B-006

ドメイン駆動設計を用いたリファクタリングにおけるレイヤードアーキテクチャの拡張と準形式的表現

Enhanced Layered Architectures and Semi-Formal Representations in Refactoring using Domain-Driven Design

上原 宗大 †　和﨑 克己 ††
Sota Uehara　Katsumi Wasaki

1　はじめに

リファクタリングとは，外部から見たときの振る舞いを保ちつつ，理解や修正が簡単になるように，ソフトウェアの内部構造を変化させることである．リファクタリングは，ソフトウェア開発中の要求事項への対応やバグの修正にも用いられる．その一方で，異なるリファクタリングを適用してしまうと適用前と異なるシステム構造となってしまう可能性があるため，リファクタリングプロセスは体系的に行われなければならない．一般的には既存のコードの整理についてをリファクタリングとすることが多いが，本研究では仕様に対するリファクタリングを対象とし，そのリファクタリングに対しドメイン駆動設計で用いられるレイヤードアーキテクチャを用いることを提案する．

2　ドメイン駆動設計とレイヤードアーキテクチャ

ドメイン駆動設計 (DDD) は，エリック・エヴァンス氏が 2003 年に提唱したソフトウェア設計手法である [1][2]．ここでのドメインとはプログラムを適用する対象となる領域のことを指し，これは DDD において最も重要視される．DDD は技術志向の設計手法では無く，ビジネスドメインの知識に重点を置いた設計手法となっており，ドメインエキスパートと呼ばれる対象のドメインについて最も詳しい専門家とソフトウェア開発者の間で対話を繰り返しながらソフトウェア開発が行われる．対話の際にはユビキタス言語と呼ばれる共通言語が用いられ，ドメインモデルを作成しながら戦略的にドメインを理解してソフトウェア開発が行われる．2003 年に提唱された DDD であるが，手法としての完成度の高さから近年再注目されている．

レイヤードアーキテクチャは DDD に登場するアーキテクチャの中で，最も伝統的でかつ最も有名なアーキテクチャである [2]．上位層から下位層への依存関係がある．各層の説明は以下の通りである．

・ユーザーインターフェース層 (プレゼンテーション層)

ユーザーインターフェース (UI) とアプリケーションを結びつける層である．主な責務はシステム利用者への表示と，システム利用者からの入力の解釈である．

・アプリケーション層

ドメイン層を取りまとめる層であり，アプリケーションサービスがこの層に格納される．ドメイン層のオブジェクトはドメイン知識の表現に徹しているため，アプリケーション層から問題解決に導く．

・ドメイン層

レイヤードアーキテクチャの中で最も重要な層である．ドメイン知識はこの層の中に格納され，他の層へドメイン知識を流出させてはならない．

・インフラストラクチャ層

他の層を支える技術的基盤へのアクセスを提供する層である．リポジトリはこの層に格納され，データベースへのアクセスなどはこの層で実行される．

レイヤードアーキテクチャのみではドメイン層へのドメイン知識のカプセル化が難しいため，次章の境界付けられたコンテキストを用いる．

3　境界付けられたコンテキスト (Bounded Context)

境界付けられたコンテキストとは，ドメイン同士の境界である．複数のモデルが存在する大規模なシステムの完全な統合は難しく費用対効果が低いため，コンテキストを分割しそれぞれの中でモデルやユビキタス言語の統一を目指す．

Ozan Özkan らの論文 “Refactoring with domain-driven design in an industrial context”[3] 中では下のコンテクスト分割の類型として図 1 に示すようなアーキテクチャが提案されている．

図 1 において，3 つのコンテキスト層に分割されてい

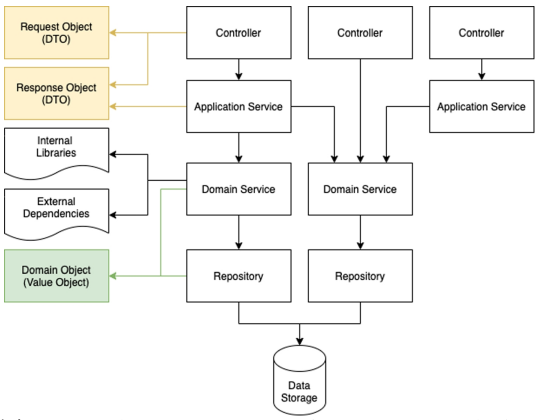

図 1　バックエンドのソフトウェアアーキテクチャ ([3] より引用)

† 信州大学大学院総合理工学研究科, Graduate School of Science and Technology, Shinshu University
†† 信州大学工学部電子情報システム工学科, Department of Electrical and Computer Engineering, Faculty of Engineering, Shinshu University

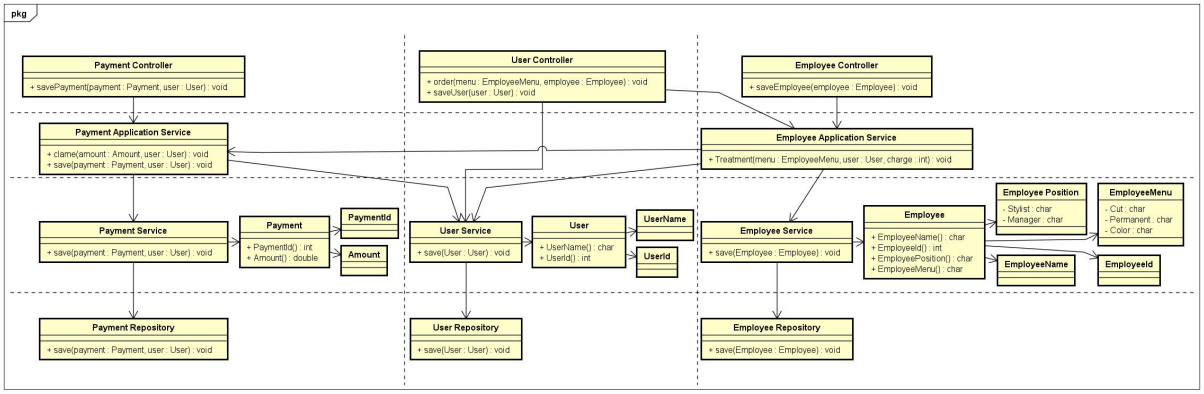

図 2　マルチコンテキストとして水平拡張を行ったドメインモデル図の例

る．記されている 3 つのコンテキストのうち，一番左のコンテキストは各層にレイヤーがあるが，中央のコンテキストではユースケースが不要なため，アプリケーション層の記述がない．一番右のコンテキストではドメイン知識が不要なため，ドメイン層やインフラストラクチャ層の記述がない．このように境界付けられたコンテキストを用いることで大規模システムを独立したコンテキストに分割し，システムの複雑さを減少させ，保守性の向上を図ることができる．

4　ドメイン駆動設計を用いたリファクタリング

本研究では，既存研究 [3] のコンテキスト分割法をベースとして，マルチコンテキストとしてのモデルの水平拡張を行った．また，自身が DDD における開発者兼ドメインエキスパートとしてモデル図を作成できるよう，身近なモデルより理容店を評価ドメインとして例示した．このモデルにおける条件は以下のとおりとする．

- 利用ユーザーは従業員にメニューを指定する．
- 利用ユーザーは従業員を指定できる．
- メニューは，カット・パーマ・カラーの 3 つがある．
- 従業員にはポジションがある．
- 金銭の授受は支払いサービスが行う．

以上を用いてドメインモデル図を作成すると，図 2 のようになる．コンテキストは，Payment・User・Employee の 3 つとし，そのそれぞれに対してレイヤードアーキテクチャを適用した．User コンテキストではユースケースが不要であるため，アプリケーション層の記述はない．上記の条件のうち，メニューや従業員のポジションはドメイン知識であるため，ドメイン層に格納する．各ドメイン層においては集約となっており，Employee コンテキストを例にすると，Employee のエンティティがルートとなっており，そのルートを通じて EmployeePosition や EmployeeName などの値オブジェクトにアクセス出来る．

Employee のルートへは EmployeeService のインターフェイスを通じて EmployeeApplicationService からアクセスできる．インフラストラクチャ層のリポジトリは，それぞれデータベースへのアクセスを担う．UserController から EmployeeApplicationservice へ，利用ユーザーは従業員とメニューを指定できる．EmployeeApplicationservice からは EmployeeService の呼出し，PaymentApplicationService へ料金の通知，User へのサービスの提供が行われる．

このように，DDD のレイヤードアーキテクチャを活用し，境界付けられたコンテキストを活用することで，アクターが複雑なドメインに対してもモデル化することができた．コンテキストを 3 つに分けることで，ドメイン知識の管理が容易になり，保守性や拡張性が向上した．

5　準形式化手法について

現状のドメインモデル図においては，形式的な記述に対応していない．準形式的表現にあたって，レイヤードアーキテクチャの後に DDD への適用が提唱されたヘキサゴナルアーキテクチャ [4] やその派生であるオニオンアーキテクチャ [5] の適用を検討している．これらのアーキテクチャは依存関係逆転の法則が用いられており，ドメイン層がインフラストラクチャ層に依存しているという問題も同時に解消できる．ヘキサゴナルアーキテクチャではドメインロジックがインフラストラクチャ層から分離されており，テストが容易となっている．オニオンアーキテクチャにおいては最上位層にユーザーインターフェース層・インフラストラクチャ層の他にアーキテクチャの中にテスト層があり，これを用いることで，アプリケーション層へのテストを考えている．

6　まとめと今後の課題

本研究では、DDD の設計手法を用いてレイヤードアーキテクチャを拡張する方法を検討し，コンテキストの拡張がアーキテクチャの再利用性とドメイン知識の管理に有効であることを示した．これにより、システムの保守性と拡張性が向上し、複雑なドメイン知識を効果的に扱うことが可能となった．今後は，より複雑なドメインにに対する検証に加え，最適なアーキテクチャの検討や準形式化へ向けて更なる手法の検討を行う予定である．

参考文献

[1] エリック・エヴァンス，今関剛監訳 (2011)，エリック・エヴァンスのドメイン駆動設計，翔泳社
[2] 成瀬允宣 (2020)，ドメイン駆動設計入門，翔泳社
[3] Ozan Özkan, Önder Babur, Mark van den Brand: “Refactoring with domain-driven design in an industrial context”, Empirical Software Engineering, 28:94, February 2023.
[4] ヴォーン・ヴァーノン，髙木正弘訳 (2015)，実践ドメイン駆動設計，翔泳社
[5] 松岡幸一郎 (2020)，ドメイン駆動設計 モデリング/実装ガイド