処理が完結できる場合と、更にRESTリクエストを作成して外部サービスにお願いする場合が考えられる。外部サービスにお願いする場合は、終了通知を受け取るためのNotification Point（この場合、前述のAPI Gateway）を含めた形で、RESTリクエストを発行する。図中右端のLong processが終了すると、その終了通知がオレンジ色の点線に沿ってAPI Gatewayに戻り、API Handlerを経由して、再度Workerに入る。Worker内では、それが終了通知であることを確認すると、自身の処理についても完了となるため、1つ前のリクエスト元から指定されたNotification Point（緑色の点線）に、終了を通知するような仕組みとなっている。**図8**において、Job Requestが、Job Repositoryサービスに入り、Job Processorサービスを経由して、実際のメディア処理サービスに渡っていく過程は、全てこのような仕組みで動作している。

3.6　MCMAで規定する候補項目

MCMAの目的については、最小限のパラメータ記述を持つメディア処理用の簡略化されたREST-APIセットと、MCMAフレームワークのシステムを構築するための共通ライブラリの提供であることは前に述べた。GithubのMCMAアプリを構成するMCMAサービスについて、そのRESTインタフェースを抜き出すと、次のようになる。今後は、この部分を共通ルールとして決めていくという流れとなる。

●Service Registry
- /services　[Get, Post]
- /services/{id}　[Get, Put, Delete]
- /job-profiles　[Get, Post]
- /job-profiles/{id}　[Get, Put, Delete]

●Job Repository
- /jobs　[Get, Post]



■図9.　MCMAサービスの構築パターン

- /jobs/{id}　[Get, Delete]
- /jobs/{id}/stop　[Post]
- /jobs/{id}/cancel　[Post]
- /jobs/{id}/notifications　[Post]

●Job Processor

- /job-processes　[Get, Post]
- /job-processes/{id}　[Get, Delete]
- /job-processes/{id}/notifications　[Post]

●Media Repository

- /bm-contents　[Get, Post]
- /bm-contents/{id}　[Get, Put, Delete]
- /bm-essences　[Get, Post]
- /bm-essences/{id}　[Get, Put, Delete]

●Azure-AI, Transform, Workflow

- /job-assignments　[Get, Post, Delete]
- /job-assignments/{id}　[Get, Delete]
- /job-assignments/{id}/notifications　[Post]

●AME, AWS-AI

- /job-assignments　[Get, Post, Delete]
- /job-assignments/{id}　[Get, Delete]

また、RESTインタフェースに渡す最小限のパラメータ記述とメッセージのフォーマットについては、mcma-coreライブラリの中に関連する情報がある。例えば、パラメータ記述については、1つのJobに対して、inputParameters、outputParameters、optionInputParametersの3項目を指定することができ、1つのパラメータは、paramterNameとparamterTypeのセットで表現し、それらをまとめるためのparamterBagというグルーピング記述が用意されている。

現在のところ、このmcma-coreの中で最低限必要なデータ構造だけを規定し、ユーザはこのmcma-coreを拡張する形で自由にアプリケーションを実装できる方向で議論が進められている。

4. おわりに

本稿では、汎用的なメディア処理基盤を構築するための標準化の1つとして、FIMSとその後継技術にあたる、EBUのMCMAプロジェクトにおける活動について紹介した。現在、本活動に参画している組織は少ない状況ではあるが、IBCなどの展示を通して、汎用的なメディア処理APIの共通化を望む声は多いと感じる。一方で、現在のIPによる映像伝送の規格化と、現場への導入が一段落した後に、この手の議論が注目されるだろうと唱える者もいる。また、クラウドなどの技術は日進月歩で変化しており、詳細な部分までしっかりと標準化をするのではなく、目的を達成するための共通ライブラリを頻繁に更新提供することが、変化の速い環境では重要であるという意見も多い。将来の柔軟で拡張性の高いコンテンツ制作のエコシステム実現のためには、汎用的な共通化されたAPIが普及することは重要であり、今後のMCMAの進展に期待したい。

参考資料

[1] EBU：https://www.ebu.ch/home
[2] FIMS：https://fims.tv/
[3] AMWA：https://www.amwa.tv/
[4] FIMS立ち上げ：柴田賀昭、“Media SOA/FIMSのご紹介、”映像情報メディア学会年次大会、11-2, 2011
[5] FIMS-Github：https://github.com/fims-tv/fims
[6] WSDL：http://www.w3.org/TR/2001/NOTE-wsdl-20010315
[7] XSD：http://www.w3.org/TR/xmlschema11-1/
[8] XML：http://www.w3.org/TR/xml/
[9] XML Schema：http://www.w3.org/XML/Schema
[10] MCMA：https://tech.ebu.ch/groups/mcma
[11] mcma-lib：https://github.com/ebu/mcma-libraries
[12] mcma-pjt：https://github.com/ebu/mcma-projects
[13] Node.js：https://nodejs.org/en/
[14] Terraform：https://www.terraform.io/
[15] Java：https://www.oracle.com/technetwork/java/
[16] Gradle：https://gradle.org/
[17] aws：https://aws.amazon.com/jp/
[18] Video Indexer：https://docs.microsoft.com/ja-jp/azure/media-services/video-indexer/video-indexer-use-apis