マイクロサービス開発への SOA 開発プロセスの拡張可能性の検討

宗平 順己
Kyoto ビジネスデザインラボ
t-munehira@kyoto-bdl.com

要旨

デジタルトランスフォーメーションの進展に伴い，SoE (Systems of Engagement)と呼ばれるタイプのアプリケーション開発の重要性が急激に高まってきており，先行する欧米では，マイクロサービスの利用がデファクト化してきている．

このマイクロサービス開発にあたっては，サービス設計の難しさが課題として共通認識されている．

このサービス粒度問題については，SOA 開発と同じ課題であることから，筆者らが開発した SOA 開発プロセスをマイクロサービス開発へと拡張できるのではないかと考えた．

本論では，SOA 開発プロセスが BPM からスタートすることから，マイクロサービス開発についても上流プロセスからの定義を行い，マイクロサービスの持つ特徴に留意して，どのように SOA 開発プロセスを拡張すればよいのかを検討する．

1. はじめに

デジタルトランスフォーメーションの進展に伴い，SOE SoE (Systems of Engagement) と呼ばれる新しいタイプのアプリケーション開発の重要性が急激に高まってきている．デジタルトランスフォーメーションの第一の目的は新たな顧客経験の提供にあるが，提供側も利用側も明確な要件が決まっているわけではなく，まずはサービスを提供して，ユーザの反応を見ながら素早く仕上げていくというアプローチがビジネス面，システム面両面において必要となる．

このため，デジタルトランスフォーメーション（以下 DX）へのチャレンジが先行する欧米では，マイクロサービスの利用がデファクト化してきている．

我が国においては，システム開発をアウトソーシングすることから，残念ながらまだまだウォーターフォール型の開発が多く，DX への対応が困難な状況にあるが，SIer の業界団体である JISA が 2018 年度になってマイクロサービス移行への必要性を強く主張する様になってきた．

しかしながら，このマイクロサービス開発にあたっては，サービス設計の難しさが課題として共通認識されており，モノリシックファーストという経験主義的なアプローチが提案されるなど，ソフトウェア工学的なアプローチがなされていない．

ところで，このサービス粒度問題については，SOA 開発においても同様の課題があり，筆者らはこの課題に対応するために SOA 開発プロセスを開発した．

そこで，この SOA 開発プロセスをマイクロサービス開発に拡張できるのではないかと考え，適用可能性について検討したので，その結果を報告する．

第 1 章では．今なぜマイクロサービスが必要となっているのかについて，背景としての DX の概念を整理し，その上で，DX 時代に求められるシステムの特性をアーキテクチャ面から整理し，マイクロサービス適用の必要性について整理する．

第 2 章では．マイクロサービスの開発における粒度問題について整理する．一般に指摘されている下流側からの粒度課題を確認するだけではなく，上流からのアプローチにおける課題についても整理する．DX の開発において，欧米ではサービスデザインから始まることがデファクト化していることから，日本ではまだ馴染みのない，サービスデザインからマイクロサービスへ展開するアプローチを紹介し，そこで課題になっている粒度問題について言及する．

第 3 章では，まず SOA 開発における粒度課題とその解決方法を紹介したのち，マイクロサービスの粒度問題との相違点を整理し，SOA 開発プロセスのマイクロサービス開発への適用に向けての拡張要件を整理する．

本論では実際の案件にて拡張した開発プロセスを検証するところまでには至っていないことから，今後の課題として，必要な検証項目について整理し，まとめとする．

2. 今なぜマイクロサービスが必要なのか

2.1. 背景としての DX

2.1.1 DX の定義

調査会社の IDC は次の様に定義している．「企業が第 3 のプラットフォーム技術を利用して，新しい製品やサービス，新しいビジネスモデル，新しい関係を通じて価値を

創出し，競争上の優位性を確立すること」

この第 3 のプラットフォームというのは IDC が提唱しているコンセプトで，「クラウド」「ビッグデータ」「モビリティ」「ソーシャル」の 4 要素によって形成される情報基盤のことである．

DX の先駆者として良く例に出されている Uber や Airbnb のビジネスモデルをみると，実にうまく第 3 のプラットフォームの 4 要素を使いこなしていることがわかる．

DX の定義については「The best understanding of digital transformation is adopting business processes and practices to help the organization compete effectively in an increasingly digital world.」という主張があり，道具としてのデジタル化ではなく，経営環境としてデジタル化をとらえていくべきであるという考え方がある．[1]

2.1.2 DX への取り組み目的

MIT Sloan が 2015 年秋に Deloitte と共に世界中の 3700 以上の CEO やマネージャに実施した調査において，図-1 に示す様に 90%近くがデジタルによって産業構造に破壊をもたらすと考えており，44%がそれに対する備えを進めていると回答している．[2]

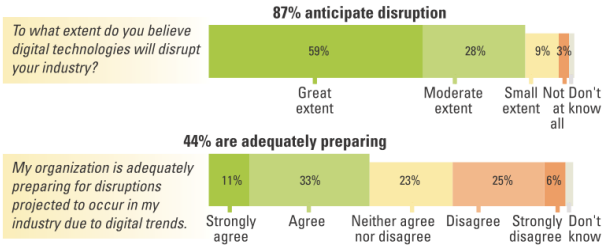


図-1 DX への対応

NoSQL サーバの提供元である Couchbase 社が 2017 年 5 月に米国，英国，フランス，ドイツで DX に積極的に取り組んでいる企業 450 社の CIO や CTO にオンライン調査を実施した．

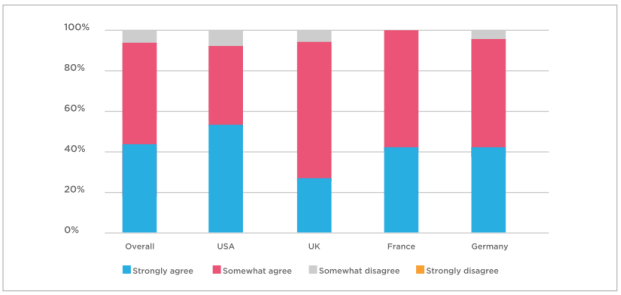


Figure 4: Do respondents agree that the ultimate aim of digital innovation should be to give customers and end-users a truly unique experience?

図-2 DX のゴール

図-2 に示すように，顧客やエンドユーザに今までとは異なる顧客経験を提供することが，DX のゴールであると回答している．[3]先に示した IDC の DX の定義では「新しい価値を創出し」と表現されているが，これはすなわち，新しい顧客経験を提供することであるといえる．

図-3 に示すように，同様の調査結果が，MIT SLOAN チームの 2017 年の調査においても示されており，いかに新しい顧客経験をデザインできるかが，課題といえる．

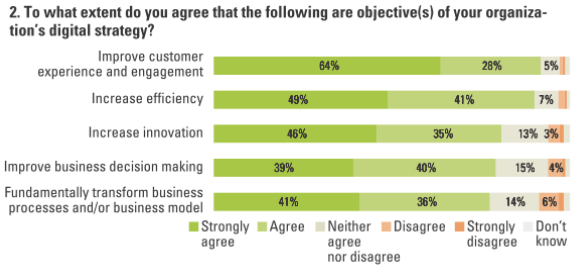


図-3 デジタル戦略の目的[4]

2.2. DX 時代に求められるシステムの特性

図-4 に示すペースレイヤーアプローチはガートナーが 2012 年に提唱したフレームワークで，主にシステムの変更頻度をもとにアプリケーションを分類するもので，ペースレイヤーごとに開発手法が異なる．[5]

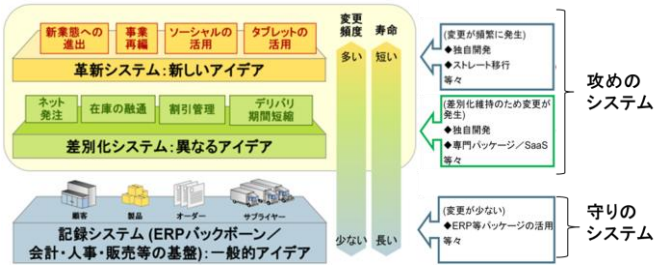


図-4 ペースレイヤーアプローチ 出典 5 を元に筆者が加筆

これまでの企業情報システムは，基幹システムと呼ぶ人が多い記録システム（Systems of Record）と戦略システムである差別化システム（Systems of differentiation）とで構成され，これらのシステム開発についてはノウハウの蓄積がなされている． 一方，フロントシステムである革新システム（Systems of Innovations）は，これらのシステムとは全く性格が異なり，最初に要件を定義することができない．差別化システムでも，記録システムほど確実性は高くはないが，企業の戦略が先に立案され，その実現に資するものとして要件が定義されている．（仮説ではあるが，それなりの自信を企業は持っている．）

これに対し，革新システムは何が正解かわからず，まずはリリースし，顧客や利用者の反応を見ながら，よりロイ

ヤリティを高めるために，素早く改良を加えるということが求められる．リスクを早期に潰す反復型のアプローチでも反復頻度が間に合わず，アジャイル開発が必須であり，リリース期間を短縮するために DevOps を利用し，スケールアウト／インを迅速に行うためにクラウドが必須となっている．なお，一般に SoE（Systems of Engagement）と呼ばれるシステムはこの革新システムを指す．

図-5 は航空会社の座席予約システムの構成をペースレイヤーアプローチと関連して説明したものである．[6]

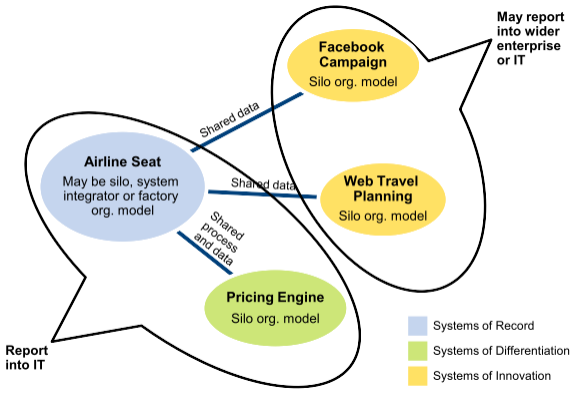


図-5 3つのレイヤーの連係モデル

DX のシステムは SoE であると書かれていることが多いが，実際にはこのように，3つのレイヤーのアプリケーションの組み合わせによって構築されることになる．

2.3. マイクロサービスの特性と DX での適用範囲

素早く改良を加えるということで多くの企業で使われてきているのが，マイクロサービスであり，その活用先進企業である Amazon は，1 年間に実に 5000 万回もシステム更新をかけている．

2.3.1 マイクロサービスの特徴

マイクロサービスの基本方針としては，マーティン・ファウラーが提唱している以下のものが共通認識とされている．[7]

＜マイクロサービスの基本方針＞

- 小規模なサービスを組み合わせてアプリケーションを開発すること
- 各サービスは，独立したプロセス上で稼働すること
- 各サービスは，REST などの軽量プロトコルで通信すること
- 各サービスは，自動的に，それぞれ個別にデプロイできること
- 各サービスは，それぞれ異なるプログラミング言語で実装可能であり， かつ，それぞれ異なる永続データストアを利用可能であること

独立性が高く入替が容易なサービス単位でアプリ ケーションを組み立て，リクエスト数の多いサービスが稼働するプロセスのみをスケールさせることができるようにすることも求められる．

マイクロサービスについてはまだ確立された仕様はなく，マーティン・ファウラーの9つのマイクロサービスの特徴が一種のガイドラインとして使われている．

＜マイクロサービスの9つの特徴＞

- サービスのコンポーネント化 - コンポーネントは独立して交換・更新可能なソフトウェアの単位である．
- ビジネス中心組織 -開発運用チームは技術や開発工程によるチームではなく，ビジネスを中心に機能横断型のチームが編成されている．
- プロジェクトではなくプロダクト - チームは開発完了とともに解散するプロジェクトモデルではなく，製品のライフサイクル全体に責任を持つ．
- スマートエンドポイントとダムパイプ - メッセージ通信は軽量かつシンプルであること．エンドポイントに高い機能を持たせ，通信はダムパイプ（メッセージのルータとしてのみ動作する単純な機構）であること．
- 分散統治 - 各サービスは独立したチーム，開発言語，ツールでアプローチする．
- 分散データ管理 - 概念モデルに関する分散だけでなく，データストレージも分散(サービス間で独立)する．
- インフラストラクチャの自動化 - テスト自動化/継続的デリバリー/継続的インテグレーションの導入
- 障害の設計 - 個々のサービスはいつでも失敗する可能性があるため，互いに障害を迅速に検出し，可能であれば自動的にサービスを復元できることが重要．
- 進化的な設計 - 頻繁に，迅速に，適切に制御されたソフトウェアの変更・廃止・構築を行うことができること．

2.3.2 DX での適用範囲

2.2 の図-5 に示したように DX のシステムは3つの異なる特性のシステムの組み合わせによって構築される．

このうち，マイクロサービスの特徴は，革新システム（Systems of Innovations）に求められるシステム要求に良く合致するものであることから，革新システム部分の開発

にあたってはマイクロサービスを適用することが最善の選択であると考えられている．

顧客起点でサービスを素早くブラッシュアップするAmazonが，マイクロサービスを採用していることがその一つの例証である．

3. マイクロサービスの開発における粒度課題

3.1. 下流側での粒度課題

マイクロサービスの適用にあたって課題となっているのはサービスの粒度である．独立性を高めようとすると，小さな処理単位でサービスを構築することが良いように思われるが，そうすると，各サービスは独立して稼働していることから，サービス間の通信が非常に多くなり，システムのパフォーマンスに深刻な影響を与えることになる．一方で，この問題を避けるために粒度を大きくすると，変更時にサービス内での影響調査の増大とアプリケーション構造の複雑さを生み，結局モノシリックシステムを構築するのと同じことになり，更新速度を著しく低下させることになる．

しかしながら，このサービス粒度については，現在ガイドラインがなく，サービスの粒度がバラバラになり，全体としての保守性を著しく損なうことになる．

この課題に対し例えば Amazon では10人規模で担当できる範囲をサービスの粒度とするとかモノシリックファーストというような考え方も提示されているが，いずれもトライ＆エラー型であり，工学的課題解決にはなっていない．

クラウドネイティブでアジャイル開発の決定打ともいえるマイクロサービスではあるが，このサービス粒度課題についての解決が必須となる．

3.2. 上流から下流への展開における課題

3.2.1 上流プロセスで使われるサービスデザイン

DX の目的は新たな顧客経験（CX）を提供することにある．その顧客経験の設計において広く採用されているのが，サービスデザインと呼ばれるアプローチである．

顧客経験の設計において最も重要なことは顧客インサイトを得ることであるが，具体的にはどのように得れば良いのであろうか．サービスデザインの先駆けであるイギリスのサービスデザイン会社 Engine のプロセスは図-6 の様に紹介されている[8]．このプロセスは一般にダブルダイヤモンドと呼ばれている．

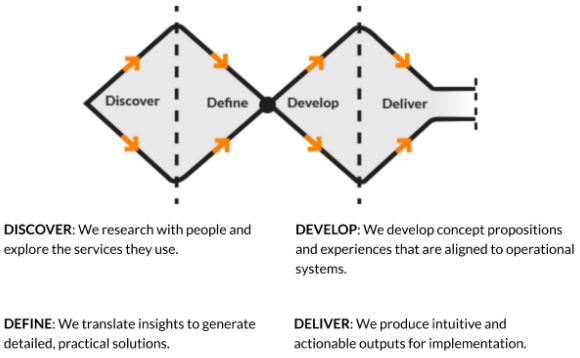


図-6　Engine 社のサービスデザインプロセス

「THIS IS SERVICE DESIGN THINKING [9]」も参考にしながらサービスデザインの具体的な進め方をまとめたのが図-7 である．

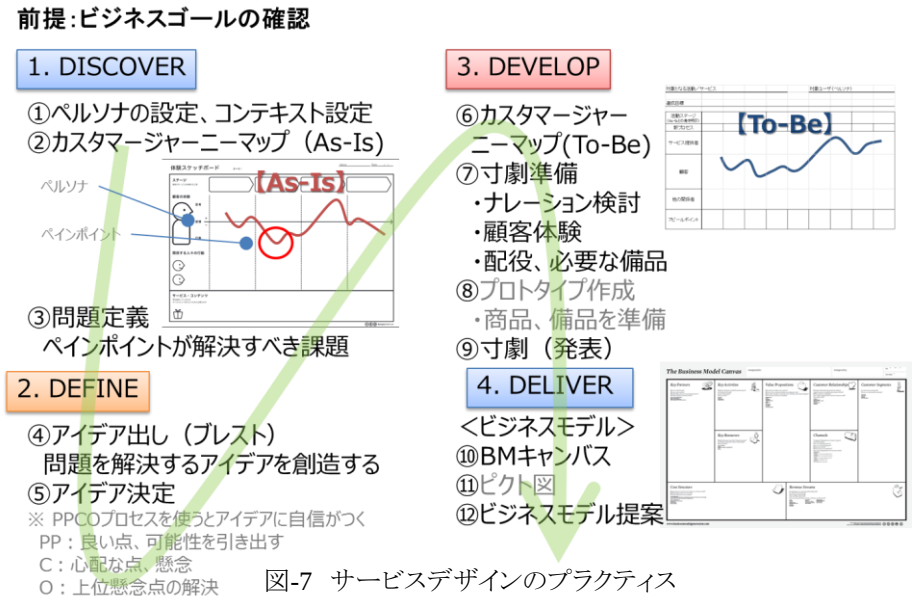


図-7　サービスデザインのプラクティス

まず前提となるビジネスゴールを全員で確認後，1. DISCOVER では，対象となる顧客（ペルソナ）の問題を認識することに専念する．この時に良く用いるのがカスタマージャーニーマップ（As-Is）であり，ペルソナが一連の活動において，どのような良い経験，悲しい経験をしているのかを書き出す．行動観察を行うと第三者でも共感しやすくなる．図にあるように顧客が不愉快，苦痛，不便と感じているペインポイントが解決のターゲットとなる．

2. DEFINE では，ペインポイントを鳥瞰して解決のアイデアだしを行う．最初は個別のペインポイントに着目したアイデア出しとなるが，次の DEVELOP と組合せてアイデアをブラッシュアップする．

3. DEVELOP は試作のステップであるが，まずは創出したアイデアを採用した場合の新しいカスタマージャーニーマップ（To-Be）を作成する．そしてその新しいシナリオをロールプレイやプロトタイピングなどの手法を用いて評価する．ペルソナの代表に参加してもらうのであるが，良い評価が得られない場合は，2. DEFINE に戻ってアイデアの再検討を行い，再び DEVELOP を実施する．

数度の Try & Error を繰り返し，良い評価が得られそうだということになれば，4. DELIVER の事業プランの作成にとりかかる．

その最初に行うのがビジネスモデルキャンバスの作成である．必要に応じてピクト図でビジネスの流れを補足して，スポンサーや関係者に対して事業プランの説明を行う．OK が出れば，具体的な組織設計や業務設計，システム設計など構築フェーズに入ることになる．

3.2.2 サービスデザインからマイクロサービスへの展開

3.2.2.1 サービスブループリント

カスタマージャーニー(To-Be)は顧客接点を中心に設計していることから，background プロセスも含めて実現可能な解とするため，図-8 に示すサービスブループリントを使って，詳細化を行う．

Actions はカスタマージャーニーで設計したものと同じであるが，顧客接点では具体的にどのようなデバイスを用いるのか，それを提供するのは誰（何か），そのためにどのような処理をバックグラウンド側で行うのかなどを設計するようになっている．

ここに示す例でも，フロント部分は SNS やサイネージ，Web サイトなど様々なシステムで，バックグラウンドとして業務システム等が関連しており，異なる Pace layer のシステムが連携している．

DX 時代になって，フロント部分にアプリ等を使ったセルフサービス型のモデルが導入され，パーソナライズ化のために戦略システムが組み込まれるなど，よりシステム依存度が増すようになってきている．

そのため，サービスブループリントの新たな拡張が提案され，実際の適用例もでてきている．

その内容を次に紹介する．

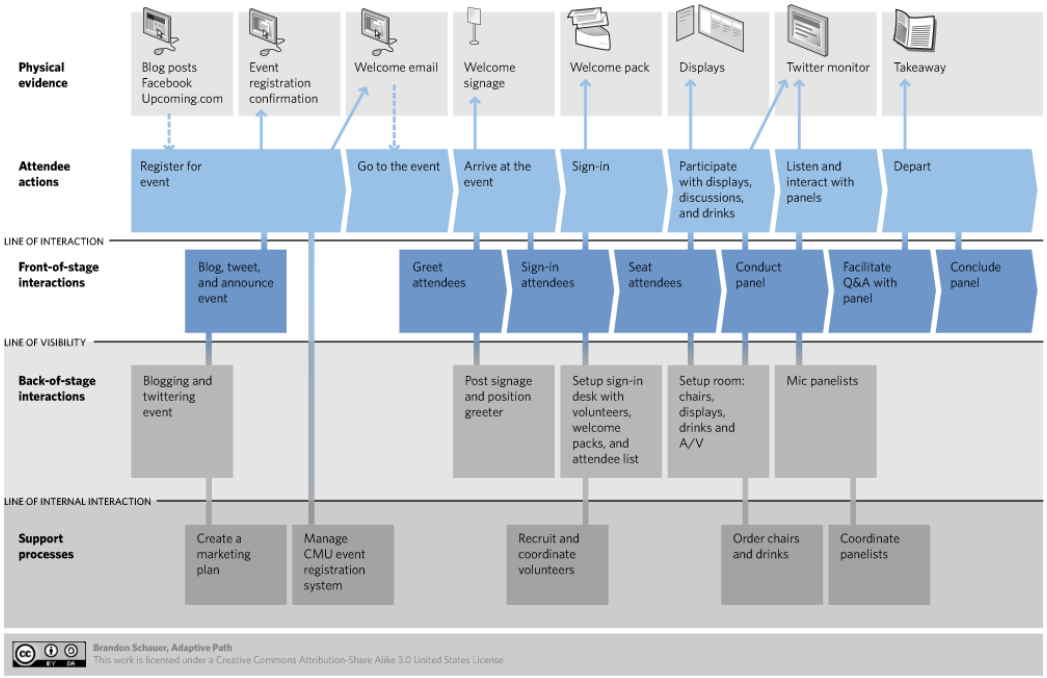


図-8 サービスブループリント(イベントマネジメントでの例)[10]

3.2.2.2 サービスブループリントからマイクロサービスへ

ダブリンの Wipro Digital からマイクロサービス環境を前提としたサービスブループリントの拡張が「Rethinking Service Blueprints for Agile Delivery」として提案されている. [11]

彼らの提案するサービスブループリントの縦方向の構造について図-9 に示す.

最上位にはカスタマーのアクションと関連するサービス提供側に関連するアクター(ロール)がリストアップされており, そのスイムレーンには, カスタマーやサービス提供者等がとるアクションに対して行うアクティビティを記述する.

その下段にある System Functionality がユーザーストーリーを記述するところになる. Develop フェーズでプロトタイプを作成し, おおよそのアーキテクチャ設計も済ませていることから, この例ではそこで定義した具体的なシステムも記述されている(The Hub と Tableau Reports).

記述されたユーザーストーリーを受けて, フロント側の機能を記述し, その実現のために利用するマイクロサービスとその機能をサービスレイヤーに記述する. 既存のマイクロサービスでは機能が不足する場合には, API 層に必要となる新たなサービスに求められる機能を記述し, DATA 層にはその新サービスが DB に対してどのような操作をするのかを記述するようになっている.

3.2.3 サービスマッピングにおける粒度問題

フロント機能とマイクロサービスとのマッピングにおいて課題となるのがサービスの粒度である. マイクロサービスの粒度が大きすぎても小さすぎてもマッピングが難しくなり, 適切な粒度でマイクロサービスが定義・登録されていることが必要となる.

4. SOA 開発プロセスのマイクロサービス開発への適用

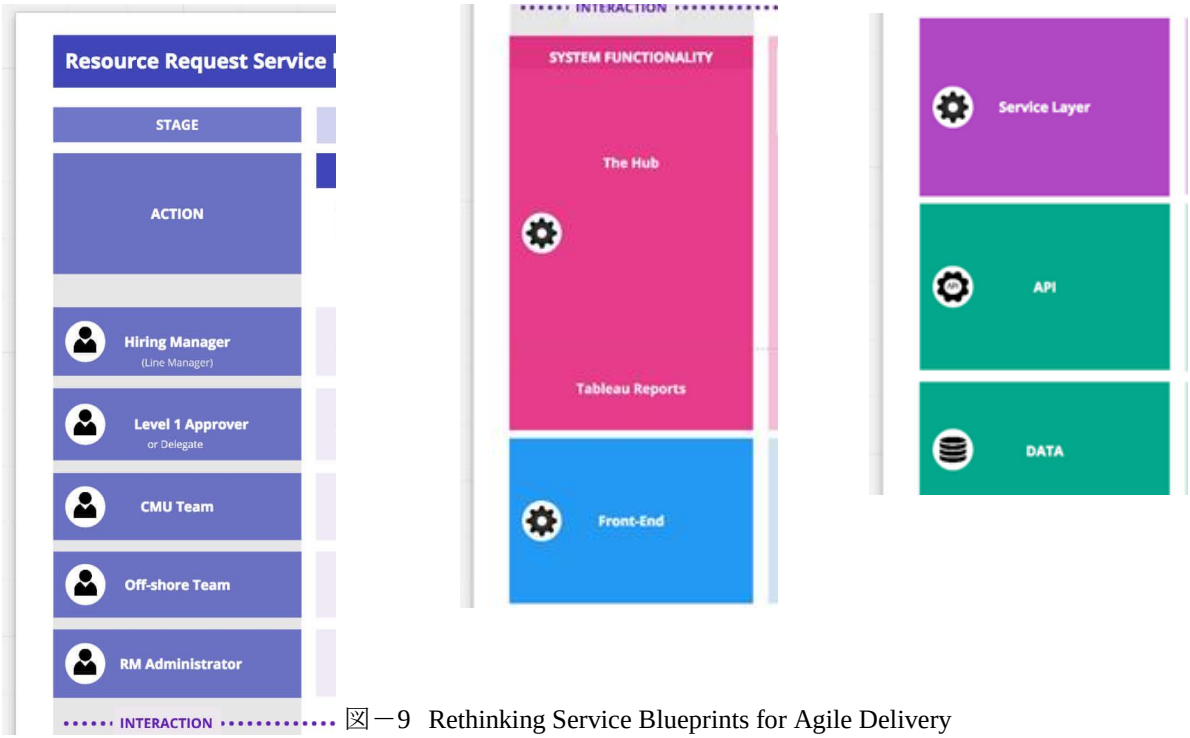
4.1. SOA 開発における粒度問題とその解決方法

4.1.1 SOA 開発における粒度問題

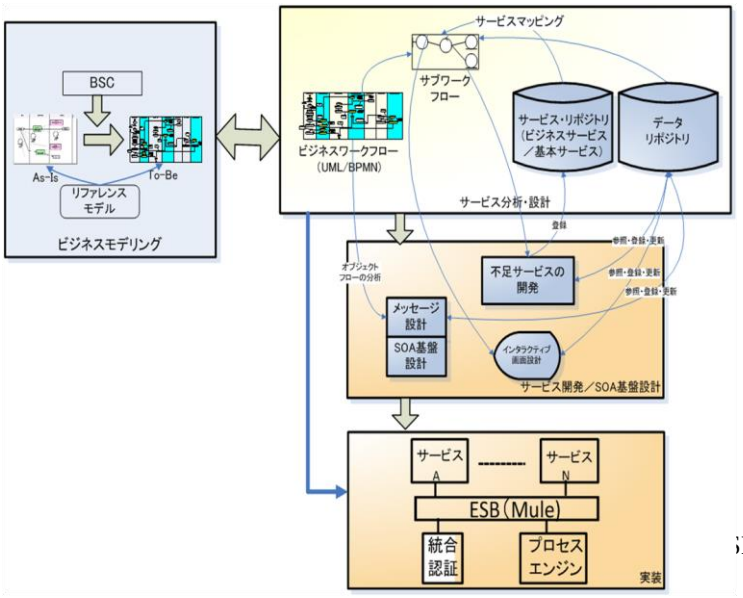
特定の業務領域を対象とした SOA 導入を進めた場合, 粒度問題というものは意識されることはない. 一連の業務フローをひとくくりのサービスとして定義しようと, 認証サービスといったように特定の機能をサービスとして定義しても, 特定業務領域内では問題とはならない.

粒度問題が明確化するのは, 複数の業務領域をまたがるシステムを構築しようとする時である.

業務領域によって, サービスの粒度が異なると事実上プロセス連携の設計はできないといった深刻な問題が生じる. ましてや, 企業間連携となるとこの問題はさらに深刻となる.



図－9 Rethinking Service Blueprints for Agile Delivery



ESB: Enterprise Service Bus

図-10 SOA の開発モデル

4.1.2 考案した開発モデル

4.1.2.1 開発モデルの概要

粒度問題を解決するために設計した SOA(Service Oriented Architecture)の開発モデルを図-10 に示す. [12]

この開発モデルは以下のオーソドックスな SOA アプローチを前提としている.

① プロセス階層を配慮しながら経営戦略を実現する To-Be(あるべき)ビジネスプロセスを設計する.
② レベル 3 のプロセスについて業務フローを作成する.
③ 業務フローのアクティビティをもとにサービスの括りだしをする.
④ 括りだしたサービスに対して既存のサービス群から割り当てができる場合には, 直ちに新しいビジネスプロセスを動かすことができる.
⑤ サービスが不足する場合には, そのサービスを実現するコンポーネントの開発のみを行う.
⑥ アクティビティ間のメッセージをもとにデータ設計を行う.

サービス粒度を揃えるために, 以下の4つの工夫を加えている.

① 詳細プロセスリファレンスを用いてビジネスプロセスモデルを設計して, ビジネスプロセスの粒度を揃える
② システムレーンのアクティビティをサブアクティビティ図を用いて BCE モデルに分解し, コントロールアクティビティに対してサービスをマッピングする)
③ 類似のサービスを構築しないようにリポジトリィによるサービス管理を行う.
④ サービスを, 業務フローをモデル化したプロセスサービス(BPM エンジンの読み込み対象となる), プロセスサービスから呼び出され DB への処理等を行うビジネスサービス, 認証認可などの基本サービスの3つに分類し, リポジトリィに登録し, 再利用の対象とするのはビジネスサービスと基本サービスのみとする. 粒度の大きいプロセスサービスを再利用性は低いからである.

4.1.2.2 サービスマッピングの方法

ビジネスモデラーが行うサービスマッピングは以下の拡張を UML モデリングツールに加えている. (図－11)

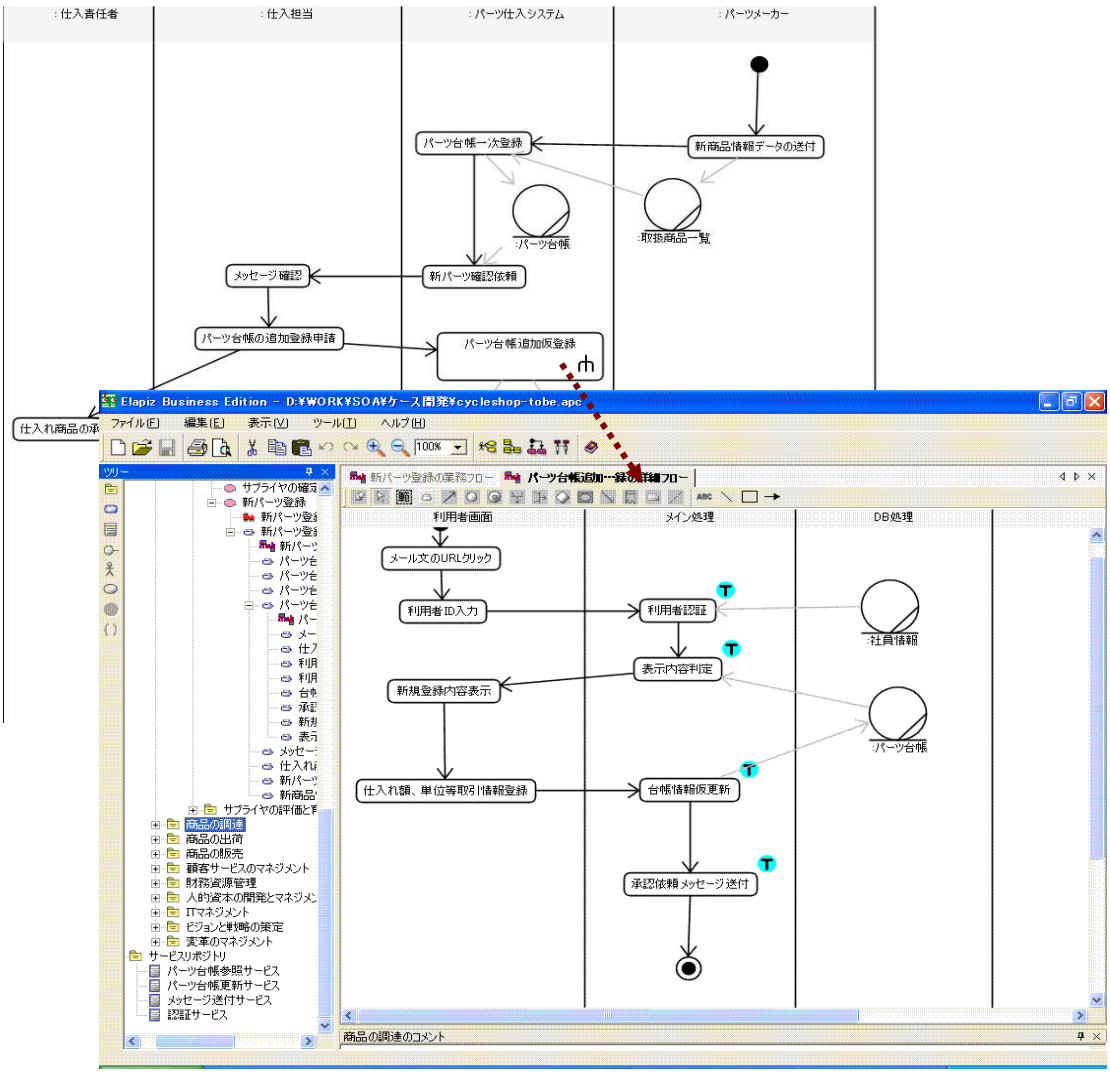
①サブアクティビティ図のフォーマット定義

通常ビジネスプロセスモデルを用いてシステムの要件定義を行う際, アクティビティ図のシステムのスイムレーンに示されるアクティビティをシステムユースケースとし, 分析モデル(BCE クラス図)を作成している.

クラス図ではフローを表記できないので, サブアクティビティ図を用いることとし, 図－11 に示すようにスイムレーンに BCE を割り当てた.

②トレース機能

アクティビティとサービスとの関係定義が必要となる. 図－11 のツリー図に示しているサービスリポジトリに登録されているサービスクラスとサブアクティビティとの結びつきを定義できるようにした. (Tマーク)



図－11　UML モデリングツールを使ったサービスマッピング

4.1.2.3　サービスマッピングからサービス設計の手順

ビジネスモデラーが実施したサービスマッピングを受けで下記の手順でサービス設計を実施する.

1.サービス分析
- サービス候補分析
- 概念モデル設計
- サービス候補の洗練
- サービス分析
- メッセージ分析
- サービスプロトコル分析

2.サービス設計
- サービス設計
- メッセージ設計
- サービス定義書作成

この中でサービスの粒度調整を行っているのが，サービス候補分析である. ここでの作業内容は下記のようになる.

「抽出されたサービス候補を整理する. このときの観点はサービスの粒度を揃えることと，再利用性である. この作業を行う際の指標となるのが，ビジネスエンティティに対する CRUD(create, read, update, delete)操作である. 例えば，複数のサービス操作候補から同じビジネスエンティティへ参照(Read)がある場合は，1 つのサービスにまとめる. サービス操作候補が複数のビジネスエンティティに操作を行っている場合，そのサービス操作候補は 1 つのサービスとし，複数のサービス操作に分割する.」

この過程を経て，工夫の四つ目に示したサービスを3つのグループに分けて定義することとした.

4.2. マイクロサービスの粒度問題との相違点

4.2.1 下流の粒度問題との相違点

4.1.2.3 及び 4.1.2.3 で示した手順は，サービスの独立性を高める手順である．この問題意識は 2.1 に示したマイクロサービスで懸念されている粒度問題と同じものである．

機能単位ではなく，画面フローに展開してからサービス候補が示され，ビジネスエンティティに着目して粒度を決めていることから，適切な粒度で設計するという要求にそのまま適用できるものではないかと考える．

4.2.2 上流の粒度問題との相違点

両者とも下記の共通のアプローチでサービスマッピングを行っている．

- サービスライブラリィを前提としている
- 上流で定義されたアクティビティに対し既存サービスをマッピングする
- 適当なサービスがない場合は，新たなサービスを開発する
- フロント，バックエンド，DB の処理を定義している

マッピングの際に適切な粒度で定義されたサービスライブラリィが欲しいという要求は共通している．

最も異なるのが，SOA はビジネスプロセスモデルを基にしていることであり，マイクロサービスはサービスブループリントをベースにしていることである．

SOA でサービス粒度を揃える起点は，ビジネスプロセスモデルを定義する際に，詳細プロセスリファレンスモデルを参照することである．この詳細プロセスリファレンスモデルは，ポーターのバリューチェーンモデルを第一階層として，ビジネスプロセスを最大第 4 層まで展開し定義したものとなっている．

サービスブループリントは顧客のタッチポイントを切り口としてプロセスを展開しており，この上流のモデリング手法の違いが適用に当たっての検討課題となる．

4.3. 適用に向けての拡張要件

4.3.1 適用に向けての方針

サービスブループリントの元になる To-Be のカスタマージャーニーマップは，複数の業務をまたがるアクティビティ図とほぼ同様のものとなっており，内容的には同一視して良い．

サービスブループリントからのアプローチでは，実はどのようにマイクロサービスをマッピングするのかが不明瞭である．SOA のアプローチでいうとビジネスプロセスモデルのシステムのレーンのアクティビティに対して直接サービスをマッピングしようとしていることと同様のことをしようとしている．

SOA 開発における粒度コントロールの鍵は，リファレンスモデルを参照して，業務フローを作成するビジネスプロセスの粒度を揃えることと，システムレーンのアクティビティを BCE モデルに展開し Control にサービスをマッピングするところにある．

マイクロサービスの設計において，粒度を整えるには，SOA 開発におけるこの2つの粒度コントロールを取り込むところにある．

4.3.2 具体的な拡張方法

前述のように，マイクロサービス様に筆者らの開発した SOA 開発プロセスを拡張するのではなく，粒度コントロールの2つの要素をサービスデザインから始まる開発プロセスに組み込むことが求められる．

筆者らの方法論は実はオリジナルの UML モデリングツールを前提として成立している．

このため，具体的な拡張は，以下の2つの観点で

① リファレンスモデル参照プロセスの組み込み

これについては，サービスブループリントのデザインにおいて，バックエンド側のプロセスを都度設計するのではなく，バックエンド側については，SOA の方法論を用いてビジネスプロセスモデルを予め設計しておくことで対応する．

② BCE 展開プロセスの組み込み

これについては，サービスブループリントを UML のアクティビティ図で記載できるように UML モデリングツールを拡張する．サービスブループリントは独自のレイヤー構造を持つが，これをスイムレーンにおいて表現できるようにすれば，アクティビティの BCE 展開，サービスマッピング，サービス設計のプロセスへと進めることが可能になる．

横書きと縦書きの違いはあるが，構造が類似しているので，拡張は難しくないと考える．

5. まとめ

本論では DX において必要とされるマイクロサービスベースの開発に対して，日本ではほとんど議論されていない上流のプロセスについてサービスデザインを適用し，サービスブループリントを経由して，マイクロサービスにつなげる手法を紹介し，その詳細な検討に筆者らが開発し

た SOA 開発プロセスを組み込むことで，課題となっているサービス粒度問題をも解決できそうであることを示した．

次ステップとして，検討結果を反映した UML モデリングツールの拡張を行い，マイクロサービス開発を実際に実施することでその妥当性を検証することとしている．

なお，マイクロサービスの利用にあたっては，実装環境の整備の課題がある．図-5 に示したように，異なるレイヤーのシステムを相互接続させる必要があるが，この点についての具体的なアーキテクチャの設計が求められる．ベンダーからの提案ではあるが，図-12 に示すような相互接続アーキテクチャが mulesoft 社から示されている[13]．実際の開発にあたっては，実装環境も大きな影響を与えることからジェネリックなモデルの検討も進めたい．

参考文献

[1] Gerald C. Kane, “Digital Maturity, Not Digital Transformation”, http://sloanreview.mit.edu/article/digital-maturity-not-digital-transformation/?article=digital-maturity-not-digital-transformation&post_type=article

[2] Gerald C. Kane, Doug Palmer, Anh Nguyen Phillips, David Kiron, and Natasha Buckley, “Aligning the Organization for Its Digital Future”, MIT Sloan Management Review RESEARCH REPORT SUMMER 2016.

[3] https://info.couchbase.com/2017_CIO_Survey_Report_LP.html

[4] Gerald C. Kane, Doug Palmer, Anh Nguyen Phillips, David Kiron, and Natasha Buckley, “Achieving Digital Maturity”, MIT Sloan Management Review RESEARCH REPORT SUMMER 2017.

[5] https://www.slideshare.net/jeffshuey/pace-layered-approach-and-winshuttle-share-point-conference-nov-2012-jeff-shuey

[6] Mary Mesaglio, Matthew Hotle, “Pace-Layered Application Strategy and IT Organizational Design: How to Structure the Application Team for Success”, Gartner, 2016.7 https://www.gartner.com/binaries/content/assets/events/keywords/applications/apn30/pace-layered-applications-research-report.pdf#search=%27Pace Layered%27

[7] Fowler, Microservices, https://martinfowler.com/articles/microservices.html, 2014/3/25

[8] http://enginegroup.co.uk/approach/, 2015/05/01

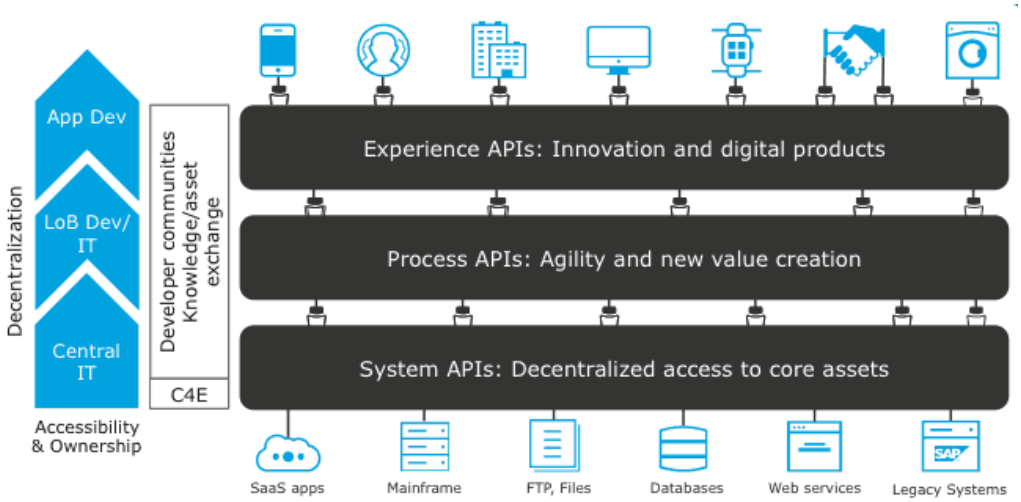
[9] マーク・スティックドーン，ヤコブ・シュナイダー ，長谷川敦士，武山政直，渡邉康太郎（監修），郷司陽子 （翻訳），「THIS IS SERVICE DESIGN THINKING. Basics − Tools − Cases - 領域横断的アプローチによるビジネスモデルの設計」，ビー・エヌ・エヌ新社，2013

[10] https://www.flickr.com/photos/brandonschauer/3363169836/sizes/o/

[11] https://wiprodigital.com/2018/08/30/rethinking-service-blueprints-for-agile-delivery/，2019/3/7 に参照

[12] 齋藤伸也，宗平順己，大場克哉，「UML モデルを活用した SOA の設計・実装プロセスの検証」，研究報告ソフトウェア工学（SE） 2009-SE-166(15)，1-8，2009-10-29

[13] https://blogs.mulesoft.com/dev/api-dev/what-is-api-led-connectivity/



図―12 DX 対応のアーキテクチャ例