早稲田大学における既存財務システムへのデータ入力機能の RPA からプロコードへの再構築事例

櫛渕 倫太郎 [1)]　　佐藤 亙人 [1)]　　神馬 豊彦 [1),2)]

1) 株式会社早稲田大学アカデミックソリューション
2) 香川大学大学院 創発科学研究科

r.kushibuchi@w-as.jp

Rebuilding the Data Input Function for an Existing Financial System from RPA to Pro-Code: A Case Study at Waseda University

Rintaro Kushibuchi[1)]　　Arito Sato[1)]　　Toyohiko Jimma[1),2)]

1) Waseda University Academic Solutions Corporation
2) Graduate School of Science for Creative Emergence, Kagawa University

概要

早稲田大学では、電子帳簿保存法への対応をきっかけに、2023 年度より Microsoft Power Platform 上に「経理アプリ」を構築し運用を開始した。現在、経理アプリから既存の財務システムへのデータ入力は、RPA（Robotics Process Automation）により自動化している。RPA は、短期間で既存財務システムへのデータ入力機能を実現した一方で、処理の遅延、ライセンス費用の増大、運用保守の複雑化といった課題も顕在化した。これらの課題を解決するため、プロコードによる既存財務システムへのデータ入力機能の再構築を検討・実践した。本稿では、既存財務システムへのデータ入力機能の RPA からプロコードへの再構築の具体的なロードマップと、その過程で得られた知見を報告する。

1　はじめに

情報処理推進機構（以下、IPA）の「DX 白書 2024」[1] では、DX 実現に必要な考え方やシステム開発手法、運用方法論としてデザイン思考、アジャイル開発、DevOps [2] をあげるとともに、システム開発の生産性を向上すべくローコード・ノーコードプラットフォームを活用した内製開発についてもその効果を言及した。ローコード・ノーコードプラットフォームのひとつとして RPA（Robotics Process Automation）が注目され、多くの大学で導入が進められている。プログラミングの専門知識を必要とせず、GUI 操作の自動化によって迅速な業務改善を実現する RPA は、多くの現場で歓迎された。

しかし、短期的な成果をもたらす一方で、大規模な業務における RPA の利用は新たな課題を生み出す。画面操作に依存することによる処理速度の限界、実行環境の増加に伴うライセンスコストの増大、そして UI 変更に弱いという脆弱性は、やがて無視できない「技術的負債」として組織に重くのしかかる。早稲田大学においても、既存財務システムへのデータ入力に RPA を導入し、その恩恵を享受する一方で、まさにこの問題に直面することとなった。

早稲田大学では、RPA による既存財務システムへのデータ入力を根本から見直し、財務システムのデータベースの参照・更新ログおよび RPA のエラーログの解析による要件の正確な把握と、プロコードによる再構築を実践した。本稿では、RPA による大規模な既存システムへのデータ入力機能のプロコードによる再構築事例について述べる。2 章では現状と課題について、3 章ではプロコード化へのアプローチについて、4 章では考察を、5 章ではまとめを述べる。

2　現状と課題

早稲田大学では、電子帳簿保存法への対応として、電子で受領した経理証憑を電子のまま提出・保管するために、「経理アプリ」を開発した。経理アプリは、それまで紙で提出されていた伝票を電子化し、電子的に

受領した証憑類や電子化した証憑類を添付して提出できる仕組みとして開発された。経理アプリはローコード・ノーコードプラットフォームの Microsoft Power Platform [3] により開発され、各部署が経理アプリに入力した伝票情報の、既存財務システムへのトランザクションデータの入力が必要であった。開発当初、連携先システムの仕様が複雑であること、法令対応に向け短期間での実装が求められていたことから、RPA を既存財務システムへのデータ入力手段として採用した。

図 1 は、経理アプリから既存財務システムへのデータ入力処理の現状について示す。経理アプリは Microsoft Power Apps、Microsoft Power Automate、Microsoft Dataverse で構築され、入力された伝票データ、証憑ファイルは Dataverse に保存される。RPA は、CData Software 社が提供するデータ接続コンポーネントの CData ODBC ドライバ [4] を利用して経理アプリ内の伝票データ等のトランザクションデータを取得し、Microsoft Azure 基盤上に構築された財務システムに対して、伝票データを入力する。

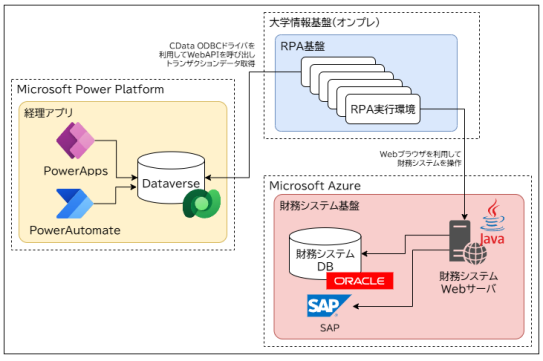

図 1　RPA によるデータ連携の現状

RPA の採用は、迅速な導入と柔軟な連携というメリットをもたらした。既存財務システムの内部ロジックやデータベース構造を深く理解せずとも画面操作を模倣することで開発期間を大幅に短縮でき、ローコードプラットフォームと Azure 上に構築された既存財務システムという異なる情報基盤で構成されたシステム同士でも比較的容易にデータ連携を実現できた。

経理アプリは、年間 25 万件程度の伝票入力に利用されているが、その利用が拡大・長期化するにつれて、無視できない以下のデメリットが顕在化した。

第一に、画面操作をエミュレートするため、API 連携に比べて本質的に処理速度が遅く、データ量の増加に伴い処理時間が延伸し、対応するためには RPA 実行環境（ロボット）の台数を増やす必要があった。この実行環境の増加は、RPA ライセンス費用の増大に直結し、コストパフォーマンスを悪化させた。第二に、運用・保守が複雑である点である。連携先システムの UI 変更に非常に脆弱で、些細なレイアウト変更でも RPA が停止するリスクがあった。また、問題の特定にはログ情報だけでは不十分な場合が多く、処理を再現しながら画面を目視で確認する必要があった。図 2 に示す通り、年間約 25 万件の処理に対し約 2000 件のエラーが発生し、その対応に約 2500 時間を要するなど、運用工数の増大も深刻であった。これらの課題から、より根本的で持続可能な連携方式への移行が不可欠であると判断した。

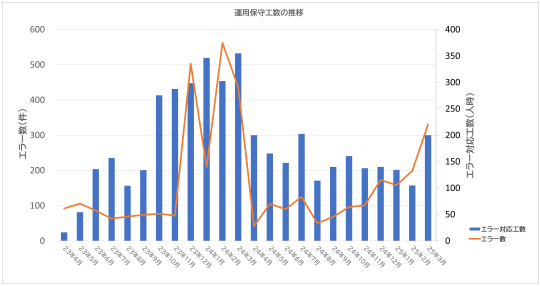

図 2　RPA の運用・保守工数の推移

3　プロコード化へのアプローチ

既存財務システムへのデータ入力機能のプロコードによる再構築にあたり、現状の業務プロセスを再評価し、より効率的で安定した連携方式へと進化させるための体系的なアプローチを計画した。

3.1　現状の可視化と分析

現状の可視化と分析により要件を明確にするため、既存財務システムのデータベースの参照・更新ログと RPA のエラーログの分析をおこなった。加えてプロコード化前後の運用工数を定量的に予測し。投資対効果の明確化をおこなった。

RPA の処理定義を追うだけでは、既存財務システムのデータ入出力要件を把握することは困難であった。また、関連する仕様書も形骸化している可能性があった。そこで、RPA がアクセスする既存財務システムのデータベースに着目し、ロボットの動作中の参照・更新ログ（SQL ログ）を処理別に収集・分析した。これにより、RPA が実際にどのテーブルのどのデータを、どのような条件で参照・更新しているかを客観的に、かつ網羅的に把握することができた。このアプローチは、仕様書の不備を補い、正確なデータ要件を定義する上で極めて有効であった。

プロコード化における重要なもう一つの課題は、エラーチェック要件の策定であった。既存財務システムの仕様書に記載された仕様を網羅的に実装しようとすると、経理アプリで検証済みの項目や、実際には発生し得ないエラー処理まで実装対象となり、開発効率の低下が懸念された。そこで、RPA が過去 2 年間に財務システムの画面から受け取った実際のエラーログを分析するアプローチを採用した。この分析に基づき、発生実績のあるエラーのみを必須要件とし、それ以外は実装の要否を個別に判断することで、現実的かつ効率的なエラーハンドリング仕様を策定した。

プロコードへの移行には、当然ながら開発コストが発生する。学内の合意形成を円滑に進めるためには、その投資対効果（ROI）を明確に示す必要があった。RPA の運用・保守にかかる月次の工数や、エラー発生時の対応時間、障害による業務影響などを具体的に数値化した。これにより、現状維持コスト（RPA を使い続けるコスト）と、プロコード化による将来的なコスト削減効果を定量的に比較することが可能となった。

3.2　新たなデータ入力アーキテクチャの設計

現状分析で得られた要件に基づき、新たなデータ入力のアーキテクチャを設計した。図 3 にその概要を示す。

既存財務システムは Java で実装されており、バッチアプリケーション化に際し流用可能な部分が多く存在したため、RPA に代わるデータ連携処理は、Java で開発したバッチアプリケーションとしてサーバー上で定期実行する方式を採用した。この方式により、RPA が抱えていた実行環境の制約やスケーラビリティの問題を解消する。

開発にあたり、2 つの主要な技術的課題があった。第一に、連携元である経理アプリのデータソース（Microsoft Dataverse）には、標準的な JDBC/ODBC インターフェースが提供されていない点。第二に、連携先の既存財務システムにも、データ入力のための公式な API が存在しない点である。

これらの課題を解決するため、複数の技術要素を組み合わせるアプローチを採用した。まず、Dataverse への接続には、CData JDBC ドライバ [5] を導入した。これにより、通常の JDBC ドライバの提供のない Dataverse をあたかも通常の RDBMS のように SQL で操作することが可能となる。次に、データ抽出・変換・ロード（ETL）処理の開発には、オープンソースの ETL ツールである Talend [6] を活用した。Talend を用いることで、GUI ベースで効率的にデータ連携処理を開発でき、開発の標準化とコスト抑制の両立が期待できた。既存財務システムへのデータ入力は、いったん Dataverse 上の構造のままワークテーブルに書き込んだデータを、Java で開発したバッチアプリケーションにより、SQL ログ解析で特定したテーブルへと書き込む方式とした。

3.3　実現可能性の検証

本格的な設計・開発に着手する前に、提案するプロコード化アプローチの技術的な実現可能性を検証するため、PoC（Proof of Concept: 概念実証）を実施した。具体的には、SQL ログから特定したクエリをデータベース上で直接実行してデータ取得を試みたり、連携先システムの REST API を実際に呼び出したりすることで、机上の調査だけでは判断できない接続性やレスポンス内容を検証した。さらに、これらの手法で得られたデータを用いて手動でデータ操作を行い、後続システムで求められるデータ形式に変換可能であること、および業務担当者が利用する画面上に正しく表示されることを確認した。この PoC を通じて、提案手法が技術的に実現可能であるという確証を得ることができた。

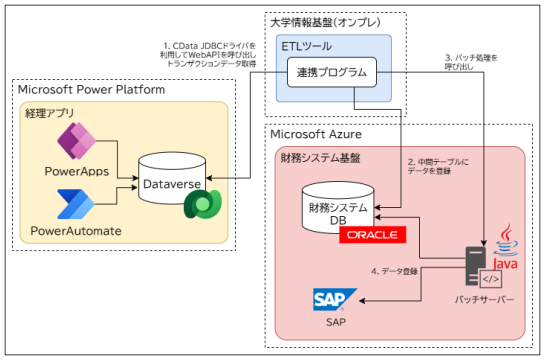

図 3　プロコードによるデータ連携

4　考察

Iden ら [7] は、ローコード・ノーコードプラットフォームの RPA を導入しているノルウェーの銀行を調査した。Iden らは「RPA の規模が大きくなると、保守の負担が大きくなるという問題に直面し、その労力は予想以上に高くなる」と述べた。特に大規模な業務での RPA の利用には、処理の遅延、ライセンス費用の増大、運用保守の複雑化といった課題がある。本稿では、大規模な RPA のプロコードへの移行のために、3 章で述べた現状の可視化と分析、新たなデータ入力アーキテクチャの設計、実現可能性の検証という 3 段階のアプローチを提案した。このアプローチを通

じて、2 章で述べた RPA の抱える課題は、以下のとおり解決した。

まず、処理速度の遅延とスケーラビリティの欠如については、現行 RPA でロボット 1 台が 30 件のデータを処理するのに 1 時間 33 分 53 秒を要したのに対し、新方式では 47.92 秒で完了した。6 台のロボットが並列稼働する現状を考慮しても、30 件あたりの実質的な処理時間（約 15 分 38 秒）を大幅に上回る速度であり、新方式の優位性が明らかになった。次に、ライセンス費用の増大という課題も解決の見込みである。本対応により、1 ライセンスあたり 100 万円以上となる、大学で契約中のロボットライセンス 23 台のうち 6 台分を削減でき、次年度以降の適用拡大により最終的には 17 台分のライセンス削減が期待される。最後に、運用・保守の複雑化に関しても改善が期待される。プロコードによるデータ入力は、RPA のように画面レイアウトやブラウザの更新に影響されず、安定した運用が可能となる。また、エラー発生時も詳細なログが出力されるため、問題の特定が RPA に比べて格段に容易になる。ただし、本プログラムの本格運用は 2025 年 12 月を予定しており、実際の運用実績に基づく評価は今後の課題である。

5　おわりに

RPA は、その手軽さゆえに無計画に利用範囲を拡大すると、処理性能の低下、コスト増、運用負荷の増大といった形で、将来的に大きな技術的負債を生む危険性をはらんでいる。

したがって、大規模な業務における RPA の利用はあくまで短期的な課題解決策、あるいはプロトタイピングの手段と明確に位置づけ、その利用と並行して、API 連携を主軸とした持続可能で堅牢なデータ連携アーキテクチャへの移行を計画的に進めるべきである。ただし、一定のサイズを下回る規模のデータ連携、例えば担当者が手元で 1 日 1 回動かせばよい、といったものであればこの限りではない。本稿で述べているのは、あくまでシステム間のデータ連携における RPA の利用についてであり、費用対効果に合わせて判断するべきであることに注意が必要である。

本稿では、早稲田大学における RPA によるデータ連携からプロコード化への移行アプローチに関する事例を報告した。SQL ログ解析、エラーログ解析、API 活用、ETL ツールと JDBC ドライバの組み合わせといった具体的な手法を通じて、RPA が抱える課題を克服し、持続可能で安定したデータ連携を実現する道筋を示した。

本稿で示したアプローチが、同様の課題を抱える多くの大学にとって、真の DX を実現するための一助となることを期待する。

謝辞

本稿の執筆にあたり、早稲田大学情報企画部情報企画課長の柴山様をはじめとする関係者の皆様には、有益なご助言やアドバイスを賜りました。また、本稿で論じた内容は、筆者らが所属する株式会社早稲田大学アカデミックソリューション IT 推進部 IT-X チームの仲間との日々の協業の賜物です。経理アプリ、並びに既存財務システムを運用・保守する中で得た貴重なデータを提供してくれただけでなく、執筆にあたり多くの有益な議論を重ねてくれたチームメンバーに、この場を借りて深く感謝いたします。

参考文献

[1] 独立行政法人情報処理推進機構, DX 白書 2024, https://www.ipa.go.jp/digital/chousa/dx-trend/dx-trend-2024.html, (2025. 09. 23. 確認).

[2] G. Kim, P. Debois, J. Willis, J. Humble, and J. Allspaw, *The DevOps Handbook: How to Create World-class Agility, Reliability, and Security in Technology Organizations*, G - Reference,Information and Interdisciplinary Subjects Series, IT Revolution Press, 2016.

[3] Microsoft, Microsoft Power Platform, https://www.microsoft.com/ja-jp/biz/dynamics/power-platform, (2025. 09. 25. 確認).

[4] CData Software Japan, Microsoft Dataverse ODBC ドライバ, https://jp.cdata.com/jp/drivers/dataverse/odbc/, (2025. 09. 26. 確認).

[5] CData Software Japan, Microsoft Dataverse JDBC ドライバ, https://jp.cdata.com/drivers/dataverse/jdbc/, (2025. 09. 26. 確認).

[6] Talend, Talend Data Integration and Data Integrity, https://www.talend.com/jp/, (2025. 09. 25. 確認).

[7] Jon Iden, Lightweight it and the it function: Experiences from robotic process automation in a norwegian bank, 11 2017.