FUJITSU ファミリ会　2020 年度論文　一般論文

みずほ新勘定系システム『MINORI』を活用した API エコノミーの実現
～次世代金融への転換をリードするための開発プロセス改革～

みずほ情報総研株式会社

■ 執筆者Ｐｒｏｆｉｌｅ ■

前田　慎平

2015 年　みずほ情報総研（株）入社
2020 年　現在　開発本部第 1 事業部第 3 部所属

岩楯　奈那

2012 年　みずほ情報総研（株）入社
2020 年　現在　開発本部第 1 事業部第 3 部所属

小山　鉄平

2010 年　みずほ情報総研（株）入社
2020 年　現在　開発本部第 1 事業部第 3 部所属

2011 年　みずほ情報総研（株）入社
2020 年　現在　技術企画部所属

石井　宏明

2011 年　みずほ情報総研（株）入社
2020 年　現在　技術企画部所属

竹島　幸之輔

■ 論文要旨 ■

昨今の金融業界を取り巻く環境の変化に伴い、既存の銀行ビジネスモデルからの脱却、次世代金融への転換が急務である。融資による金利収入だけでなく、新規ビジネス（銀信証連携の推進、外部企業連携によるイノベーション、銀行機能の他行提供など）の確立に向け、銀行業務の API 化を推進しなければならない。

＜みずほ＞では、多様な勘定系機能を API 化することで、早く安くシステム構築する「作り方改革」や、様々な利用ニーズに応えるサービスの高度化に取り組んでいる。SOA で設計した新勘定系システム「MINORI」を活用し、店舗における行員業務を API 化することで、店頭タブレット端末やスマートフォンなど取引チャネルの多様化を実現し、CX 向上を進めている。さらなる作り方改革のため、銀行勘定系 API の理想的な設計思想や、コンテナ技術を活用した API プラットフォーム像を構想した上で POC を実施し、新規サービス構築を短期間・低コストで実現するための技術検証を行った。

■ 論文目次 ■

■ 図表一覧 ■

1. はじめに

1.1. 当社の概要

弊社はみずほフィナンシャルグループ（以下、みずほ FG）の IT 戦略&コンサルティングを担っている。

主な事業内容は、IT をコアとした「コンサルティング」「システムインテグレーション」を通じ、グループ各社、並びに民間企業から官公庁まで幅広い分野のお客様に対し、経営戦略の方向性を具体化するための最適なシステムの構築を行っている。

1.2. 新勘定系システム「MINORI」

みずほ FG は、1988 年に構築し、規制緩和・制度対応・顧客ニーズの多様化への対応、銀行の統合等により複雑化・肥大化していた勘定系システム「STEPS」を、全面的に一から構築し直し、2019 年に新勘定系システム「MINORI」を完成させた。

「MINORI」は、SOA(Service Oriented Architecture）を拡張した「みずほ SOA」を取り入れた。「みずほ SOA」とは預金・融資・内国為替・外国為替等の銀行業務の単位で機能をコンポーネント化し、各機能をハブ＆スポーク構成で配置・集約すると共に、以下図 1 のような「機能のサービス化」と「層の定義」に基づいた設計思想である。

勘定系の各システムを対顧客チャネルとなる「アプリケーションフロントエンド層」と勘定系業務コア機能となる「サービス層」、それを繋ぐ「エンタープライズサービスバス層」に分割し、変更時の影響箇所局所化を実現した。

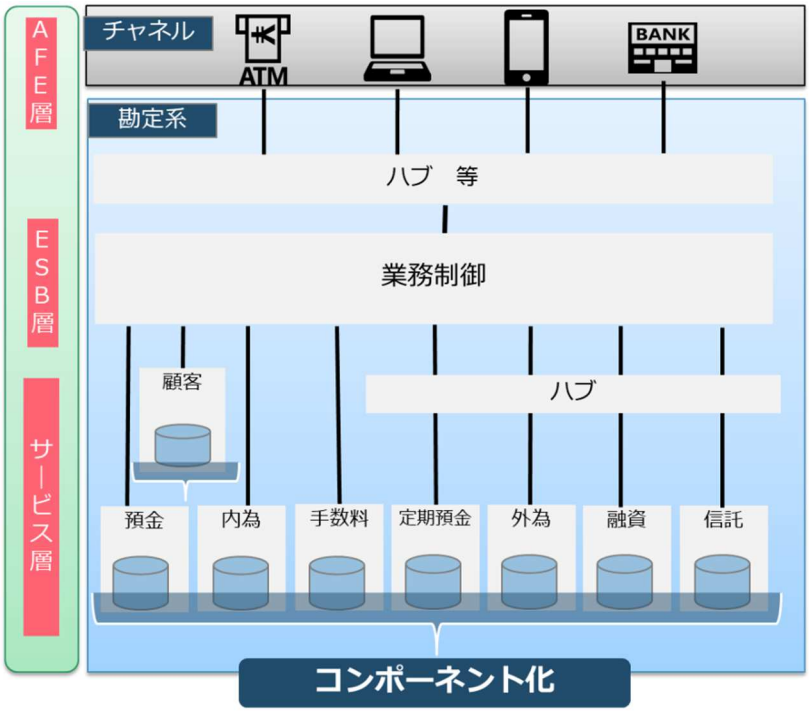

図. 1　みずほ SOA の設計思想

2. みずほ銀行における API 開発の現状と将来像、解決すべき課題

2.1. みずほ銀行オープン API の現状

2017 年 5 月に成立した改正銀行法により、銀行には「オープン API に係る体制整備の努力義務」が課されており、みずほ銀行もこれに取り組んでいる。

Fintech[1]企業は従前より、金融機関の口座情報等をスクレイピング[2]により抽出し、Fintech サービスの利用者に提供していた。スクレイピングではログイン ID／パスワード等の顧客情報を Fintech 企業が保管し、Fintech サービスの利用者に代わって自動でログインを行う。この方法では第３者である Fintech 企業が顧客のログイン ID／パスワードを保持するという、セキュリティ上の懸念がある。また、銀行としてもアクセスコントロールができず、サーバー負荷増大の懸念がある。

みずほ銀行では 2017 年の法改正に先駆けて、Fintech 企業が従前スクレイピングを行っていたインターネットバンキングの参照・更新 API を中心にオープン API 化を実現した。行内システムへの負荷軽減のため API-GW[3]（API-Management）を構築、みずほダイレクト等のインターネットバンキングのフロントシステム上に API 変換機能を構築することで、勘定系システムの改修なく API 提供を可能とした。

みずほダイレクトの API 対応概要は図２の通り。

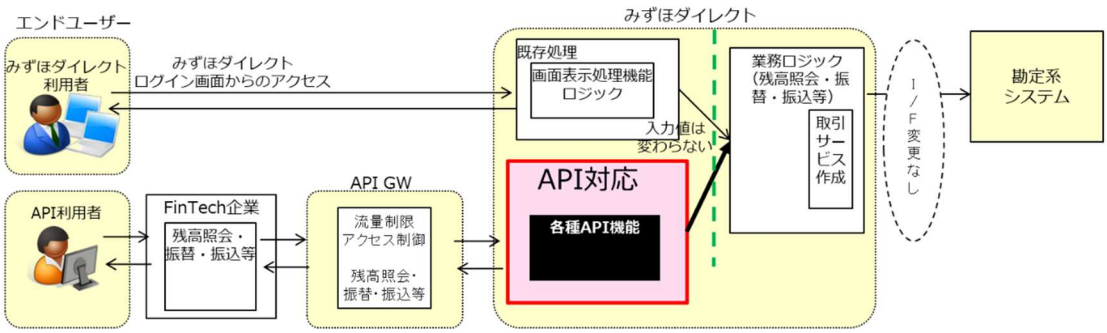

図. 2　みずほダイレクトの API 対応概要

2.2. API 活用方針・将来像

前述のインターネットバンキングの API 対応を進める一方、金融業界を取り巻く環境からは、Fintech 企業の台頭に加え、デジタル通貨の普及、マイナス金利による逆鞘など、既存の銀行ビジネスモデルからの脱却、次世代金融への転換が急務となっている。次世代金融への展開の１丁目１番地として位置付けられるのが、店舗戦略の転換であり、既存店舗の削減とバックオフィスへの事務集約のために、既存の行内事務の大幅な改修が求められる。加えて、次世代金融のビジネスモデルにおいては、融資による金利収入以外の収益確保が必須

[1] 金融を意味する「ファイナンス（Finance）」と、技術を意味する「テクノロジー（Technology）」を組み合わせた造語。概ね「ICT を駆使した革新的（innovative）、あるいは破壊的（disruptive）な金融商品・サービスの潮流」という意味で使われる。
[2] サービス利用者がサービス提供側のサーバーに主に WEB からアクセスし、必要な情報を抽出する手法。
[3] API Gateway の略。API ビジネスで必要となるトラフィック管理や API カタログ等の機能を総称したもの。

であり、トランザクションサービスによる手数料収益の確保はその柱の一つとなる。

行内事務の再編と新規トランザクションサービスの構築においては、API 化による既存機能の再利用、再構築が有効であり、行内利用も含めた API 活用を推進している。将来的な API 活用により目指すべき将来像は以下の通り。

2.2.1. 早く安く作る「作り方改革」

API として必要な機能を洗い出し構築、「部品化」し、重複開発やテストの低減を可能とする。さらに、図３に示すように新規システムとの接続時の再利用が容易となることで、「作る」から「使う」開発が可能となり、早く安くシステム構築する「作り方改革」を実現する。

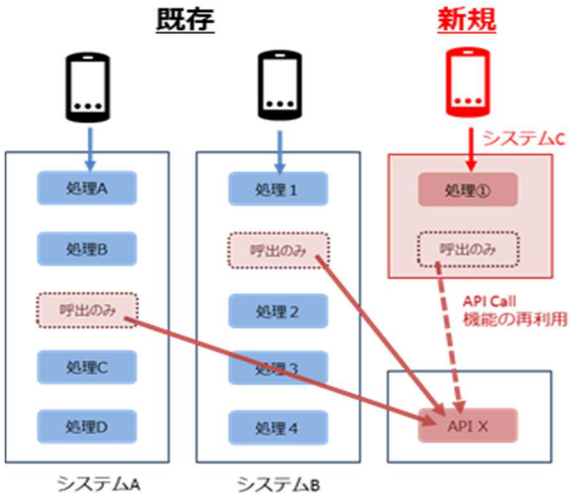

図. 3 「作り方改革」のイメージ

2.2.2. グループ間連携のさらなる推進

勘定系システムを中心に API 化を進めているが、国際系、市場系などの機能も API 化する。これらを API 連携することにより、様々な利用ニーズに応えるために必要な機能を提供、新たなサービスの創出を実現する。

さらに、銀行だけでなく、信託、証券でも API を構築し、図４に示すように API を集約・組合せることで、サービスの高度化を実現し、みずほが掲げる「One Mizuho」戦略を加速させる。

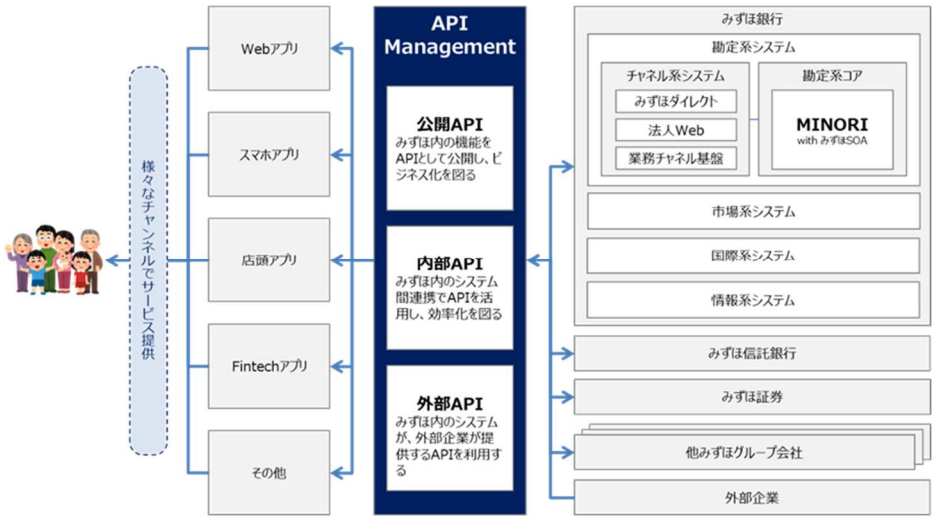

図. 4　API 化によるグループ間連携

2.3.　解決すべき課題

上記 2.2 章で述べた将来像を実現するにあたり、以下２つが課題となっている。

2.3.1.　銀行業務機能の API 化

新たなサービスを創出していくためには、API として利用できる「部品」のバリエーションを増やすことが、提供できるサービスの種類に直結するといえる。銀行業務取引の内、ATM やインターネットバンキングで実施出来るのは支払・振込等の一部取引であり、大半の取引が営業店端末での実施に限定されている。そのため、この営業店業務を API 化することが、API 活用の上で必要不可欠となる。

営業店業務は、固有のチェック・認証・承認機能（以下、固有処理と呼称）が多数存在する。これらは行内業務としては必要な機能であったが、銀行業務の API を外部公開する場合は、承認機能などは不要となる。しかし、従来の勘定系システムはモノリシック（一枚岩）な造りとなっていて、機能ごと・サービスごとに流用・迂回する場合複雑な分岐を組み込むことになる。新規に別フローを作るとしても工数が膨らむので、現実的ではなかった。みずほでは、SOA を採用した MINORI を活用することで、この課題解決を可能とした。取り組みの内容について、３章で紹介する。

2.3.2.　使いやすい API 提供を迅速に実現する為のレイヤ構築

銀行業務機能を API 化し外部公開したとしても、それが業務知識を持った銀行員しか使い方が分からないものでは意味がない。サービスとして誰もが使いやすい API を提供することが望ましい。例えば、現在は Fintech 企業をはじめとする外部企業が、スクレイピング等の技術や情報加工するアプリケーションを構築してサービス展開しているが、みずほの保有する顧客情報を無償で使用されている状態であるため、みずほとしてのビジネス展開

には繋がらない。サービスとして誰もが使いやすい API として銀行側で公開すれば、付加価値としてビジネスになる。

一方、付加価値のある API を、従来の勘定系システム及びその周辺システムで公開することを想定すると、堅牢性を目的とした既存の開発ルールやシステム構成が制約となり、コスト・期間が増大してしまう。

既存の堅牢性を重視した勘定系システム（Systems of Record）を維持しつつ、顧客要望に迅速に対応するシステム（Systems of Engagement）に API を提供したい。しかし、単純に一対一で繋ぐと、チャネルの追加や顧客要望への対応の都度、新たな API を作る必要が出てきてしまい、迅速性が損なわれ、かつ無駄なコストが生じる可能性がある。そのため、再利用性を高められるように API を階層化（レイヤリング）し、早期に API を公開可能とする仕組みがあることが望ましい。

今般、API を「早く」「安く」公開する為、API 開発基盤「Digital Agility Layer」を構想したため、その検証内容および結果について 4 章で解説する。

3. 銀行 API のサービス拡大に向けた取り組み

2 章で述べた通り、銀行業務には固有処理が多数存在する為、API 化が困難であった。

しかし、みずほ SOA を取り入れ「MINORI」を完成させたことにより、サービスのコンポーネント化とインターフェース整理、固有処理の制御機能を勘定系システムから外出しし対顧客システムに集約が出来た為、MINORI 本体には影響を与えずに「アプリケーションフロントエンド層」にあたる一部のシステムの対応のみで銀行業務の API 化が可能となった。事例を紹介する。

3.1. 店頭タブレット案件（事例①）

3.1.1. 案件概要

2020 年 10 月から営業店端末業務のオンライン取引が可能となる顧客用の店頭タブレットを各支店に設置する。

従来設置していたタブレット端末は、勘定系システムと直接繋がっておらず、顧客が必要事項をタブレット端末へ入力した後その内容を紙帳票に出力し、紙帳票を行員が別途営業店端末に入力し、取引を実施していた。今次案件により、図 5 に示すように、顧客がタブレット端末に入力した取引内容を勘定系システムに API を使って直接送信することが可能となる。口座開設や住所変更、入出金等の主要 8 業務で行員による営業店端末の操作なしに取引を完結させることができ、行員事務の削減と顧客の利便性向上を実現する。

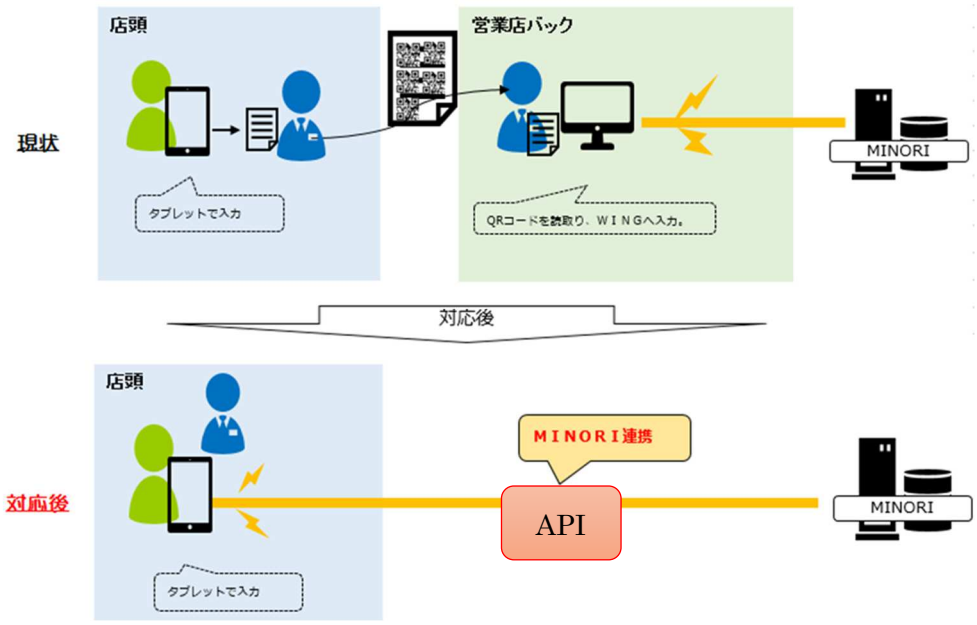

図. 5　店頭タブレット案件の対応イメージ

3.1.2.　API 対応

MINORI では前述の通り、みずほ SOA のアーキテクチャで構築したことにより、「預金・融資・内為・外為等の勘定系コア機能」と「ATM や営業店端末、みずほダイレクトといったチャネル機能」を分割して、階層構造で構築している。勘定系コア機能を、どのチャネルからでも汎用的に呼び出すこと（チャネルフリー）ができるようになり、勘定系業務コアシステムに改修を加えずに、外部システムに対して新たな勘定系サービスを API で機能提供しやすい環境となっている。

今回タブレット端末から提供可能とする営業店端末業務では、ユーザ認証機能による行員ログイン、勘定系システムの参照・更新による取引実行、権限者による承認といったフローが発生した。API 取引では行員ログインは不要であるが、状況に応じてユーザ認証、取引実行、承認それぞれの機能を個別に API として呼び出す必要がある。図 6 に示すように「業務チャネル統合基盤」システムにはこれらの営業店端末固有機能が集約されているため、それぞれの機能を部品として呼び出すことで、営業店端末業務を API で実施することが可能となる。また、業務チャネル統合基盤は営業店端末、勘定系システム間の取引サービス電文を生成する役割を担っているため、API 制御機能を業務チャネル統合基盤上に構築することで、勘定系システムとの従来のインターフェースを流用することが可能である。

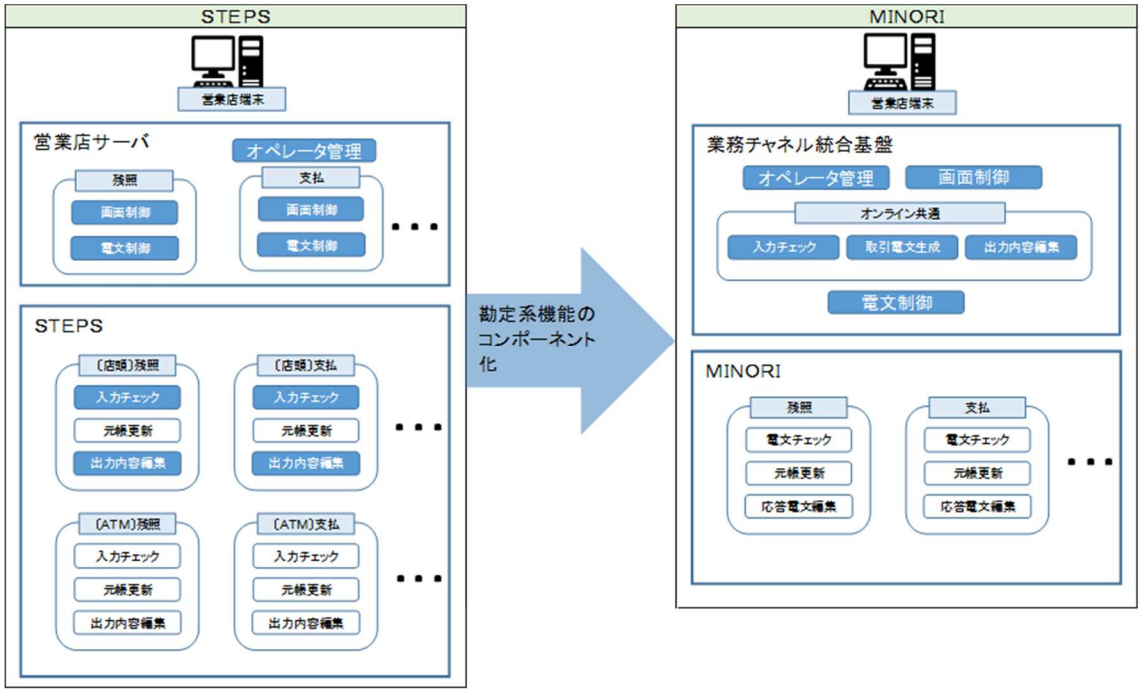

図. 6　営業店端末固有機能の集約

営業店端末のインターフェースを制御する「業務チャネル統合基盤」にて API 制御機能を構築することにより、図 7 の通り、みずほ SOA における ESB 層およびサービス層にあたる勘定系コアシステムの改修をせずに、アプリケーションフロントエンド層のみの API 化でサービスを実現することができた。

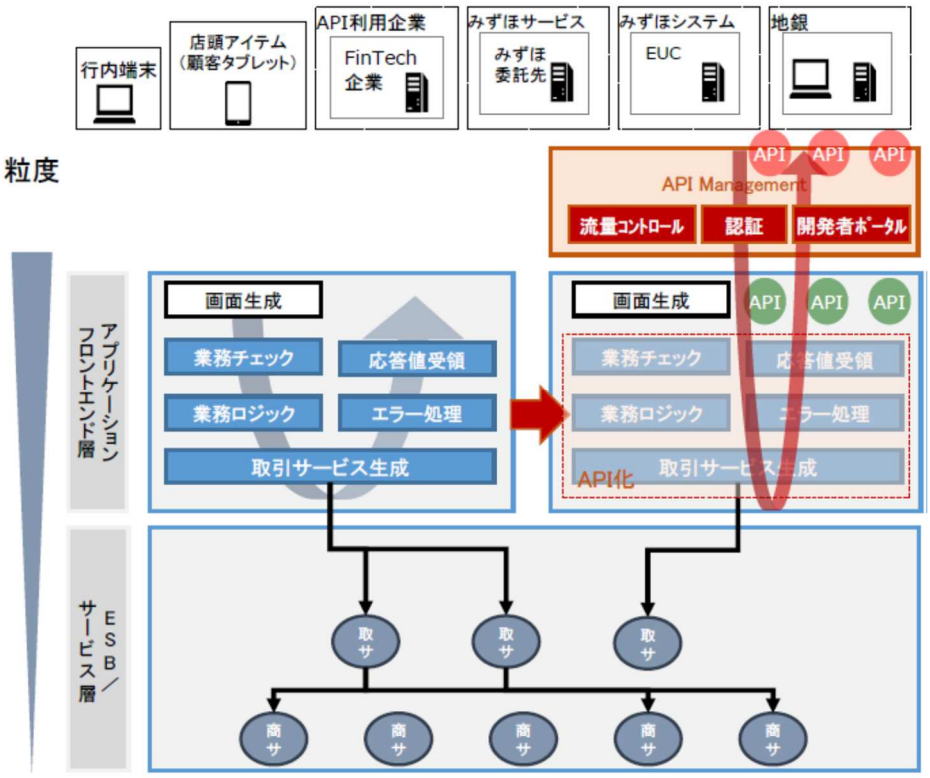

図. 7　みずほ SOA における API の位置付け

2.3.2 章で挙げている API の階層化について、課題解決を見据え、以下の 2 層の API レイヤを業務チャネル統合基盤上に構築することとした（図 8 に示す通り）。

・　SystemAPI

API 提供システム（業務チャネル統合基盤）の機能を実現する。勘定系コア機能を呼び出す。利用システム固有の要件は実装していないため、一度作成すると再利用が可能である。

・　ExperienceAPI

利用システムの要件を実現する。ExperienceAPI で、SystemAPI を呼び出し、組み合わせることで、利用システムの要件に合わせたサービスを実現する。

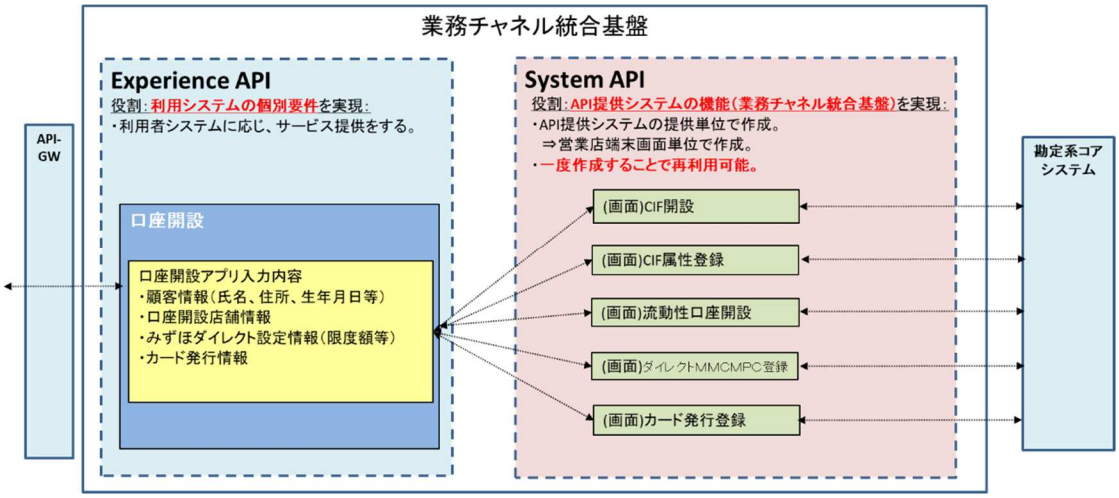

図. 8　業務チャネル統合基盤の API レイヤ概要

上記図８は、口座開設取引を例に図示したものである。従来、営業店窓口では口座開設を営業店端末で実施する場合、お客様情報の登録、口座開設店舗の登録、カード発行情報の登録などをそれぞれの営業店端末画面から行員事務にて行っている。この各画面単位の機能を SystemAPI にて実現し、ExperienceAPI にて口座開設取引に合わせた SystemAPI を呼び出す。タブレット端末にお客様が必要な情報を入力するだけで、API を使って勘定系コアシステムに連携し、取引を実施することができる。

店頭タブレット案件では、お客様がタブレット端末に入力した内容に対して、営業店端末固有処理を「業務チャネル統合基盤」に構築した ExperienceAPI にて中間的に処理した上で、SystemAPI に連携することで、勘定系サービスを提供することができた。

MINORI 構築以前は、新規チャネル追加の度に勘定系システムへの変更対応が入っていた。一方、本案件では勘定系コアシステムへの修正なしに、アプリケーションフロントエンド層の API 制御機能の構築のみの対応で、新たなチャネルでの取引を実現することができた。

3.2.　API を活用した諸届アプリ開発案件（事例②）

3.2.1.　案件概要

みずほは、昨今のコロナ禍の状況において個人提携取引の非対面化を加速させており、その一環として、「API を利用した諸届アプリ」の開発を実施中である。

「業務チャネル統合基盤」の API を使って、スマートフォンのアプリと勘定系システムを連携することによって、お客様が営業店窓口に来店せずに、個人で保有するスマートフォンから直接、諸届取引（住所変更や電話番号変更、通帳・カード再発行等）を実施すること

が可能となる（2021 年リリース予定）。

お客様は営業店に来店せずにアプリから諸届取引を実施できるため、行員の窓口業務量を削減できる。諸届取引の一部（住所変更など）については、みずほダイレクトから実施することができるものの、事務センターにて行員が事務処理を行うことで MINORI 連携している。諸届アプリでは、API を利用して MINORI 連携するため、事務センターにおける事務についても軽減することができる。API 対応概要は以下図 9 の通り。

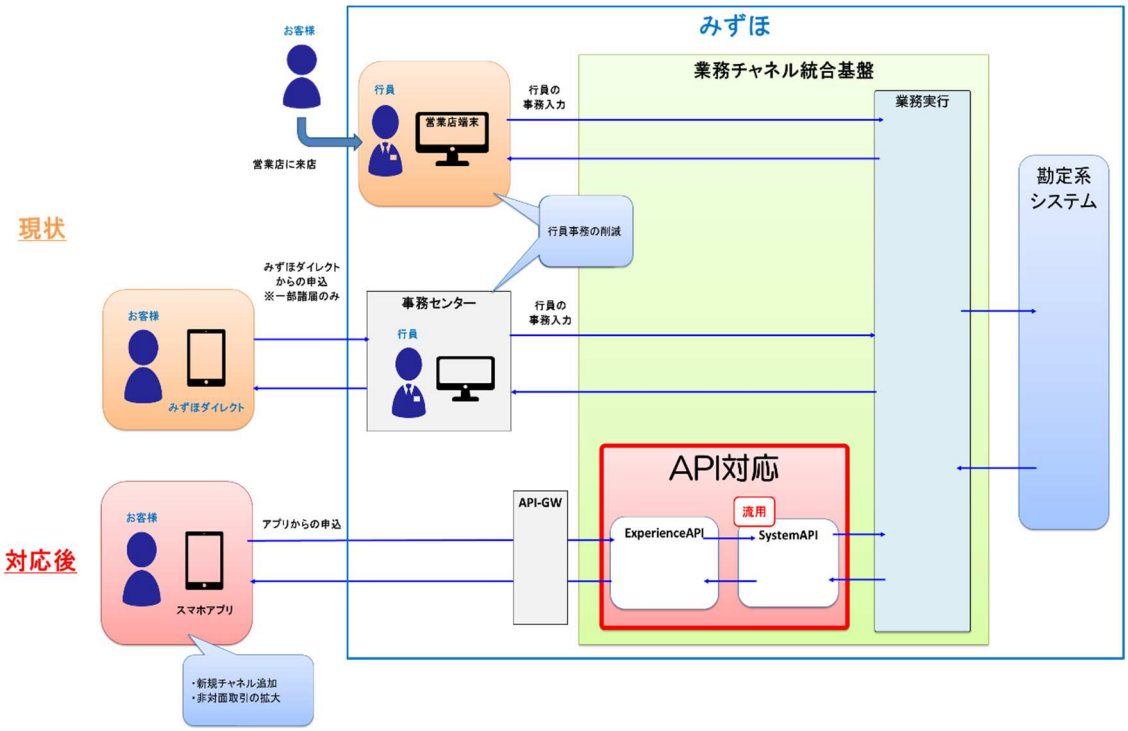

図. 9　諸届アプリ開発案件の対応イメージ

3.2.2.　API 対応

店頭タブレット案件にて SystemAPI の構築が出来ている。本案件では、業務要件に合わせて、この SystemAPI を流用し、ExperienceAPI を構築することで、開発期間の短縮・コストの削減を実現する。

4. Digital Agility Layer の POC（Proof of Concept）

4.1. 4 象限と Digital Agility Layer

弊社は銀行勘定系システムが中心であるため、非常に高い堅牢性・安全性を求められている。一方、Fintech 企業をはじめとする外部企業との連携においては、様々な要望が挙がると予想され、素早く対応・変化できるような仕組みが必要である。弊社では、このような要件を整理するために、システム特性を 4 つの象限[4]で分類した。

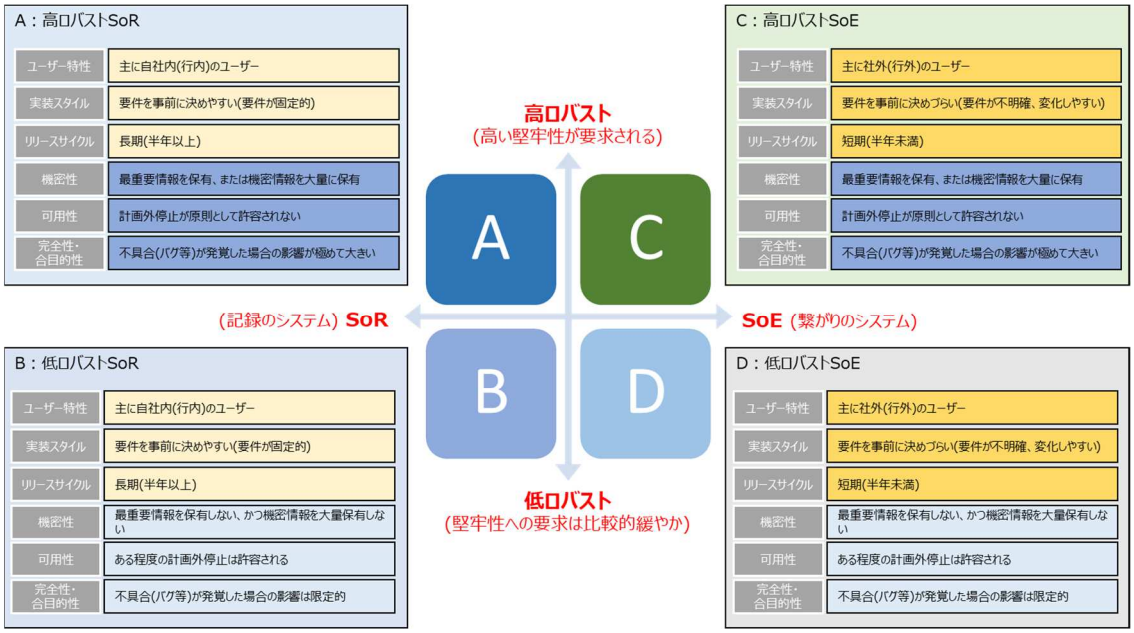

図. 10　銀行勘定系システム特性の 4 象限

システム領域ごとに特性があるので、全てを同様のアーキテクチャにするのではなく、「必要な堅牢性」と「激しい変化への対応必要性」を軸に、必要なアーキテクチャを考える。

例えば MINORI（勘定系コア）は、「堅牢性が必要だが、サービスのコアは急激に変化しない」（A 領域）。チャネル系システムは、「堅牢性が必要で有りながら変化スピードが必要」（C 領域）であり、異なるアーキテクチャが必要と考えた。

そこで今回、目指すべき将来像として、C 領域の対顧チャネル系システムについて、図 11 の通り「Digital Agility Layer[5]」構想を提唱する。

4　独自用語。システムの性質を「必要な堅牢性」と「激しい変化への対応必要性」を軸に 4 つの象限に分けて表したもの。
5　独自用語。銀行勘定系システムにおける、アプリケーションフロントエンド層のマイクロサービス化構想。

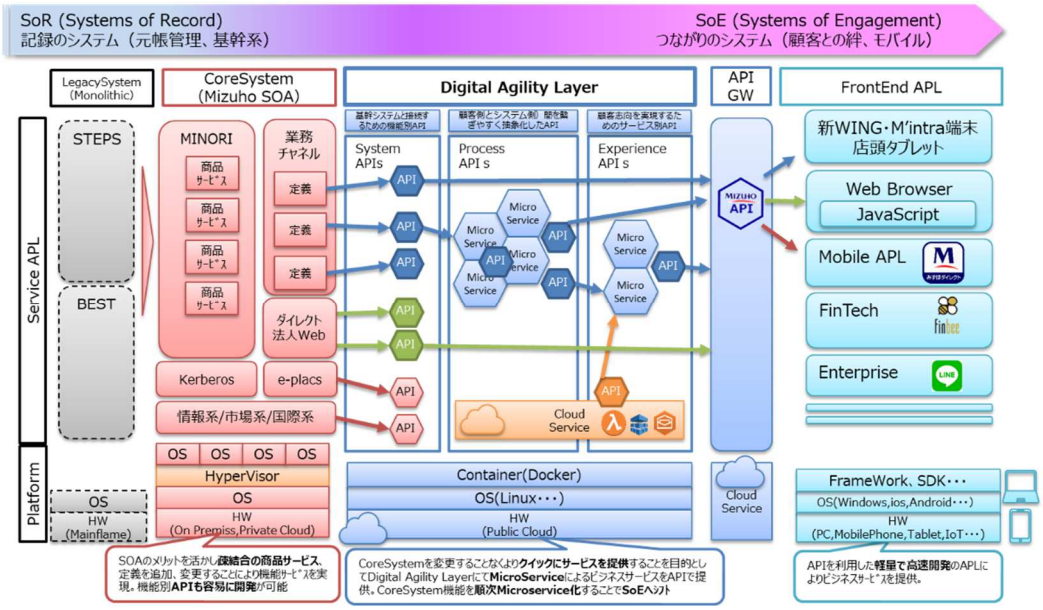

図. 11　Digital Agility Layer アーキテクチャイメージ

4.2. Digital Agility Layer（以下 D.A.L）とは

D.A.L とは、C 領域向けのマイクロサービス化構想であり、API の開発・管理に特化したコンテナ[6]基盤を指す。銀行システムの各機能に接続する System 層[7]と、機能のラッピングを行う Process 層[8]、対顧デバイス・画面に対応する Experience 層[9]の 3 つに分け、API の再利用性や業務拡張性の向上を期待するもの。

D.A.L を銀行システムと API-GW の間に配置し、勘定系システム及びその周辺システムの各機能を D.A.L から呼び出し可能とすることで、付加価値の高い API を「早く」「安く」公開すること可能とする。

尚、Process 層を含めた 3 層構造としているが、これは概念的なものであり、System 層と Experience 層の 2 層構造（Experience 層の API から、System 層の API を直接呼出し）でも構わない。各システムの API 制御機能を一つに纏める際、中間処理として再利用可能なものがあれば Process 層を配置する。

4.3. D.A.L にて採用している技術・概念

4.3.1.　マイクロサービスアーキテクチャ

マイクロサービスアーキテクチャとは、一つのアプリケーションを、ビジネス機能に沿った複数の小さいサービスの疎に結合された集合体として構成するもの。マイクロサービス

6　コンテナ型の仮想化環境の意味で使用。稼働中のオペレーティングシステム（OS）の一部を分離して、他と隔離された専用のエリアを用意し、その上でソフトウェアを動作させる。

7　独自用語。D.A.L における層の概念で、銀行勘定系システムの各機能を呼び出す API 群を持つ層。

8　独自用語。D.A.L における層の概念で、機能をサービスの単位に集約した API 群を持つ層。System 層と Experience 層を直接接続してしまい拡張性・再利用性を損うことを防ぐための、中間的な集約層。

9　独自用語。D.A.L における層の概念で、顧客体験を実現する API 群を持つ層。

アーキテクチャでは、各サービスを細かい粒度で作り、サービス同士を API で繋いでいく。

例えば、一つのモノリシック（一枚岩）なシステムで構築した場合、変更は全体に影響するので、影響調査やテストに時間がかかる。細かい粒度でサービスアプリケーションを構築・デプロイし、インターフェースだけ変えないようにしていれば、個々の変更は他サービスに影響を与えない。また、必要なリソース（CPU 能力が必要/メモリを大量消費等のアプリ毎のニーズ）も分割できるので、スケーリングの際に不要なリソースを抱えずに済む。新しいサービスを追加する際に言語・フレームワークを新規に選択できるのもメリットと言える。

4.3.2. コンテナオーケストレーション

マイクロサービスには、開発効率性やスケーリング時のメリットがある一方で、「多くのマイクロサービスが生まれ、管理が煩雑になる」というデメリットがある。今回 POC では上記欠点を緩和するべく、コンテナオーケストレーション（コンテナ群管理）に Red Hat 社が提供している「OpenShift[10]」を活用した。「OpenShift」は、ダッシュボードでの視覚的な Pod[11]の管理（死活監視、冗長化、無停止切替）や設定が行え、CI/CD[12]パイプラインの機能も備えている。

4.3.3. サービスメッシュ

サービスメッシュは、マイクロサービス同士が連携する通信を仲介し、制御するレイヤ。マイクロサービスアーキテクチャでは、サービス間の通信が頻繁に行われるので、サービスメッシュで通信を仲介しながら制御・監視を行わないと、サービス間通信で問題が発生した際に素早く障害解析が出来ない。

今回 POC では、サービスメッシュで監視・トレーシングを行う為、「ネットワークトラフィックの管理」「セキュリティ（暗号化・認証）」「監視とロギング」の機能を持つ、OSS[13]の Istio[14]を導入した。

4.3.4. ステートレスなアプリケーション

「ステートフル」なアプリケーションとは、以前のトランザクションやセッション情報を引き継ぎながら対話／応答するもので、「ステートレス」なアプリケーションとは、情報を

---

[10] Red Hat 社が販売しているコンテナ群を管理する為のソフトウェア（オーケストレータ）。Kubernetes をベースとしている。
[11] Kubernetes(OpenShift 含む)で、コンテナを管理するための最小単位であり、１つ以上のコンテナの集合体。
[12]「Continuous Integration／Continuous Delivery」の略。ビルド・テスト・デプロイを自動化し、継続的に開発を行う手法のこと。
[13]「Open Source Software 」。ソースコードを広く一般に公開し、誰でも自由に扱ってよいとする考えに基づいて公開されたソフトウェアのこと。
[14] OSS。オープンソースのサービスメッシュ・プラットフォームで、マイクロサービスが相互にデータを共有する方法を制御する手段を提供するもの。アーキテクチャはデータプレーン・コントロールプレーンに分かれている。データプレーンでは、環境内にサイドカープロキシをデプロイして Istio サポートをサービスに追加する。サイドカープロキシはマイクロサービスと並んで存在し、リクエストを他のプロキシ間で転送する。また、これらのプロキシはメッシュネットワークを形成し、マイクロサービス間のネットワーク通信を仲介する。コントロールプレーンはプロキシを管理および設定し、トラフィックをルーティングする。コントロールプレーンでは、ポリシーの適用や監視・測定情報の収集を行うコンポーネントも設定する。

引き継がずに「常に初めて接続する場合と同じ」状態で対話／応答をするものを指す。

マイクロサービスアーキテクチャにおいて、オートスケーリングなどを活用する想定の場合、ステートフルなアプリケーションだと同じサーバー／アプリケーションに接続しなければ情報連続性が切れてしまう為、スケールアウトで生成されたアプリケーションへの負荷分散が阻害されてしまう。その為、アプリケーションはステートレスに作成したい。電文の形としても利用を想定している HTTP(S)は状態を保持しないプロトコルで、その上で成り立つ Restful API[15]もステートレスなため、サービスアプリケーションもステートレスなものを配置する。

図 12 のように基本的にステートフルな処理はフロントエンドで行い、API-GW から先はステートレスな処理とする。

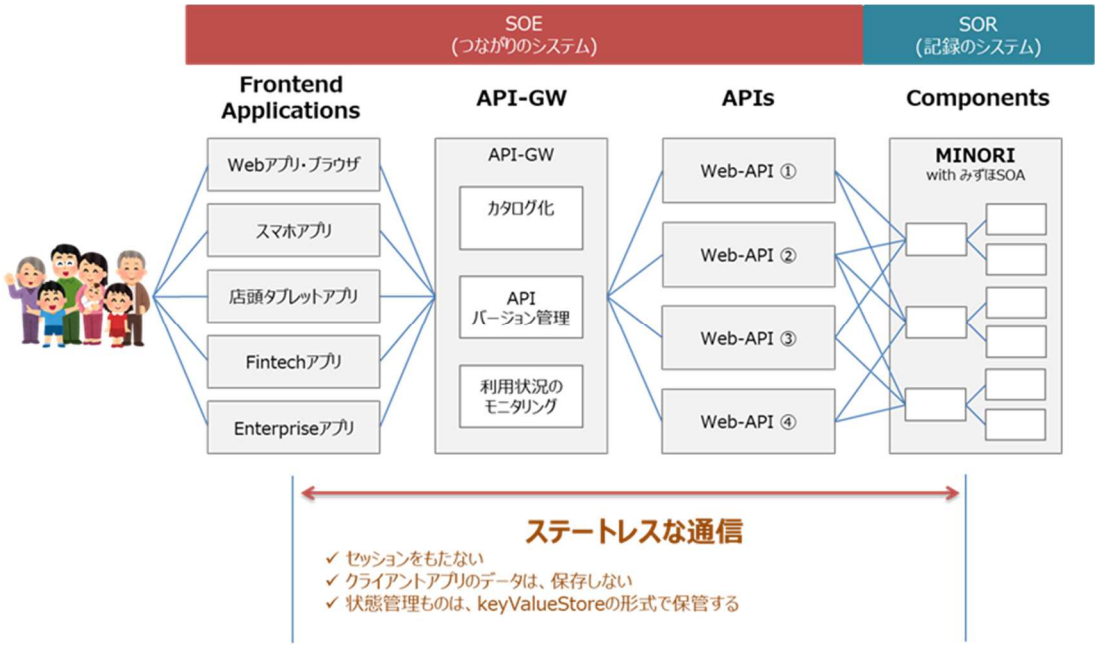

図. 12　ステートレスなアプリケーションイメージ

現状のオンラインバンキングにおいては「日跨り」や「為替口座照会結果」等、情報を引き継ぎながら処理を行う「ステートフル」なものがある。これらも、必要な状態管理処理は基本的にフロントエンドアプリケーションで作るか、どうしても必要な場合は、Process 層にデータ管理機能を入れる。

15　パラメータを指定して特定の URL に HTTP でアクセスすると、JSON 等で記述されたメッセージが送られてくるようなシステム、および、そのような呼び出し規約（「RESTful API」と呼ばれる）のことを指す。

4.4. POC(実現性検証)

4.4.1 検証内容

前述の D.A.L について、C 領域として「必要な堅牢性」と「激しい変化への対応必要性」の両方を兼ね備えた開発基盤となっているか検証するべく、POC を計画した。

C 領域のシステム特性を踏まえ、求められる可用性を満たしているか確認するため、「サービス安定稼働」を検証のポイントとした。また、C 領域では不具合・エラー発生時に影響が大きい事から、「サービスメッシュによる通信制御・監視」も検証のポイントとしている。加えて、リリースサイクルが短期になることから、OSS による開発を視野に入れ、「OSS フレームワークによる Java アプリ開発」という観点でも項目を追加した。（図 13 の通り。）

	技術検証のポイント			実施内容
A	**コンテナ技術による サービス安定稼働**	可用性		✓ アプリの異常停止時において、自動復旧させるための設定方法や、実際の復旧速度を検証
		性能・拡張性		✓ アプリのスケーリング方法、及びスケールアウトの速度を検証
		移行性		✓ **AWS**上で構築した**OpenShift**環境を閉域環境（別AWSリージョン）へ、同一のアプリ・サービスをデプロイ（移植）できるか検証
B	**サービスメッシュによる 通信制御・監視**	運用 保守性	通信の制御	✓ 各コンテナの通信を統合管理するサービスメッシュ技術について、関連機能を提供する **Istio** を導入し、その有用性を検証
			通信の可視化	✓ 各アプリ・サービスの障害インシデント発生時において、それぞれの通信を可視化させ、原因調査の迅速化を施す **Jaeger** 及び **Kiali** について、導入・利用方式を検証
			ログ・メトリクス収集	✓ アプリおよびコンテナから出力される各種ログについて、**Fluentd**（収集・転送ツール）による収集方法を確認・検証 ✓ また、みずほ現行監視システム（**JP1**）との連携を想定し、ログデータの伝送方法を確認・検証
C	**OSSフレームワークの Javaアプリ開発**	開発プロセス		✓ JavaのOSSフレームワーク（**SpringBoot**）、及びコンテナ技術を用いて、**IR社員のみで**アプリ開発を実施し、開発難易度を検証

図. 13　技術検証のポイント

検証にあたって、銀行 API の要件を想定し、サンプルとして『通常は別の事務手続きである「CIF 開設[16]」、「住所変更」、「結果照会」を一度に行うアプリケーション』を作成した。作成したアプリケーション群は図 14 の通り。色付きボックスはそれぞれ一つのサービスアプリケーションを表し、別々にデプロイしている。Experience 層には顧客に提供したい全体の機能を、System 層には銀行勘定系システムの API 制御機能を、Process 層には中間的に機能呼び出しをするサービスを配置。青線は、電文の流れを表現している。

想定としては、Experience 層は要件次第で都度新規作成されるが、System 層は銀行システムが提供する最小単位で構成され変化しないもので、Process 層は種々の Experience 層サービスの要件に応じて使い回されるものとしている。

[16] CIF は「Customer Information File」の略。顧客の基本属性項目を管理する顧客マスタのことで、このマスタに普通預金などの個別商品や各業務のマスタが紐付けられる。

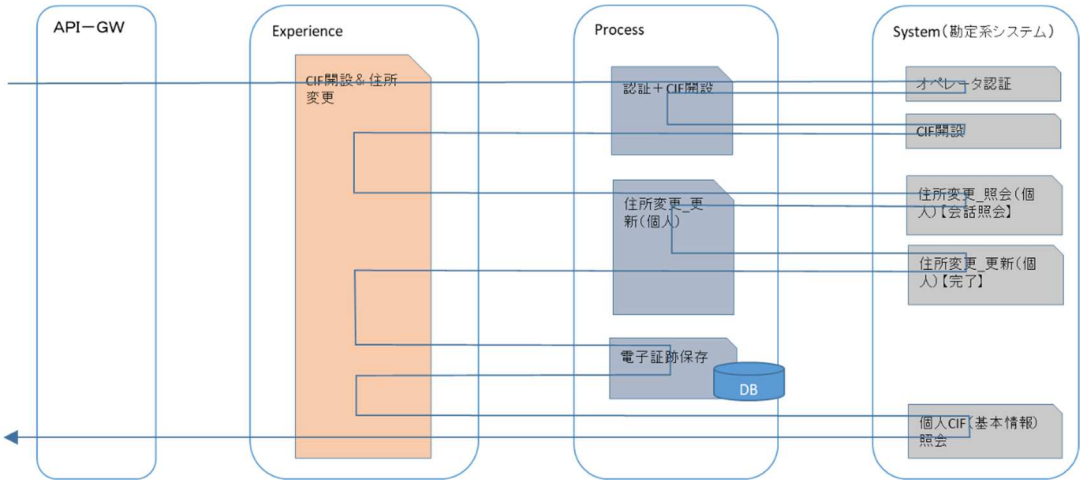

図. 14　「CIF 開設」、「住所変更」、「結果照会」を一度に行うアプリケーション

次に、サービスメッシュ(Istio による監視の仕組み)の形は図 15 の通り。サービスはそれぞれ一つの Pod(複数のコンテナで構成)でデプロイしており、Pod 間通信は各 Pod 内サイドカーコンテナとして封入した Envoy[17](ネットワークトラフィックを送受信するソフトウェア)を通して実施するよう、アプリケーションに組み込んだ。

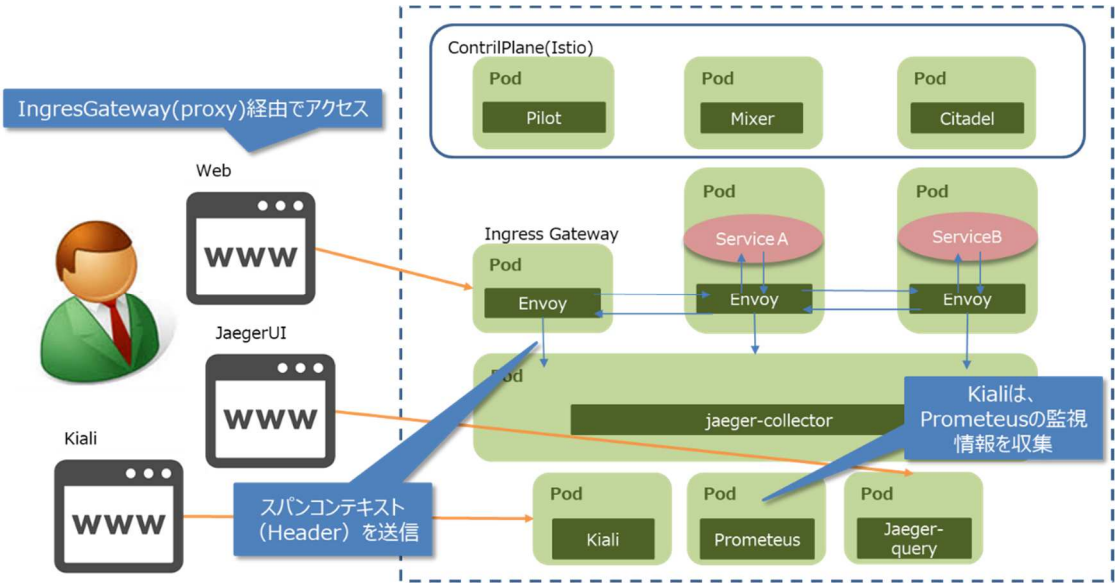

図. 15　Istio による監視の仕組み

[17] OSS。マイクロサービスと並んで存在し（サイドカー）、マイクロサービス間のネットワーク通信を仲介するソフトウェア。マイクロサービスのサービスメッシュを形成する。

サービスメッシュ外からの HTTP/TCP トラフィックは Ingress Gateway[18]を通る。

通信状況は、Jaeger[19]及び Kiali[20]で視覚的に確認可能。Jaeger では、Envoy サイドカーから送られてくる、通信のスパンコンテキストを以下のように表示できる（図 16)。

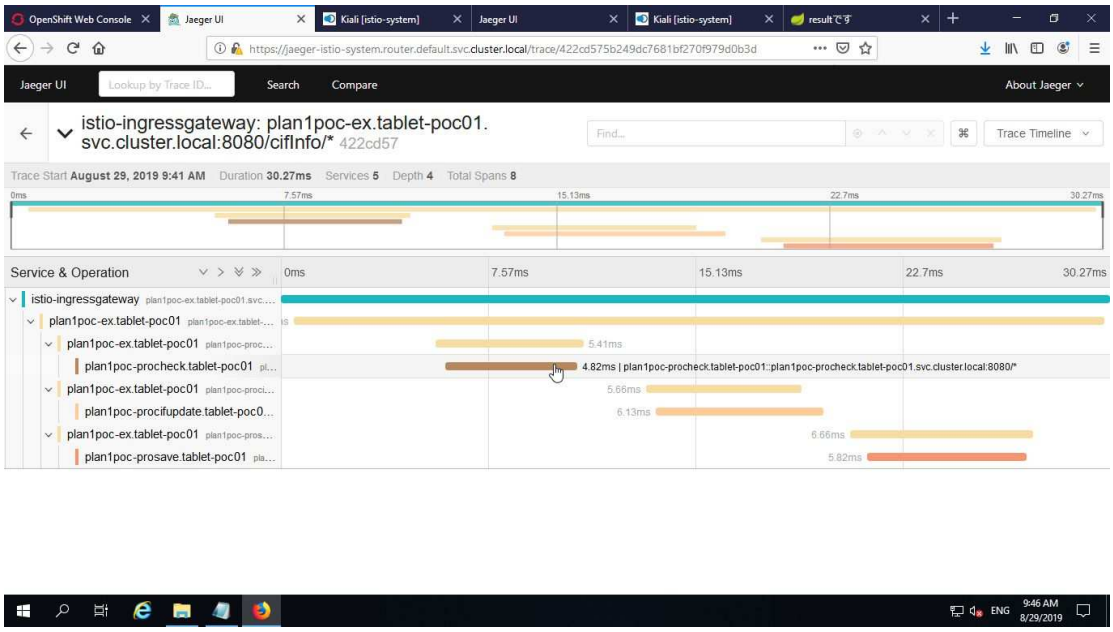

図. 16　Jeager を用いた通信状況の監視イメージ

Kiali でも、Jaeger の情報を以下のように表示できる。Prometeus[21]の監視情報も使って、メッシュトポロジーの判別、メトリクスの表示、健全性の算出、可能性のある問題の表示などを行う（図 17)。

[18] OSS。サービスメッシュの入り口で制御を行うソフトウェア。サービスメッシュ外から来た通信に対し、内部の各サービス URI(Pod)へ振り分ける制御を行う。

[19] OSS。分散サービス間のトランザクションをトレースするためのソフトウエア。サービスメッシュの一部。Envoy から送られてきた通信を集積し、可視化することができる。

[20] OSS。サービスメッシュの視覚的に観測することができるソフトウェア。Jaeger の通信情報と Prometeus の監視情報も使って、メッシュトポロジーの判別、メトリクスの表示、健全性の算出、可能性のある問題の表示が可能。

[21] OSS。監視ソフトウェア。Docker や Kubernetes といったコンテナ/クラスタ管理ツールとの連携機能もあり、容易に監視対象を設定できる。

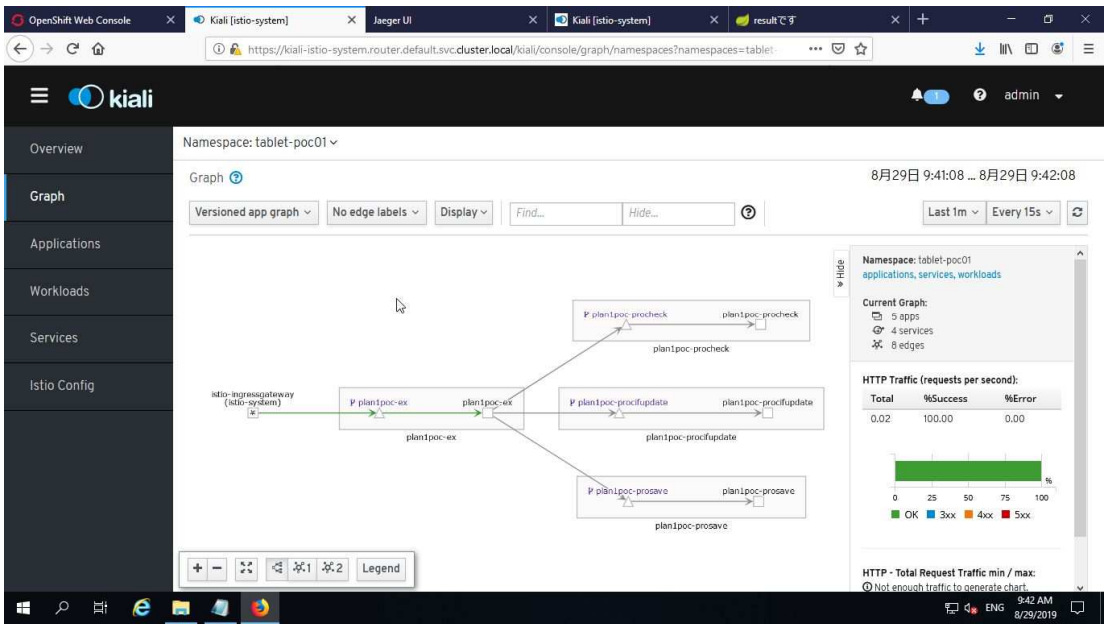

図. 17　Kiali を用いた通信状況の監視イメージ

4.4.2　検証結果の評価

今回の POC で、D.A.L が実現可能であり、運用面／開発効率面でメリットを享受できることが確認できた。

検証項目と照らし合わせて纏めたものが以下。

A)　コンテナ技術によるサービス安定稼働

以下の内容から、アプリダウン・急なアクセス増・ハード障害等、状況に応じて容易に対処が可能と評価。

- ✓ Pod 停止時に、自動復旧（オートヒーリング）が瞬時に行われることを確認。ダウン検知からリスタートする復旧リードタイムを削減可能であり、可用性に寄与するものと評価。
- ✓ トランザクション量の増減に伴うアプリ(Pod)の負荷状況にあわせたスケールイン／アウトがクリック操作で簡易に実現できることを確認。業務状況に応じて、より柔軟にリソース拡張可能であるものと評価。
- ✓ AWS 上で構築した OpenShift 環境を閉域環境（別 AWS リージョン）へ、同一のアプリ・サービスをデプロイ（移植）できることを確認。柔軟な環境変更が実現可能であると評価。

B) サービスメッシュによる通信制御・監視

- ✓ Istio（通信制御ツール）・Jaeger（通信トレーシングツール） ・ Kiali（監視・通信可視化ツール）を用いることで、各アプリの「挙動監視」「障害箇所特定」が簡易・迅速化できることを確認。システム監視において有用であると評価。
- ✓ コンテナ型仮想化でのログ監視において、Fluentd[22]（転送ツール）を用いて情報抽出し、syslog の形式で、みずほ現行監視システム（JP1[23]）へログ転送が可能であることを確認。みずほ現行監視システム（JP1）の利用も検討可能であると評価。

C) OSS フレームワークの Java アプリ開発

- ✓ SpringBoot (Java フレームワーク)及びコンテナ技術を用いることで、短期間で一定品質のアプリを実装可能であることを確認。今回アプリケーションは全て Spring Boot を使用して作成したが、別の言語・フレームワークでも作成が可能であり、機能別に分けられた弊社各部署がそれぞれ、新技術や慣れ親しんだ言語で参画できる仕組みとなっている。サービス開発の生産性向上に寄与するものと評価。
- ✓ OpenShift を活用すれば、CI/CD(コードコミット~自動テスト~デプロイのサイクル)が可能で、インフラ移行時の可搬性も保つことが可能である。また、サービスメッシュで「サーキットブレイカー」や「カナリアリリース」の設定も可能であり、安全なリリースが可能と評価。

5. 結論と今後について

課題となっていた「銀行業務機能の API 化」は、モノリシックであった勘定系システムをみずほ SOA の思想で刷新し、「機能のコンポーネント化」と「層の定義」によって固有処理をアプリケーションフロントエンド層に外だし集約したことで、「業務チャネル統合基盤」に API 制御機能を構築することができた。銀行業務の大半を占める営業店端末取引を API 化することが可能となった。

また、「使いやすい API 提供を迅速に実現する為のレイヤ構築」に対しては、D.A.L 構築の POC 実施により、既存システムに捉われず新しい言語・フレームワークで新規サービス構築を短期間・低コストで実現するための技術検証が完了した。

2 章でも課題定義した通り、次世代金融サービスの創出には、既存のシステム資産の再利用・再構築によるシステム開発の短期間・低コスト化が必要であり、そのためには、既存システム・機能を API エコノミーでいかに流通させていくかが重要となる。つまり、今後の開発においては『4 象限』（図．10 参照）における A 領域（MINORI（勘定系コア））をし

[22] OSS。ログ収集ソフトウェア。ログの収集方法やログの記録先などを柔軟にカスタマイズできる。
[23] 日立社が販売する統合システム運用監視ソフトウェア。みずほ銀行の運用監視センターで利用している。

っかりと保守しつつ、サービス開発の主軸を C 領域（チャネル系システム）へ移していく必要がある。

具体的には、オープン API の早期実現を優先して構築した「みずほダイレクト」や今回事例で紹介した銀行業務の API 化案件は、個別システムの中に API 制御機能を実装しているが、将来的に D.A.L へ移行することで、既存システムの制約を受けない、低コストで迅速な API 開発を実現したい。

一方で、D.A.L でサービスを構築し易くなる分、統制・管理が煩雑になっていくことも予想される為、サービス作りの要領・ルールの整備が今後必要となる。それらの解決により、銀行取引を使い易い API として、統制・管理可能な環境で提供することで、Fintech 企業との連携の強化・拡大を加速し、各種金融サービスを銀行口座と結びつけるプラットフォームビジネス、新たな金融サービスに繋げるべく検討を進めていきたい。