B-006

分割サブドメインに基づくマイクロサービス設計
A Design of Microservices Decomposed by Divided Subdomain

鹿糠 秀行[1)] 木下 崇央[1)] 東 岳人[1)] 田中 優利[1)] 太田 聡[1)]
Hideyuki Kanuka Takahiro Kinoshita Taketo Higashi Yuri Tanaka Satoshi Ota

1 はじめに

マイクロサービスアーキテクチャは，アプリケーションを独立した規模の小さいサービスに分割して実装するアーキテクチャスタイルである．機能追加や変更などアプリケーションの保守容易性を高める手法として，様々な業務のアプリケーション開発において導入が進んでいる [1].

業務アプリケーションを適切なマイクロサービスの単位に分割する基準の一つとしてサブドメインによる分割 (Decompose by Subdomain) パターン [2] が知られている．このパターンは，業務アプリケーション開発対象に対してオブジェクト指向のドメインモデルを設計し，さらにドメインモデルをサブドメインとして境界で区切られた単位を基準としてマイクロサービス化することを説明している．サブドメインは業務に対応した単位で定義され，業務要求の追加や変更によって発生する業務ロジックの追加や変更による影響を，サブドメインに対応したマイクロサービスに閉じることが期待できる．

その一方で，最初に設定したサブドメインに対して，業務要求は時間を経るに従って追加され，同時にドメインモデルも進化する．ドメインモデルの進化に準じて実装されるマイクロサービスの業務ロジックも追加され，マイクロサービスが複雑化して肥大化し，保守容易性が逆に低下してしまう可能性がある．

そこで，業務要求に対してサブドメインを共通部分と可変部分に分けた上で，共通部分に対して基底サブドメインと可変部分に対して派生サブドメインを定義し，分割サブドメインの単位に対応してマイクロサービスを分割する方法を提案する．既存のマイクロサービスに影響を与えることなく，新しく追加される業務要求に対応した業務ロジックの追加がしやすくなる．

本稿では，マイクロサービス分割方法として分割サブドメインによる分割 (Decompose by Divided Subdomain) パターンと，この設計と実装の中で必要となる派生マイクロサービスオーケストレータを提案する．

2 分割サブドメインによる分割パターン

以下ソフトウェアパターンの形式と図 1 の例で説明する．関連パターンについては，独立した項目を設けずに，説明の中で必要に応じて参照する．

2.1 文脈

業務要求の追加に伴い，業務アプリケーションのサブドメインに対応して実装されたマイクロサービスが大きくなっていく．これは，マイクロサービス化のメリットである保守容易性を低下させる原因となり得る．

2.2 問題

業務要求に対応した共通部分の基底サブドメインと可変部分の派生サブドメインを，保守容易性の問題を考慮してどのようにマイクロサービス化すべきか．

2.3 解決策

基底サブドメインと派生サブドメインへのマイクロサービス分割方針として，以下 2 つの方針を提案する．

2.3.1 設計方針 A について

派生サブドメインの単位で基底サブドメインをセットにしてマイクロサービス分割（これを派生マイクロサービス (DMS) と定義）する．DMS には基底サブドメインに対応した業務ロジックや DB の一部が重複して実装されることになる．また，DMS をオーケストレーションするマイクロサービス (これをマイクロサービスオーケストレータ (DMO) と定義) を作成する．

各分割サブドメインに含まれるドメインモデルは，業務アプリケーションの実装においてリレーショナル DB を採用する場合にテーブルへマッピングされる．DB も含めたマイクロサービスには高い独立性が求められるためサービス毎に DB を保持する (Database per Service) パターン [1] ではマイクロサービスごとに業務ロジックとセットで DB を持つことが推奨されており，この場合に DMS の DB には基底サブドメインに対応したドメインモデルを構成するクラスからマッピングされるテーブルも持つことになる．

なお，基底サブドメインに対応したクラスと派生サブドメインに対応したクラスの間に継承関係があるために，具象テーブル継承 (Concrete Table Inheritance) パターン [3] を適用する．

2.3.2 設計方針 B について

派生サブドメインの単位で DMS を作成し，基底サブドメインに対応して DMO と対応した DB を作成する．DMO には DMS のオーケストレーションに加えて基底サブドメインに含まれるドメインモデルに対応した業務ロジックや DB も実装する．

なお，派生サブドメインと基底サブドメインに含まれるドメインモデルを構成するクラスから各 DB のテーブルへのマッピング方式は，クラステーブル継承 (Class Table Inheritance) パターン [3] を適用する．

2.4 結果

設計方針 A のメリットは，業務要求に対してサブドメイン間の共通性を意識せずに，DMS を独立に追加や修正できることである．逆にデメリットは，派生サブドメイン間の差分が小さい場合に，DMS がクローン化することで新たな保守性の問題が発生する可能性がある．

設計方針 B のメリットは，業務要求に対してサブドメイン間の差分だけを DMS として独立に追加できることである．逆にデメリットは，派生サブドメイン間の差分が小さい場合に DMO が大きくなっていくことや，DMO の修正が派生ドメイン全てに影響を及ぼすため，保守容易性の問題を解決できない可能性がある．

両方針共のメリットは保守容易性以外に，1 コンテナに対応してマイクロサービスを実装（Service as a Container）パターン [2] を適用し，DMO と DMS を実

1） (株) 日立製作所 Hitachi, Ltd.

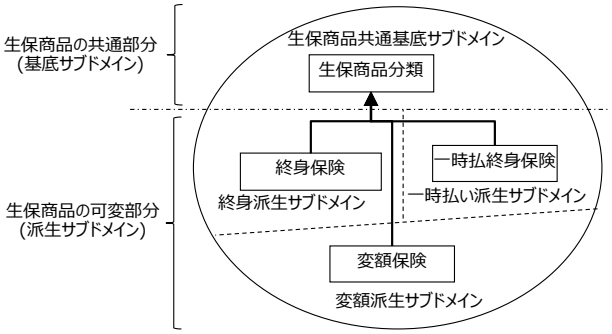

図 1　分割サブドメインの例

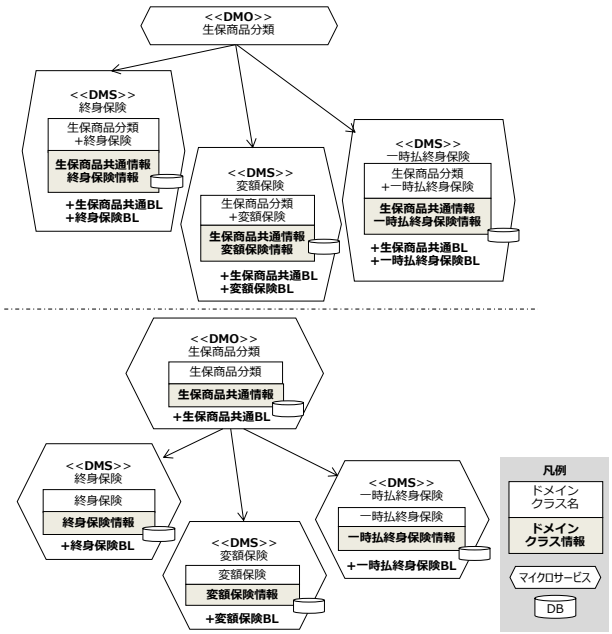

図 2　設計方針の例 (上部：方針 A，下部：方針 B)

行時にスケールして実行時の負荷分散に対応できることである．逆にデメリットは，派生サブドメイン間で共通部分が大きくなる場合である．この場合，派生サブドメインに共通した部分を，別のサブドメインとしてマイクロサービス分割することを検討する必要がある．また，マイクロサービス分割によってマイクロサービスの数が増えると，マイクロサービス間の通信が増加しレイテンシが低下する懸念がある．

2.5　例

ここでは業務アプリケーションとして生命保険契約を管理するシステムのマイクロサービス分割を例に，図 1 に分割サブドメインの例と図 2 の上に設計方針 A と図 2 の下に設計方針 B の例を示す．

従来の生命保険契約管理システムは新しい商品が追加されていくごとにシステムが複雑化し保守容易性が低下していくことが問題であった．そこで商品サブドメインを中心としたマイクロサービス分割し，継続的に商品追加を伴う生保保険契約管理システムの保守容易性を担保できるように検討する．

生命保険の中には，終身保険，変額保険や一時払い終身保険，など様々な商品の種類がある．様々にあるが契約情報の観点から共通する部分も存在する．その一方で，新しい種類の生命保険の追加といった業務要求追加が想定されるため生命保険の種類ごとに派生サブドメインを設定して DMS を実装し，DMS を統制する DMO として保険商品分類を実装する．

3　派生マイクロサービスオーケストレータ

DMS を統制する DMO について説明する．

3.1　DMS へのファサード

DMO で DMS の API を呼び出し側が意識せずに済むように，DMO の設計に Façade パターン [4] を適用し DMS の API を DMO 側で統制し隠蔽できるようにする．

この際に，呼び出しのパラメータから適切な DMS を特定できる仕組みが必要である．例えば，終身保険の契約情報をパラメータとして DMO が受け付けた場合に，リクエストデータから保険種類を特定し終身保険に該当する DMS の API を呼び出す．

3.2　DMS の API の統制

業務要求によって DMS の増減が想定され，その場合に必要となる DMO の改修は，できる限り最小限に留めたい．そこで DMO と DMS に Strategy パターン [4] を適用する．具体的には各 DMS が同じ API を持ち，DMO は呼び出す DMS を切り替える，または順次に API を呼び出すなどして統制し，結果を統合して DMO 呼び出し元へ返す．

例えば，ある条件で全保険種類の契約情報を取得したい場合に，各 DMS の契約情報検索 API を順次呼び出していく．なお，この例では DMS が多数になると DMS の呼び出しが多数になるため性能的な問題が懸念される．この場合，DMO 側に DMS が持つ DB の一部を移す対応が考えられる．

3.3　DMS 間のインタラクションの統制

DMS の処理の間に依存関係がある場合，DMO で DMS に跨る処理を実装し統制する．各 DMS に分散している DB 間のデータ整合性を維持するために，DMO で各 DMS の DB に対する分散トランザクション [5] で統制する必要がある．

例えば，変額保険の契約から終身保険の契約へ切り替える場合には，変額保険 DMS で契約解約処理した後に終身保険 DMS で新規登録する．終身保険 DMS 側で何らかの原因で処理が中断した場合，変額保険 DMS で契約解約状態になった DB をロールバックして元に戻す．このような DMS に跨るトランザクションの統制も DMO で行う．

4　おわりに

今後は，提案方法に基づくマイクロサービスの開発を推進し，実績を積み重ねていきたい．

参考文献

[1] Mehmet Söylemez, et al, “Challenges and Solution Directions of Microservice Architectures: A Systematic Literature Review,” Applied Sciences 12, no. 11: 5507 (2022).

[2] Chris Richardson, “Microservices patterns: with examples in Java,” Simon and Schuster (2018).

[3] Martin Fowler, “Patterns of Enterprise Application Architecture,” Addison-Wesley (2002).

[4] Erich Gamma, et al, “Design patterns: Elements of reusable object-oriented software,” Addison-Wesley (1995).

[5] 野田昌太郎 ほか,“マイクロサービスアーキテクチャ向け分散トランザクション管理技術の提案と評価,” 第 20 回情報科学技術フォーラム (FIT2021) 講演論文集 第 2 分冊, pp.169-170, (2021).