AvailablSecoinelinnecaet Dwwirwe.sccitencedirect.com
サイエンスダイレクト
Procedia Computer Science 225 (2023) 609–616
Procedia Computer Science
www.elsevier.com/locate/procedia

第27回知識ベースと知的情報工学システムに関する国際会議(KES 2023)

データ駆動型プロセスベースのUnicageアーキテクチャ

當仲寛哲学[a] 、山本修一郎[b] 、シルネンブヤンジャルガル[a,b,] ，*

a USP研究所　〒105-0003 東京都港区西新橋3-3-3 株式会社
b 名古屋国際工科専門職大学　〒450-0002　名古屋市中村区名駅4-27-1

概要

デジタル技術を駆使して、あらゆる業界でビジネス価値や革新を生み出すことがデジタル変革には不可欠です。企業は前例のないビジネス環境に直面し、資源や市場の競争によって、より適応力を持たなければならなくなっています。

その結果、企業の情報システム(EIS)も適応性を持つ必要があります。従来の情報システムアーキテクチャでは、適応性を実現するために必要な緩く結合されたマイクロサービスアプリケーションアーキテクチャを実現するために、アプリケーションコンポーネント間の依存関係が高いです。
本論文では、特に企業資源計画(ERP)システム用に、ビジネス、アプリケーション、データ、技術アーキテクチャから成る、入出力データに基づく構築ブロックだけを使用した緩く結合された、プロセスベースの企業情報システムアーキテクチャ(ユニケージ)を提案します。
これにより、フルサイクル開発が可能になります。また、ArchiMateを用いたこのアーキテクチャのモデルも提案します。

この論文は、第27回知識基盤およびインテリジェント情報・エンジニアリングシステム国際会議の科学委員会の責任のもとで査読されました。
DOI: 10.1016/j.procs.2023.10.046

1. はじめに

最新の企業アーキテクチャ（EA）の目標は、ITシステム全体の最適化ではなく、ビジネスの変革です。
さらに、EAの目的がビジネスの変革であるため、ビジネス能力に基づいたEA開発方法では、段階的にビジネス価値を生み出す部分から実装を開始します。ビジネス価値を生み出さない活動は推奨されません。そのため、アーキテクチャの開発は継続的で循環的なプロセスであり、アーキテクチャ開発方法（ADM）を時間をかけて繰り返し実行することが必要です。
TOGAFはこの反復モデルを採用しており、これはアジャイルの重要な構成要素です。マイクロサービスアーキテクチャでは、ビジネス能力に対応するマイクロサービスがデジタル変革（DX）を促進するために緩く結合されています。これを実現するためには、TOGAFの核となるアーキテクチャ層に示されているように、全体的なビジネス、データとアプリケーション、および技術アーキテクチャを、緩く結合された方法で設計し実装する必要があります。
このタイプのアーキテクチャは、ビジネスプロセスへの入出力情報だけを考慮するため、アジャイルやTOGAF ADMのような反復的開発に対する柔軟性を可能にします。入出力データ駆動型のアーキテクチャは、各ビジネスプロセスの入出力だけを理解することが必要で、これが直接IOデータベースのアプリケーションコンポーネントにマッピングされるため、システム開発者が全サイクルのシステム開発を達成する利点もあります。
上述の事実に基づき、私たちはビジネス、アプリケーション、技術アーキテクチャ層を構成する緩く結合された構築ブロックからなるData Driven Process Based (DDPB) アーキテクチャを提案し、ArchiMateを用いたアーキテクチャの実装モデルを示します。
次に、関連研究について第2節で説明し、第3節でArchiMateによるDDPB企業アーキテクチャモデルを提案します。その後、第4節ではDDPBアーキテクチャの具体的な適用例を説明します。第5節では議論を行い、第6節で本論文をまとめ、将来の課題について述べます。

***担当者 / USP研究所　TEL & FAX 03-3432-1174**
メールアドレス　: s-buyan@usp-lab.com

第27回知識ベースと知的情報と工学システムに関する国際会議の科学委員会が担当するピアレビュー
10.1016/j.procs.2023.10.046

2.関連研究

スティーブン・H・スピークは、企業アーキテクチャ計画（EAP）として、ビジネス駆動またはデータ駆動の情報システムアーキテクチャを提案しました。彼は、アプリケーションを構築する前にデータを定義し、そのデータの依存関係がアプリケーションシステムの実装順序を決定すべきであると推奨しています。データは、手頃かつ適正なコストで提供されるべきです。データベース管理システム（DBMS）は、データにアクセスし保存するための単なるツールであり、DBMSを使用するだけではその全機能を果たすことはできません。EAPでは最初にビジネスをサポートするために必要な全データを定義するアーキテクチャを設計し、これが完成すると、次にそのデータを管理するためのアプリケーションを定義する第二のアーキテクチャに移ります。

TOGAFでは、要件管理のフェーズを通じて全プロセスにわたり反復的に行われるアーキテクチャ開発方法（ADM）を提唱しています。ADMのフェーズCは情報システムアーキテクチャに焦点を当て、スティーブン・スピークのEAPをデータ駆動型アプローチの一例として引用しています。

山本による研究では、EAフレームワークを比較するための具体的な評価枠組みが提案され、その中でTOGAFが最も優れたEAフレームワークであると結論付けられました。ただし、モデルと方法に関する特徴の部分では、TOGAFが組織特有のニーズに応じたモデルと方法の調整を行うアーキテクトの役割を明確に示していないものの、調整作業自体については説明されています。

また、TOGAFは多くの企業アーキテクチャフレームワークが特定の成果物に集中している傾向がある一方で、それらの成果物を生成する方法については詳しく言及していないと指摘しています。

山本は、ArchiMateを使用してビジネス価値、ビジネスモデル、ビジネスプロセスを視覚化する新たなアプローチを提案し、ArchiMate記法での統合モデルの構築を提案しています。さらに、山本はビジネス価値を生み出さない、またはプロセスが重複するビジネスプロセスを排除することに焦点を当てたデータ駆動プロセスデザインメソッド（DDPDM）を提唱しました。

Netflexは、「構築したものを運用する」という理念や、フルサイクルソフトウェア開発を推進しています。ここでの主張は、アイデアを顧客のための有効な製品やサービスに変換することで、「価値到達までの時間」を最適化することがソフトウェアライフサイクルの目的であるというものです。ソフトウェアサービスの開発と運用には、設計からサポートに至るまで、ライフサイクル全体に渡る一連の責任が伴い、これらが分断された場合には全体の非効率性が生じるリスクがあります。フルサイクルデベロッパーは、これらのアイデアを一つの開発チームに統合し、高い生産性を支えるツールを用いてソフトウェアライフサイクル全体の責任を負います。

CRIBB [8]、James P. Houghton [9]、João M.P. Moreira [10] は、Unicage Architecture の技術性能について言及または評価している。

3.データ駆動型プロセスベースのUnicageアーキテクチャ（UA）

UAアーキテクチャは、TOGAF（The Open Group Architecture Framework）**と一致しており、Table Iは、TOGAFにおけるコアアーキテクチャ層の基本的な構築ブロックを示しています。** これは、エンタープライズ情報システム（EIS）実装のための実用的なモデルを提案するものです。

UAアーキテクチャのアーキメイトメタモデルは、TOGAFアーキメイトメタモデルに基づいており、コア層のために構築されています。 詳細は、続く章で説明します。

Table Iに示されているように、ビジネスプロセス設計は、IOデータまたは情報に基づいて設計されているため、アプリケーションとテクノロジーの構築ブロックに直接マップされます。 これにより、マイクロサービスアーキテクチャとフルサイクル開発が可能となります。

つまり、ビジネスプロセスを分析し、設計することは、開発サイクルの効率とフルサイクル開発を可能にするデータとアプリケーション設計を時間内に準備することを意味します。　※　UA開発方法については、ここでは除外します。

表1:データ駆動型プロセスベースのUnicageアーキテクチャ(UA)コアビルディングブロック

Togaf コア要素	UAコア ビルディングブロックの定義	注記 (IO:入出力)
ビジネスアーキテクチャ	IOデータ・ベースのビジネスプロセス	ビジネス・プロセスは、入出力データを持つ関数として考えます。 プロセスは、情報またはデータのみを介して接続されます。 データ駆動型プロセス設計法[6]で分析し、設計します。
データアーキテクチャ	名前インデックス付きテキストファイル	開発手法におけるデータアーキテクチャ設計原則、 ファイルおよびディレクトリのインデックス付け規約に準拠します。
アプリケーションアーキテクチャ	IOデータ・ベースのアプリケーションコンポーネント	ビジネス・プロセスをそのままアプリケーション・コンポーネントに実装します。 コンポーネントは、ビジネスプロセスと同様に、データによってのみ接続されます。
技術アーキテクチャ	IOデータ・ベースコマンド	コマンドは、ビジネス・プロセスおよびアプリケーション・コンポーネントと同じIOデータ・ベースの構造を持つ最小の構成要素です。このコマンドには、入力データを処理するための高度な単一の機能があり、任意のプログラミング言語によって実装することができます。
アーキテクチャ開発手法	UA開発手法	UA、Full Cycle Development、プロジェクト管理方法、およびエラーハンドル、低コード、コード記述、ディレクトリツリー規則を含むプログラミング標準を含むリーンとアジャイルの原則で構成されています。

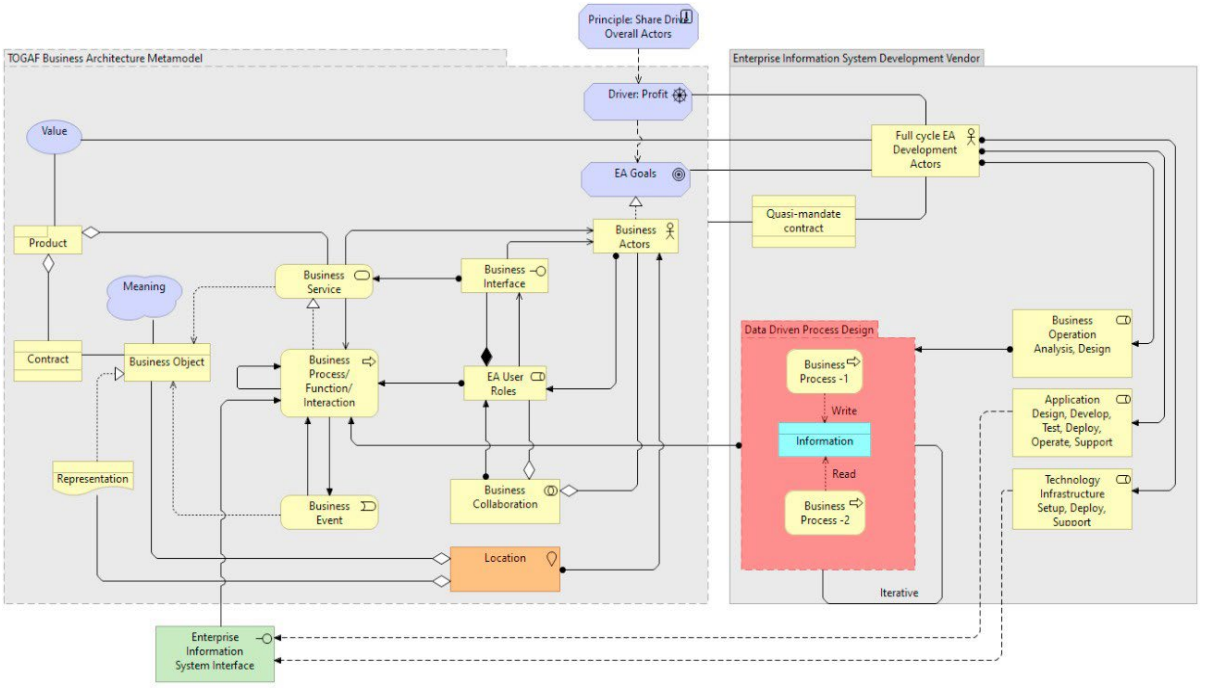

図 1.ArchiMateのビジネスアーキテクチャメタモデル

3.1. UAのビジネスアーキテクチャモデル

UAのビジネスアーキテクチャモデルでは、BAはTOGAFのBAと共に、ビジネスプロセスの実装のための構築ブロックを提案しています。図1は、特にビジネスプロセスの分析と設計の観点から、EAとEISベンダー間の関係を示しています。
UAはそのEIS開発において原則を持っています。

原則1：利益指向。
原則2：原則1に向けた全体の開発プロセスは目標指向で、目標最適化される。
原則3：エンタープライズアーキテクチャ開発チームは、原則2を実現するミッションを持つフルサイクル開発者で組織される。
原則4：すべてのビジネスプロセスを入出力プロセスとして分析し設計する。

UAは、原則4に準拠して任意のビジネスプロセスを分析し設計するためにデータ駆動プロセスデザインメソッド[6]を適用します。UAのデータアーキテクチャモデルでは、以下の原則に基づき、全てのデータはEISの動作の完全な時系列記録を残します。

これにより、問題が発生した際に、システムの動作を遡って視覚化することで原因を簡単に突き止めることができます。表2は、データタイプ、その応用、およびデータ管理能力を示しています。少数のデータタイプと、アプリケーションコンポーネントを実現するのにも使用されるデータ入出力ベースのコマンドによって実現されるデータ管理ツールを持つことで、EIS開発者の学習コストが少なくなります。データ管理のためのミドルウェアを使用せずに、開発者の学習および運用コストが少なくなり、全体的なEIS開発のコストも少なくなりますが、これはEAP[2]の提案もサポートしています。

原則1：冗長性を許容し、不要なものを排除する。
原則2：どんなデータも上書き、更新、削除されることはなく、生成される。
原則3：データ依存を避ける。

表2:UAのデータ型

データ型	アーティファクト	用途	データマネジメント
論理/レイアウトデータ	標準テキストファイル	設計、コーディング、文書化	データIOベースのソフトウェアコマンド
データ	標準テキストファイル	ビジネスアプリケーション	データIOベースのソフトウェアコマンド
メディア	バイナリ	ビジネスアプリケーション	データIOベースのソフトウェアコマンド

UAのデータアーキテクチャメタモデルは、図2に示されており、それはTOGAFメタモデルに沿ったものです。

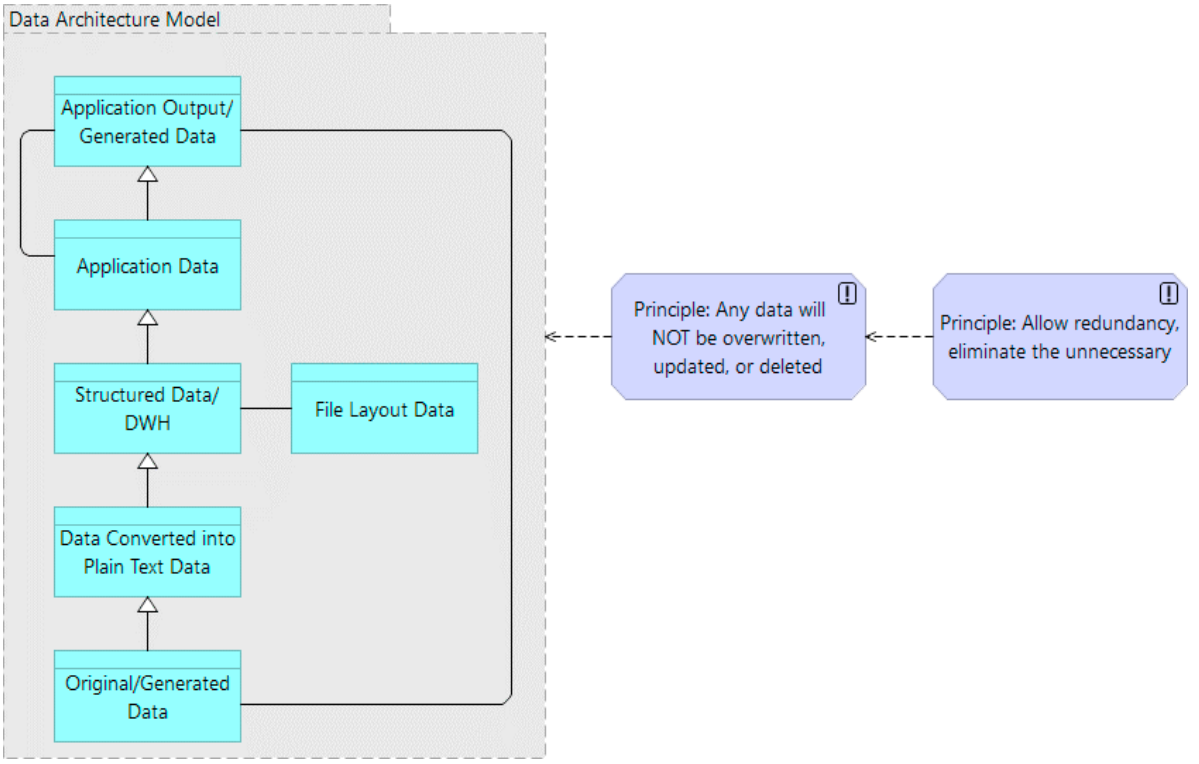

図 2.ArchiMateのデータアーキテクチャメタモデル

3.3. UAのアプリケーションアーキテクチャモデル

アプリケーションコンポーネントは、図3に示されるように、コンポーネント、ライナー、コマンドから構成されます。ライナーはUnix Pipeによって接続されたコマンドの集合であり、アプリケーションコンポーネントまたはビジネスプロセスを構成するステージを表します。あらゆるアプリケーション、ライナー、またはコマンドは、データオブジェクトのみを通じて接続されます。言い換えると、アプリケーションアーキテクチャはマイクロサービスアーキテクチャ、または疎結合です。

IOに基づくアプリケーションコンポーネントは、同じIOデータを持つ構造で提供するビジネスプロセスと一致し、これはビジネスプロセスの設計が準備されていること、そしてアプリケーションの設計も準備されていることを意味します。アプリケーションコンポーネントは、コードの理解可能性、コードの単純さ、データ管理、およびシステム保守の点で以下の原則にも準拠しています。

原則1：コマンドへの入力と出力ファイルレイアウトについての説明を書きます。
原則2：一行の入力データで単一のコマンドを書きます。
原則3：各ライナーおよびアプリケーションコンポーネント／ビジネスプロセスについての説明を書きます。
原則4：データを生成するアプリケーションは、データを使用するアプリケーションよりも先に実装するべきです。

UAによって実現されたアプリケーションコンポーネントは、IOデータに基づいたマイクロサービスアプリケーションおよびテクノロジーアーキテクチャのため、EAP[2]が提案する以下の基準を満たします。

基準1：理解可能であるべきです。アプリケーションの定義は理にかなっており、企業全体の人々によって容易に理解されます。
基準2：完全であるべきです。アプリケーションはほとんどのビジネス機能をサポートし、すべてのデータエンティティが管理されており、アプリケーション機能の重複や重複がないように一貫性があります。
基準3：安定しているべきです。各アプリケーションの定義は、ビジネスモデルおよびデータアーキテクチャにのみ基づいており、

アプリケーションを使用する人、アプリケーションの動作方法、アプリケーションの位置、またはアプリケーションの操作時間には依存しません。

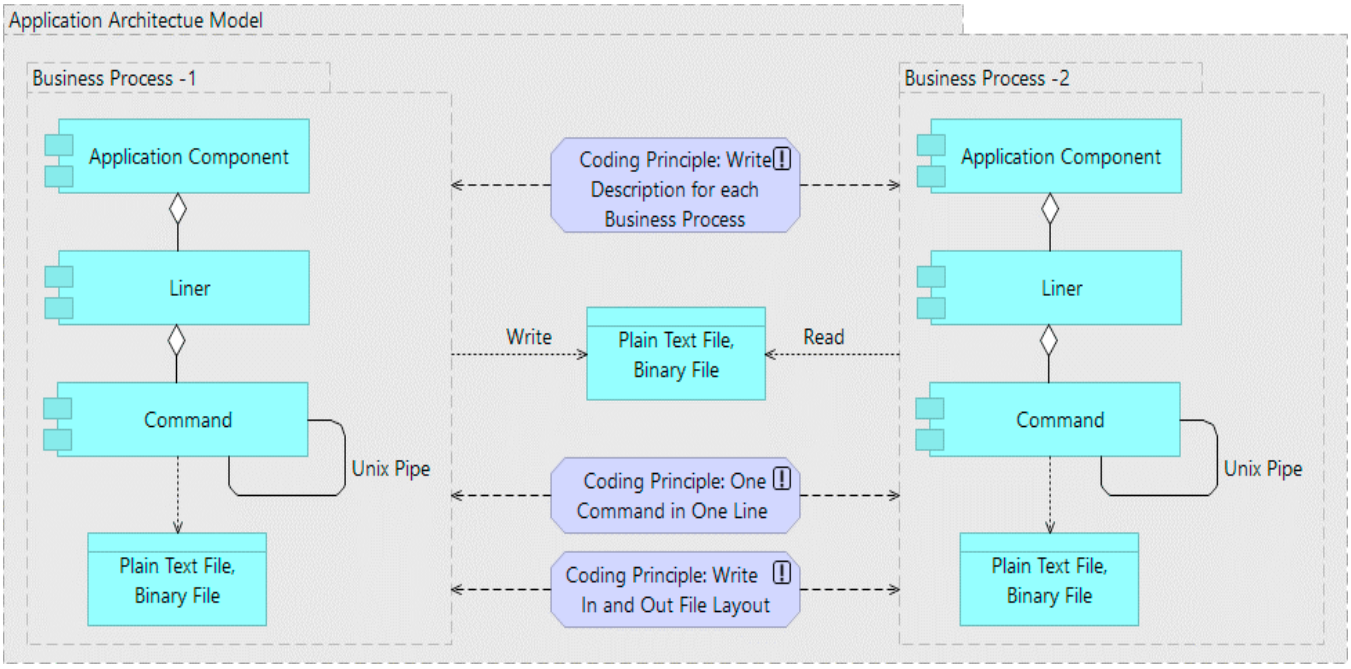

図 3.ArchiMateのアプリケーションアーキテクチャメタモデル

3.4. UAにおけるテクノロジーアーキテクチャモデル

図4は、IOデータに基づくコマンドが中核となるテクノロジーアーキテクチャモデルを示しています。
コマンドは、POSIX（ポータブルオペレーティングシステムインターフェース）に基づいたオペレーションシステムのコアライブラリを使用して実装されます。コマンドは任意のプログラミング言語で実装することができ、以下のようなシンプルなコーディング構文またはインターフェースを提供しながら、高度なデータ処理アルゴリズムを実行します。
コマンドは、コンピュータプログラミング言語で実装された場合には長いコードが必要になるかもしれない複雑なアルゴリズムを実装するソフトウェアクッションとして機能します。

コマンド構文：{コマンド}、{パラメータ}、{データ}

コマンドまたはツールは、以下の原則に準拠する必要があります。

原則1：コマンドの実装には、コアで高度なPOSIXライブラリのみを使用します。

原則2：上記のコマンド構文のように、コマンド構文を正規化します。

原則3：コマンドの数はできるだけ少なく保ちます。したがって、新しいコマンドを作成する前に、必要なデータ処理を解決するためにコマンドの組み合わせを試みます。

上記のソフトウェアクッションと原則に感謝することで、UAのテクノロジーアーキテクチャは、EAP[2]がオープンシステムの概念を適用することを提案する基準を満たします。つまり、オペレーティングシステムは以下のようになるべきです。
ポータブル：複数のベンダープラットフォームで実行します。

スケーラブル：小さなコンピュータから大きなコンピュータまで、幅広いパワーレンジで実行します。
相互運用可能：異質な環境で実行します。
互換性：既存のソフトウェアへの投資を保持し、他のコンポーネントとの技術進歩を統合できるようにします。

表3は、コマンドセットの分類を説明し、図4はArchiMateでのテクノロジーアーキテクチャモデルを提案しています。

表3.IO Data-based コマンドカテゴリ

カテゴリー名	用法
データ操作	データの選択、削除、並び替え、結合、分割
計算	数学的、統計的、要約、比較
フォーマット化	データ変換、文字列操作
入出力コントローラ	データ検索、送信
システムプロセス制御	ジョブ制御、システム連携、アクセス制御

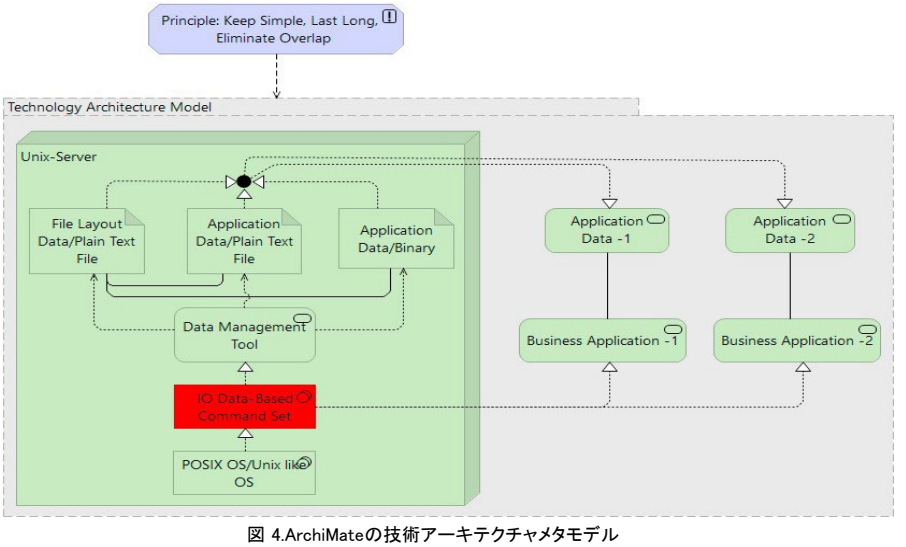

図 4.ArchiMateの技術アーキテクチャメタモデル

4.ケーススタディ

酒屋問題

私たちは、ArchiMateで示すことにより、その適応性を示すためにソフトウェア設計問題にUAを適用します。この問題は「酒屋問題[6]」と呼ばれ、店舗倉庫での注文および在庫管理プロセスを実現するソフトウェア設計に挑戦しています。

表4は、酒を含むコンテナを受け取ることから、酒の注文を出荷すること、空のコンテナ管理、注文を行う顧客に連絡するまでの管理プロセスの流れを記述しています。

表4.酒屋発注・在庫管理プロセス

プロセスアクター	ビジネスプロセス	情報を次工程へ出力 ・入力
倉庫管理者	物流コンテナを受け取り、倉庫に配置します。	コンテナの受領書を注文ディスパッチャーに渡します。製品受領情報: {コンテナ番号、配送日、酒の名前、酒の数量}
注文ディスパッチャー	1日に10件の注文を受けます。	コンテナから酒を取り出して各注文を満たすよう倉庫管理者に依頼します。依頼は電話または依頼フォームを通じて行われます。 注文情報: {酒の名前、数量、配送先住所} 依頼フォーム情報: {注文番号、配送先住所、コンテナ番号、酒の名前、数量、空のコンテナのマーク} 在庫不足または数量不足で注文した酒を満たすことができない場合は、顧客に電話をかけ、それを在庫不足リストに記入します。 不足リスト情報: {配送先住所、酒の名前、不足量}
注文ディスパッチャー	不足リストのお酒が入荷したか確認します。	不足している酒が入荷したら、倉庫管理者に不足リストへの記入を依頼します。
注文ディスパッチャー	コンテナが空になる予定かどうかをチェックします。	倉庫管理者にコンテナ情報を連絡し、倉庫内のコンテナ数をできるだけ少なくします。

問題で分析され、DDPDM【6】によって最適化されるビジネスプロセスを仮定します。
これにより、プロセスの数が１０から８に　最適化され、提案されるArchiMateメタモデルに結果が反映されます。

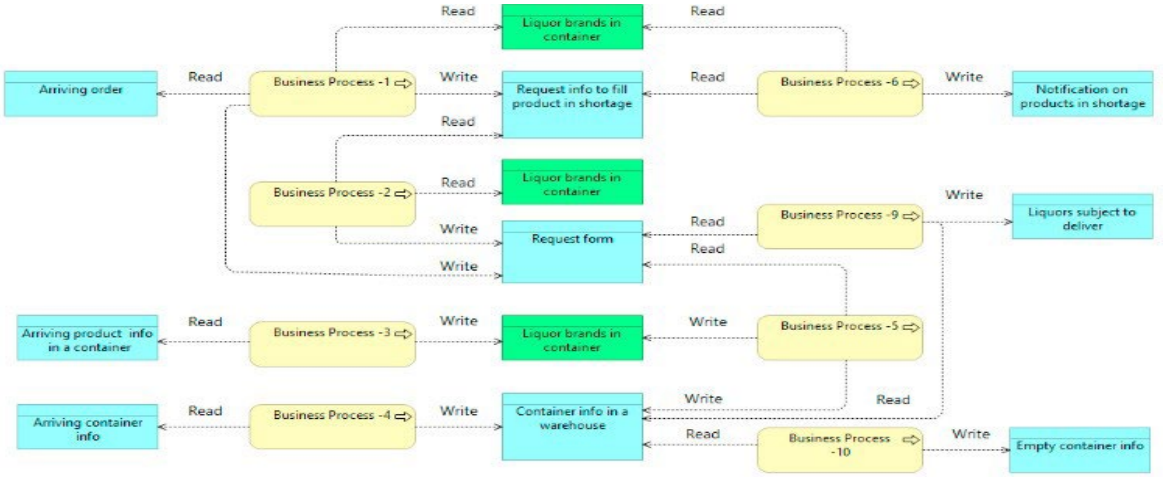

図5.ArchiMateにおける酒屋発注・在庫管理プロセス

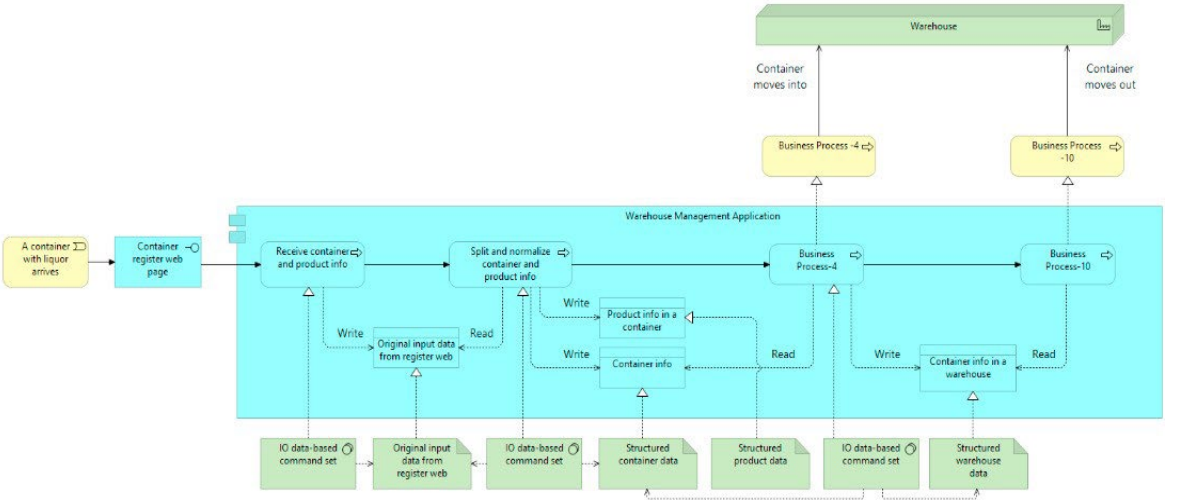

Fig.6.ArchiMateにおけるUAにおけるプロセス実装

4.2. ArchiMate記法

図5は、DDPDM[6]を適用した後の注文及び在庫管理プロセスを示しています。ArchiMateモデルの簡素化のために、「コンテナ内の酒ブランド」データオブジェクトが複製されています。
図6は、プロセスの一部を実装するUAの能力を実演しています。

5.議論

5.1 革新性

主要な企業アーキテクチャフレームワークはフレームワークの成果物のセットを定義する一方で、ビルディングブロックの実践的な実装は比較的少なく言及されています。本論文では、TOGAF[3]が推奨するコアアーキテクチャレイヤーを実装するためのビルディングブロックを提案し、ケーススタディを通じてその適応性を示しました。

5.2 効果性

提案されたアーキテクチャは、IOデータに基づいてビジネスプロセスを分析し、データのみを介してビジネスプロセスに直接マッピングするアプリケーションコンポーネントを実装するために効果的に適用されています。
また、アーキテクチャがマイクロサービスまたは疎結合アーキテクチャであることも示されています。
ArchiMateでのTOGAFアーキテクチャレイヤーメタモデルと共に提案されたメタモデルも、ケーススタディにおけるアーキテクチャの適応性を実演するために使用されています。

5.3. 制限

本研究で提案したアーキテクチャは、ArchiMateを用いた図解を伴うケーススタディに効果的に応用され、既に多くの企業で採用されています。しかしながら、このアーキテクチャはとりわけ統合的リソース計画（ERP）のために設計されたものです。アーキテクチャ全体に対応する統合されたArchiMateメタモデルを作成する必要があり、また、その有効性について定量的な評価を行う必要があると思われます。

6 結論

本論文で我々は、ArchiMateメタモデルを活用した実用的な実装ブロックを含むデータ駆動型のエンタープライズアーキテクチャを提案しました。

今後の課題としては、提案したArchiMateモデルを用いてさらなるケーススタディを図解し、アーキテクチャ全体に対する統合されたArchiMateメタモデルを設計する作業が予定されています。

参考文献

1 Phillip Beauvoir & Jean-Baptiste Sarrodie.(2022) ”Archi -archimate モデリング”、Archi バージョン:4.9.3 ArchiMate®3.1 言語をサポート。

2 Steven H. SpewakとSteven C. Hill。(2008) ”Enterprise Architecture Planning: Developing a Blueprint for Data, Applications, and Technology”, John Wiley & Sons, Inc.

3 オープングループ。(2018) ”The TOGAF® Standard, Version 9.2”

4 山本修一郎、Nada Ibrahem Olayan、森崎修司。(2018) ”Another Look at Enterprise Architecture Framework”, Journal of Business Theory and Practice: Vol. 6, No. 2, 2018.

5 山本修一郎。(2020) ”A A DX Visualization Approach Using ArchiMate”, 情報処理学会論文集: Vol.2020-SE-204 No.18.

6 山本修一郎、細見順子。(2022) ”A Proposal on Data Driven Process Design Method”, IEICE Technical Report KBSE2022-24: vol. 122, 第139,79-84号

7 フィリップ・フィッシャー=オグデン、グレッグ・バレル、ダイアン・マーシュ(2018) ”Full Cycle Developers at Netflix – Operate What You Build”, Netflix Technology Blog.

8 ボストンとビヨンドにおける計算研究(CRIBB)(2013) ”Hadoopやビッグアイアンを使わずに1秒以内に500億レコードを分析する方法”、https://math.mit.edu/crib/2013/aug2.html

9 ジェームズ・P・ホートン他(2017年)「ソーシャルメディアにおける会話クラスターの進展の追跡」、https://doi.org/10.1177/0049124117729705

10ジョアン・M・P・モレイラヘレナ・ガルハルダヘレナ・ガルハルダミゲル・パルダルミゲル・パルダル(2018) ”LeanBench: IoTデータのバッチ処理とクエリ処理のためのソフトウェアスタックの比較”、DOI: 10.1016/j.procs.2018.04.067