6A-05

既存資産を活用したシングルページアプリケーション(SPA)開発方式の考察*

岡田　太†　　森田　孝哉†　　佐藤　裕介†

株式会社日立製作所　プロジェクトマネジメント統括推進本部‡

1. はじめに

近年、エンタープライズアプリケーションの分野でもシングルページアプリケーション[1](SPA)やマイクロサービスアーキテクチャ[2](MSA)といった比較的新しい Web アプリケーション開発方式の適用検討が進んでいる。一方、Servlet/JSP といったアーキテクチャで開発したソフトウェアの既存資産が多く存在している。

エンタープライズアプリケーションの再構築にあたり、新しい開発方式を採用しながら既存の大規模システム資産を活用することは、検討すべき課題の一つである。

本稿では、MSA を前提とした SPA における既存資産を活用した効果的な開発方式について、開発事例を通して考察する。

2. SPA と MSA の特徴と効果

MSA を前提とした SPA によるアーキテクチャ(図 1)は、JSON 形式の REST API を使用した連携により SPA と MSA の親和性が高い。

これにより、既存資産のアーキテクチャに比べて、次のような効果が期待できる。

- SPA が持つモバイルクライアントへの順応性
- SPA が持つビジネスロジックの明確な分離によるサービスの再利用性の向上
- MSA が持つ疎結合なサービスの設計によるレジリエンスおよびスケーラビリティの向上
- MSA が持つリリースまでの期間短縮の効果

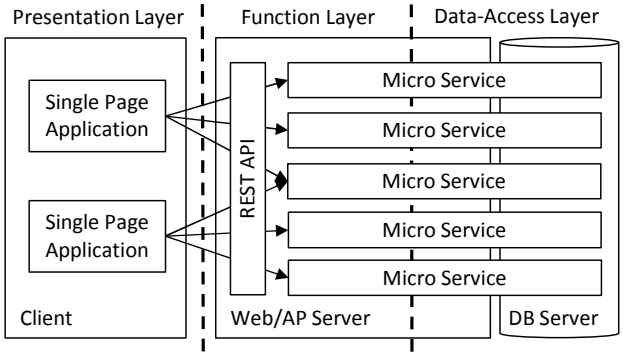

図 1：MSA を前提とした SPA によるアーキテクチャ

*A consideration of single-page application architecture with legacy application software

†Futoshi Okada, Takaya Morita, Yusuke Sato

‡Project Management Division. Systems & Services Business Management Division. Information & Communication Technology Business Division. Hitachi, Ltd.

3. 既存資産のアーキテクチャとの違い

既存資産の多くは、Servlet/JSP による MVC に基づくサーバサイドベースのモノリシックなアーキテクチャ(図 2)である。

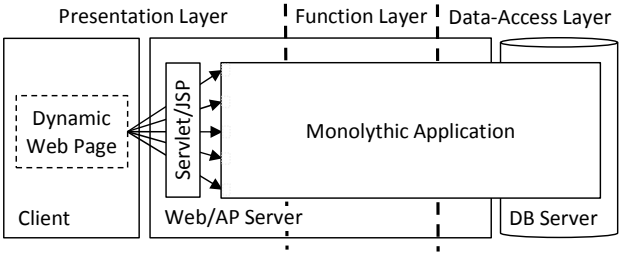

図 2：既存資産のアーキテクチャ

図 2のアーキテクチャが全ての処理をサーバサイドの Java で実装するのに対し、SPA は画面制御に関する処理をクライアント側で実装することになる。また、MSA ではアプリケーション全体に対してサービス単位に分離が求められるため[2]、全面的な作り直しが必要となる。従って、既存資産の活用を検討することが重要となる。

4. 既存資産の活用

既存資産の活用に関する研究には、異なるプログラミング言語間の再構築における効率化[3]や、同等なアーキテクチャでの移行[4]に関するものがある。前者は画面制御に関する処理の再利用に着目内容になっており、アプリケーション全体の再利用には触れていない。後者は設計変更の影響が少なく、再利用性が高い。本稿では、図 2のアーキテクチャで構築された既存資産に対し、全面的な作り直しを伴う SPA と MSA の適用に向け、再構築の効率化を目的にアプリケーション全体の再利用性に着目して漸進的な対応を検討する。

SPA の適用は基本的にビジネスロジックの見直しが不要で、リライト[5]での対応が見込める一方、MSA の適用はビジネスロジックの見直しが必要となる。MSA については、画面制御に関する処理の分割に留め、将来的に MSA の全面的な適用を想定した課題を抽出する。

5. SPA の適用方針

既存資産の流用による生産性向上を目的として、以下 2 点を前提条件にリライトする方針とした。

① 画面遷移先の決定処理をサーバ側に残す。
② クライアントとサーバ間のインターフェースにおけるデータ項目を変更しない。

開発事例では、既存資産は MVC フレームワークとして Apache Struts 1.2 を使用しており、SPA の適用には MVC フレームワークとして AngularJS 1.4 を使用した(図 3)。

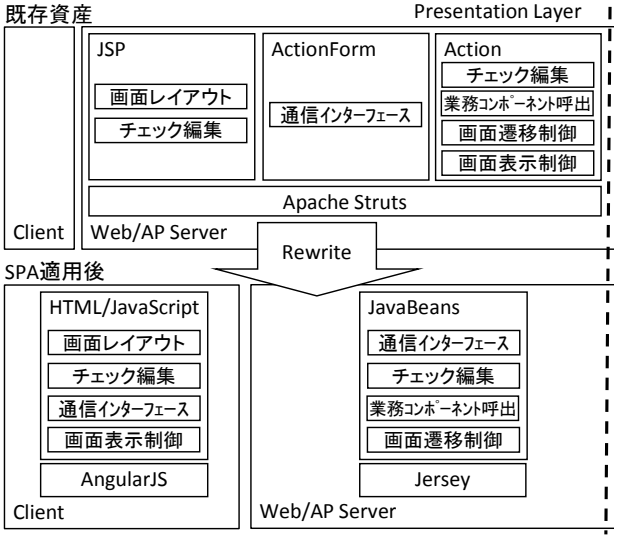

図 3：SPA 適用におけるリライトのイメージ

6. 評価

6.1 設計作業の効率化

リライトを前提とした適用方針により、作り直す場合に比べ、画面制御に関する処理の設計項目における再設計の対象を削減できた(表 1)。

表 1：画面制御に関する処理の再設計要否

#	設計工程	設計項目	再設計要否
1	画面基本設計	画面遷移設計	不要
2		画面レイアウト設計	不要
3	画面詳細設計	画面項目設計(通信インターフェース設計)	不要
4	画面イベント	イベント処理クラス設計	不要
5	設計	チェック編集処理設計	不要
6	画面イベント	画面遷移制御ロジック設計	要(一部)
7	詳細設計	画面表示制御ロジック設計	要
8		業務コンポーネント呼出ロジック設計	不要

6.2 開発規模(SLOC)

既存資産と SPA 適用後におけるリライト対象の開発規模を比較した(表 2)。#1 は小さくなったが、#5 と#7 は大きくなった。HTML の部品化により規模が圧縮された反面、通信インターフェースの制御処理で規模が膨む結果となった。

表 2：プログラム開発規模の比較(単位:Kstep)

#	処理要素	既存資産		SPA適用後	
		種別	開発規模	種別	開発規模
1	画面レイアウト	JSP	6.8	HTML	2.7
2	チェック編集(クライアント)	JavaScript	2.8	JavaScript	1.4
3	通信インターフェース(クライアント)	ー	ー	JavaScript	1.1
4	画面表示制御	Java	3.8	JavaScript	3.7
5	通信インターフェース(サーバ)	Java	2.0	Java	14.9
6	チェック編集(サーバ)	Java	1.5	Java	1.5
7	画面遷移制御	Java	7.5	Java	13.9
8	計		24.3		39.1

6.3 技術的な実現性の評価

リライトによる技術的な実現性を、リライト作業中に挙がった問題点(表 3)から評価した。結果的に未対策となったのは 3 件だけであり、実現性には概ね問題無いと評価する。未対策の課題として、画面全体の再描画におけるタイミングの差異や、HTML の部品化に伴う CSS の名称競合の他、サブウィンドウ制御に関する処理方式等について、再検討が必要なことが判明した。

また、サービスを跨ぐ箇所での認証情報の引き渡しや二重送信防止用の作り込みに関する、MSA の全面的な適用に向けた課題が判明した。

表 3：リライト作業中の問題点

#	処理要素	対策済	暫定対策済	未対策	計
1	画面レイアウト	7	5	0	12
2	チェック編集(クライアント)	0	0	0	0
3	通信インターフェース(クライアント)	3	0	0	3
4	画面表示制御	5	1	1	7
6	チェック編集(サーバ)	0	0	0	0
7	通信インターフェース(サーバ)	1	0	0	1
5	画面遷移制御	9	0	1	10
8	その他	4	0	1	5
9	計	29	0	3	38

7. まとめ

サーバサイドベースのモノリシックなアーキテクチャによる既存資産を活用した SPA 開発方式の研究の結果、画面制御に関する処理における再設計の対象項目の削減により、開発工数の削減に効果が得られた。SPA 開発方式の妥当性を評価できたことに加えて、残った問題点における技術的な課題を洗い出した。

8. 今後の課題

本稿で抽出した課題に加えて、既存資産の流用を目的に設定した前提条件に対して、SPA としてのメリットを活かす観点で改めて検討する。

また、MSA の全面的な適用に向けて、リバースエンジニアリングを活用したアプリケーション全体の最適化の手法を検討する。

参考文献

[1] 池添 明宏, 金井 健一, 吉田 徹正. AngularJS リファレンス. インプレス, 2014, 166-168p.
[2] Sam Newman 著, 木下哲也 訳. マイクロサービスアーキテクチャ. オーム社, 2016, 1p-9p, 93-119p.
[3] 米谷 雅樹, 『Visual Basic から J2EE 環境への移行支援ツール』, 情報処理学会第 66 回全国大会.
[4] 倉持 和彦、他『Struts アプリケーションのパブリッククラウド(Google App Engine)への移行に関する考察』, 情報処理学会第 72 回全国大会.
[5] IT モダナイゼーションソリューション, "IT モダナイゼーションとは". 日立製作所. 2016, (URL:http://www.hitachi.co.jp/Prod/comp/soft1/solution/search_s/modernization/whats_modernization.html)