Vol.2018-DPS-174 No.34
Vol.2018-CSEC-80 No.34
2018/3/6

モノリシックサービスからマイクロサービスへの構造変更を容易化するためのウェブアプリケーションフレームワークの検討

東野 正幸[1,a)]

概要：ウェブサービス開発の初期段階においてはモノリシックなフレームワークによる開発が効率的である一方で，ウェブサービスが成長するにつれてマイクロサービスアーキテクチャの優位性が高まるものの，そのアーキテクチャの移行は複雑で難しい作業となる．この難しさの要因は，これまでに利用されてきた多層アーキテクチャを前提をしたモノリシックなフレームワークが持つ技術的制約に一因があると考えられる．本稿では，モノリシックサービスからマイクロサービスへの構造変更を容易化することを目的とした既存の技術的制約を持たない原子的で一貫性のあるウェブサービスフレームワークを実現するための課題について考察する．

キーワード：ウェブサービス，モノリシックサービス，マイクロサービス，フレームワーク，モバイルエージェント

Masayuki Higashino[1,a)]

1. はじめに

情報通信技術の発展と共に情報システムは複雑化してきた．実空間の人とモノが，情報ネットワークを介して相互に繋がり，自律的かつ実時間でお互いに影響を及ぼしあえる社会となった．また，プログラミング教育の普及やさらなる情報通信技術の発展により，人々は今後ますます有機的に情報システムを複雑化させてゆくと考えられる．

複雑化する情報システムを人間が理解可能とする手段として，人々はこれまでソフトウェア工学を駆使し，その複雑性に対応してきた．複雑性に対応するための基礎的な考え方として，問題の抽象化と分割が挙げられる．近年では，マイクロサービスアーキテクチャと呼ばれ，情報システムをサービスと呼ばれる小さな独立性の高いサブシステムに分割し，それらを相互に連携させて動作させる技術が，ウェブシステムやクラウドサービスの分野で普及している．また，人間が複雑な情報システムと容易に意思疎通を可能にするための自律性，協調性，反応性といった生物的な特性を与えるエージェント技術も浸透している．

多数のあらゆるモノ（物，人，生物，データ，プロセス等）が情報ネットワークを介して影響を及ぼしあい，そして多数の情報システムが分割・統合・複製・生成・消滅し，時々刻々と構造が変化する状況において，これらを管理することは難しい問題である．

このような問題に対処するために，これまで多種多様なモノを透過的かつ効率的に扱うための基盤や自己適応により変化に対応する機構が，ウェブシステム，並列分散処理技術，クラウド技術，仮想化技術，及びソフトウェア工学の分野で研究され提案されている．ウェブサービスにどの程度の需要が発生するかを予測することは難しい問題である．高い需要が発生している場合，メンテナンスに伴うウェブサービス全体の停止は，機会損失につながる可能性があるため，可能な限り停止する機能を小さく局所化することが有効である．また，大規模化したサービスの統合テストには高いコストを要するため，ビジネスに基づいたドメイン単位でテストするための開発範囲の局所化が有効である．このような背景からマイクロサービスと呼ばれるアーキテクチャが普及しつつある [1], [2], [3], [4]．マイクロサービ

[1] 鳥取大学 総合メディア基盤センター
Center for Information Infrastructure & Multimedia, Tottori University, 4-101, Koyama-Minami, Tottori, Tottori 680-8550, Japan
[a)] higashino@tottori-u.ac.jp

スとは，単一のアプリケーションを複数の小さなサービスの組み合わせにより開発する手法であり，大規模なソフトウェアをビジネスに基づいたドメイン単位で分割することで開発範囲を局所化しようとする意図を含んでいる．

しかし，ウェブサービスを開発する場合，初めからマイクロサービスに基づくアーキテクチャとして構築することは，開発にかかる初期コストを大きくする可能性がある．一般的に，マイクロサービスのような動的で疎結合かつ高い凝縮度を持つ小さなソフトウェア部品でシステムを構築するには高度な技術を要する．リリースして間もない小中規模のウェブサービスの開発効率においてはモノリシックなフレームワークを用いて開発する方が効率的であると考えられる．

モノリシックなフレームワークは三層アーキテクチャなどの比較的数の少ない層にシステムが分割されている場合が多いが，既存の層をさらに多数の層や横方向にモジュール分割することはあまり考慮されていない．さらにモノリシックなフレームワークで開発されたウェブサービスをマイクロサービスへ構造変化させるには，ビジネスに基づくドメインの設計だけでなく，既存技術を組み合わせた際に発生する様々な技術的な制約条件を満たす必要がある．つまり，ウェブサービスの初期段階においてはモノリシックなフレームワークで開発した方が効率的である一方で，ウェブサービスが成長するにつれてマイクロサービスアーキテクチャの優位性が高まるものの，そのアーキテクチャの変更はより複雑になり非常に高度な技術を要するという問題がある．この難しさの要因は，これまでに我々が作り上げてきた，HTTP に基づくリクエスト/レスポンス型プロトコルや三層アーキテクチャを前提をしたモノリシックなフレームワークがもたらす技術的制約に一因があるのではないかと考えられる．

本稿では，HTTP のリクエスト/レスポンス型といった通信プロトコルの方式や三層アーキテクチャといった多くのウェブアプリケーションフレームワークが前提としている技術的な制約を持たないウェブサービスフレームワークの実現方法を考察する．このフレームワークにより，ウェブサービスの任意箇所での分割・統合・複製・生成・消滅の自由を保障し，それらの操作による性能的影響を観測的に評価可能することで，ウェブサービスの開発者はドメイン駆動開発におけるドメイン設計と性能要件を満たすための一貫性のある定量的・定性的な議論に集中可能となる．

2. 関連研究

プロセスの自己組織化や自動管理に関する研究は活発に行われている [5], [6], [7], [8], [9]．しかし，これらの手法の多くは既存のプロセスを効果的に連携させるための手法であり，モノリシックサービスからマイクロサービスへ容易に構造を変化させるための手段を提供するものではない．

クラウドコンピューティングの分野では QoS 制御のために仮想マシンが動作する物理的なホストの場所を動的に変更する研究が進められている [10]．仮想マシンのライブマイグレーション [11], [12], [13] は，仮想マシン内の複数のリソースが複数のホストマシンにまたがって同時にフラグメント化され，なおかつ同時にアクティブな状態で実行される．この技術はモノリシックサービスを複数のマイクロサービスへ透過的に分割するための技術として有用であると考えられる．しかし，これらの仮想マシンのライブマイグレーション技術では，ライブマイグレーションにより 1 つの仮想マシンが 2 つ以上のインスタンスに分割され，その後独立して動作することは想定されておらず，モノリシックサービスをマイクロサービスへ分割する場合とは条件が異なる．

分散データベースの分野では，Azure Cosmos DB[14] などの一貫性レベルとデータベースのモデルを組みわせて利用可能なプロダクトが普及している．しかし，多層アーキテクチャを前提としているためビジネスロジックが複数の層を横断している場合にはビジネスモデルの分割や統合の容易性には寄与しない．

3. アイデア

マイクロサービスの開発においてはビジネスロジック単位で個々のマイクロサービスを設計する場合が多い．既存のモノリシックサービスをマイクロサービスに分割する場合，各マイクロービスが担当するビジネスロジックが疎結合性と高凝縮性の両方を達成できる領域を探索する方法が有効である [1]．しかし，モノリシックサービスをマイクロサービスへ分割する場合には，この領域が技術的境界を越える可能性がある．例えば，三層アーキテクチャに基づいたフレームワークにおいては，特定のデータとその処理に関するサービスを独立させてマイクロサービスとして分離したい場合にこれらの層を横断する可能性がある．

このため，既存のモノリシックなフレームワークをそのままマイクロサービスのアーキテクチャに転用することは技術的境界により難しい．そこで，データの保持とデータの処理の両方が可能な独立した小さなソフトウェア部品であるモバイルエージェントをマイクロサービスのフレームワークとして採用することでこの問題を解決できると考えられる．

本アイデアの実現には，モバイルエージェントの物理的な場所やモバイルエージェント間の通信の高度な仮想化が必要になる．モバイルエージェント群が正しくシステムのドメイン群を表現していれば，ドメインを分割したり，そのドメインを持つエージェントを異なるホストへ移動させても，システムとして動作する．これには同じホストで通信しているエージェントと異なるホストで通信しているエージェントがドメインモデルの上では同一のものとして

開発できるようなフレームワークが必要である．

ただし，システムが動作する計算機群の物理的構造を考慮せずドメインモデルにのみ基づいてシステムを分割すれば性能が低下する場合がある．一般的には，計算機群の物理的なトポロジや通信プロトコルといった技術的制約を考慮してシステムを分割するが，本提案で重要な点はこれらの技術制約を排して分割容易性を優先することである．言い換えれば性能的問題を先送りする．仮に性能が低下した原因を把握できるのであれば，特に問題の大きい性能的問題を避けるように別角度で分割するアプローチをとる．これにはモバイルエージェントのデバッガ及びプロファイラが必要になる．これにより，システムの構造変化に対する容易性が結果的に性能的問題の解決も容易にする可能性がある．

さらに，モバイルエージェントは柔軟に計算機間を移動可能であるため通信プロトコルとしても機能する．これは，HTTP のリクエスト/レスポンス型プロトコルや WebSocket の双方向型プロトコルなど様々な通信プロトコルに対応可能であることを意味する．

4. 課題

4.1 原始的アーキテクチャの設計

多くの情報システムは多層アーキテクチャを採用しており，層ごとにアーキテクチャが分かれている．このため，プログラムの実行方式やデータの処理方式が層ごとに異なる．このような多層アーキテクチャは層単位の技術の交換容易性を高めるものの，その反面で各々の層に点在するプログラムとデータを横断的に統合したり分割したりすることは難しく，マイクロサービスのような細粒度の情報システムの分割や統合の実施では層の境界が技術的境界となり制約になる．このため，情報システムに対して，任意の分割・統合・複製・生成・消滅を可能にする自由度を与えるには，情報システム全体で統一されたプログラムの実行方式とデータの処理方式を定義する必要がある．これは，エージェント間のメッセージ交換とエージェントの計算機間の移動によって分散システムを実現するモバイルエージェント技術（※ MASIF，FIPA などによって標準仕様が提案されている．）を応用できると考えられる．これにより自由に分割・統合・複製・生成・消滅を実施可能な情報システムの原始的アーキテクチャを明らかにする．

4.2 トラストの担保機構の設計

情報システムにおけるトラストには主に 2 種類ある．1 つはプログラムのトラストである．これには，情報システムを構成するプログラムが信頼可能かどうかを監査・統制する仕組みが，Grafeas と呼ばれる API などによって提供され始めており，この知見を利用できる．もう 1 つは，データのトラストである．多数のサービスで構成される情報システムにおいて，あるデータが，どこで生産され，どのサービスにおいて入力され，経由し，処理され，出力され，どこで消費されたか，といったデータの生産から消費までのトレーサビリティを担保することで検証可能になると考えられる．そこで，サービスの分割・統合・複製・生成・消滅を実施した場合に，それぞれのサービス間の家系図を表すグラフを管理し，データ処理（入力，加工，出力）の履歴と共に記録することで，データのトレーサビリティを確保する．これにより，あるデータどのように処理されてきたのかを追跡可能になり，データのトラストが検証可能になる．

4.3 定量的分析機構の設計

情報システムを任意のサービスとして任意の機能で分割・統合・複製・生成・消滅可能にし，かつサービス間でのデータのトレーサビリティを担保することで，エンジニアは既存技術による仕様的制約や性能的制約の保留しながら現実の問題領域のドメインに基づいて情報システムの構造変更が可能になると考えられる．しかし，理想的なドメインのモデルであっても，そのままでは性能的に実用できない場合がある．このため，エンジニアが理想的なドメインのモデルを使用し，かつシステムがデータのトレーサビリティを担保した上で，ボトルネックになっているサービスの性能改善や最適化を後から実施できる必要がある．そのために，各サービスでの処理負荷とサービス間のデータのトラフィックを測定・評価する機構を設計する．これにより，トラスト性と現実の問題領域を良く表現したドメインに基づいたモデルと，情報システムの性能を満たした，システムの構造をインタラクティブに見つけ出すことが可能になる．

4.4 定性的分析機構の設計

情報システムを構成する複数のサービスがどのような関係で連携しているかを分析する方法はこれまでに多数提案されている [9]．しかし，これらの手法は，サービス間の関連を表現できるが，情報システムの分割・統合・複製・生成・消滅による構造変更の差分を記録・表現することはできない．トラスト性を担保するためにプログラムとデータのトレーサビリティを保証するためには，あるサービスがどのサービスから分割されたものか，または複製されたものか，それとも新しく生成されたものかを追跡できる必要がある．オペレーティングシステムにおいてはプロセスやスレッドにおいて実現できているが，提案機構においては，ネットワークで接続された複数の計算機間を横断しても追跡できる機構が新たに必要となる．

4.5 既存サービスのラッピング機構の設計

本研究で提案する機構は，まず初めに小さな情報システ

ムがあり，システムに対する需要や提供する機能の変化に応じて，柔軟にサービスを分割・統合・複製・生成・消滅させてゆくことを主な運用シナリオとして想定している．しかし，既存の情報システムから本研究で提案する機構へ移行する手段も必要となる．そこで，既存の情報システムを構成している各サービスを，本機構でのサービスへマッピングし，徐々に本機構へ移行させる方式を検討する．そのためには，既存サービスをラッピングする方式として，デザインパターンに基づき，プロキシ方式（同じ仕様のサービスを 1 対 1 で対応させる．），アダプタ方式（異なる仕様のサービスを 1 対 1 で対応させる．），ファサード方式（サービスを n 対 n で対応させる．）を検討する．これらのデザインパターンにより，サービスを対応させ，徐々に既存システムのサービスから提案機構のサービスへ移行させる方法を検討する．

4.6 フレームワークの設計と実装

課題 4.1〜4.5 の成果を実現したフレームワークを設計及び実装する．また，提案フレームワークは，実用性を高めるために，一般的に普及している AWS や Heroku などのクラウドサービスにおいて実行可能な実装とする．特に，モノリシックアーキテクチャに基づく Ruby on Rails や CakePHP といったウェブアプリケーションフレームワークから，分散型のアーキテクチャである提案フレームワークへ移行しやすいよう，ウェブサービスとの接続容易性を高めることを検討する．

4.7 アプリケーションへの適用と評価

提案機構を実アプリケーションに適用し，本研究の有用性と有効性の評価をする．

5. おわりに

本稿では，モバイルエージェント技術を用いてモノリシックなウェブサービスをマイクロサービスへ容易に分割できるフレームワークの実現方法について考察した．本提案で重要な点は，本フレームワークが多くのウェブサービスの開発で用いられる通信プロトコルの方式や三層アーキテクチャといった技術的制約を排除することである．この性質がモノリシックなウェブサービスからマイクロサービスへの移行を容易にし，ウェブサービスのリリース期から普及期までのシステムの構造的変更に流動性を与えることが可能になると考えられる．

謝辞 本研究の一部は JSPS 科研費 15K15982 の助成を受けた．

参考文献

[1] Nadareishvili, I., Mitra, R., McLarty, M. and Amundsen, M.: *Microservice Architecture: Aligning Principles, Practices, and Culture*, O'Reilly Media, Inc., 1st. edition (2016).
[2] Awesome Microservices: Awesome Microservices: A curated list of Microservice Architecture related principles and technologies, CC0 1.0 Universal (CC0 1.0) Public Domain Dedication (online), available from ⟨https://github.com/mfornos/awesome-microservices⟩ (accessed 2018-02-08).
[3] Garriga, M.: Towards a Taxonomy of Microservices Architectures, *Proceedings of the 1st Workshop on Formal Approaches for Advanced Computing Systems, located at the 15th International Conference on Software Engineering and Formal Methods*, pp. 1–15 (2017).
[4] Richardson, C. and Smith, F.: *Microservices From Design to Deployment*, NGINX, Inc. (2016).
[5] Immonen, A. and Pakkala, D.: A survey of methods and approaches for reliable dynamic service compositions, *Service Oriented Computing and Applications*, Vol. 8, No. 2, pp. 129–158 (2014).
[6] Toffetti, G., Brunner, S., Blöchlinger, M., Dudouet, F. and Edmonds, A.: An Architecture for Self-managing Microservices, *Proceedings of the 1st International Workshop on Automated Incident Management in Cloud*, pp. 19–24 (2015).
[7] Immonen, A. and Pakkala, D.: A survey of methods and approaches for reliable dynamic service compositions, *Service Oriented Computing and Applications*, Vol. 8, No. 2, pp. 129–158 (2014).
[8] Dustdar, S. and Schreiner, W.: A Survey on Web Services Composition, *International Journal of Web and Grid Services*, Vol. 1, No. 1, pp. 1–30 (2005).
[9] Garriga, M.: Towards a Taxonomy of Microservices Architectures, *Proceedings of the 1st Workshop on Formal Approaches for Advanced Computing Systems, located at the 15th International Conference on Software Engineering and Formal Methods*, pp. 1–15 (2017).
[10] Melo, M., Maciel, P., Araujo, J., Matos, R. and Arajo, C.: Availability Study on Cloud Computing Environments: Live Migration as a Rejuvenation Mechanism, *Proceedings of the 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pp. 1–6 (online), DOI: 10.1109/DSN.2013.6575322 (2013).
[11] Wei, B., Lin, C. and Kong, X.: Dependability Modeling and Analysis for the Virtual Data Center of Cloud Computing, *2011 IEEE International Conference on High Performance Computing and Communications*, pp. 784–789 (2011).
[12] Anandkumar, A., Bisdikian, C. and Agrawal, D.: Tracking in a Spaghetti Bowl: Monitoring Transactions Using Footprints, *ACM Sigmetrics Performance Evaluation Review*, Vol. 36, No. 1, pp. 133–144 (2008).
[13] Melo, M., Maciel, P., Araujo, J., Matos, R. and Arajo, C.: Availability Study on Cloud Computing Environments: Live Migration as a Rejuvenation Mechanism, *Proceedings of the 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pp. 1–6 (2013).
[14] Microsoft Corporation: Azure Cosmos DB, Microsoft Corporation (online), available from ⟨https://azure.microsoft.com/en-us/services/cosmos-db/⟩ (accessed 2018-02-08).