3L-4

CRUD分析中心の業務ロジックモデリングと機能型プロトタイプ自動生成

奥田 博隆† 小形 真平†† 松浦 佐江子††
†芝浦工業大学 システム工学部 電子情報システム学科
††芝浦工業大学 大学院工学研究科 機能制御システム専攻

1. はじめに

システム開発では，顧客と開発者間で共通理解を得られず，顧客の要求が最終成果物に反映されない事がある．そこで，顧客との共通理解を得る為にプロトタイピング手法がある．

我々は，業務系 Web アプリケーションを対象に UML(Unified Modeling Language)を用いたユースケース駆動要求分析手法(以下，従来手法)を提案してきた[1]．従来手法では，要求仕様化段階で顧客と開発者がシステムの入出力データとサービス手順に注目して妥当性確認を行う為に，UML 要求分析モデル(以下，従来モデル)を定義し UI(User Interface)プロトタイプの自動生成を行う．

しかし，従来手法では，要件の共通理解を得る事を目的として，特定の操作状況における「入力に対する期待される結果」の「想定」を実現する業務ロジックの振舞いやデータフローが非形式にモデル化し，そこからプロトタイプを生成するため，業務ロジックフローをたどる事で起こるデータライフサイクルの実現可能性を保証できない問題がある．

その為，設計及び実装工程で手戻りが発生する可能性がある．そこで，従来モデルの業務ロジック定義を洗練する手法や，洗練されたモデルに系統的にマッピングされる1つのフレームワーク(図 1)を用いて「CRUD に関わらない機能」以外の機能を実現し，実現可能性を示す手法を提案する．

本稿では，本学授業のソフトウェア開発演習内で使用されるグループワーク支援システムを適用事例として，提案手法の有効性を議論する．

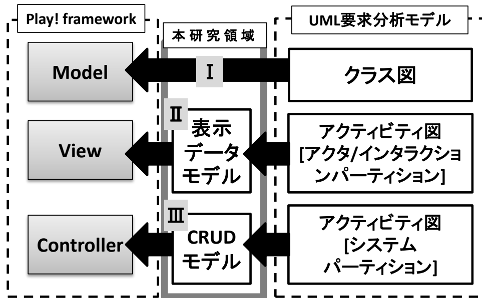

図 1:系統的マッピングの対応図

2. 従来モデルからのプロトタイプ生成手法

我々は分割・統治で洗練したユースケースをサービス単位として，アクタとシステムのインタラクション(以下，単にインタラクション)に着目し，振舞いとデータを分析・定義する手法を提案してきた．振舞いは UML アクティビティ図によって定義する．

表 1:アクティビティ図定義ルール

アクタの行為(A)	インタラクション(B)	システムが行う行為(C)
データ入力(1) 動作選択(2)	入力データ精査(3) 入出力データ定義(4)	業務ロジック(*)(5)

(*)入力の有無や値の形式・範囲の正しい入力データに基づく

表 1 から UML アクティビティ図を用いて(A), (B), (C)をパーティションとして定義し，(1)-(3), (5)を「～を～する」の形のアクションとして定義し，(4)をオブジェクトノードとして定義する．オブジェクトノードは，データ構造をクラス図に定義し，具体値をオブジェクト図に定義する．グループワーク支援システムの「個人情報を修正する」というサービスの具体例を以下に示す．(図 2，図 3)

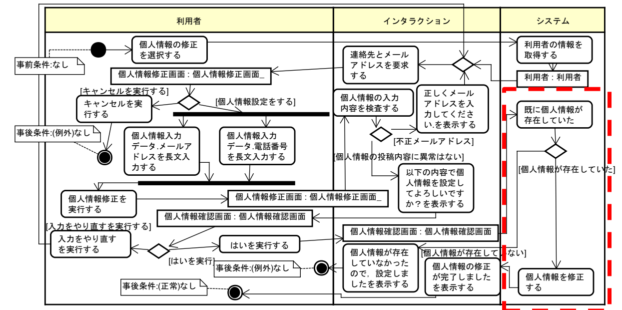

図 2:サービス単位のインタラクションアクティビティ図

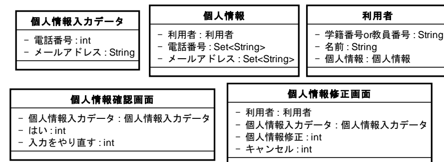

図 3:オブジェクトノードのデータ構造定義したクラス図

3. 従来手法の問題点

従来手法では，要件を十分具体的に理解できるように，特定の操作状況に対し「入力に対する期待される結果」の「想定」を具体値を用いシナリオとして定義する．

業務ロジックでは，シナリオにより想定したユーザの「入力」から「期待される結果」を出力する処理をサービス全体において一貫して要件を満たすように定義しなければならない．しかし，従来手法における業務ロジックの「想定」は，「入力」から「期待される結果」を出力できるとは限らない為，想定通りにデータライフサイクルが成立しない可能性が高く，実現可能性が保証できないことが問題である．

4. 実現可能性を示す提案手法

MVC(Model View Controller)アーキテクチャ採用の Java 言語 Web アプリケーションのフレームワークである play!framework[2]上にアプリケーションとして機能型プロトタイプを従来モデルから自動生成する．play!framework は，O/R マッピング，Web Server 等を内包し，オブジェクト指向に基づく Web アプリケーションを容易に構築できる．そこで，play!framework と従来モデルとの対応関係を構築するため図 1 のⅠ～Ⅲにある中間表現を用い従来モデルの洗練を図る．そして，データ永続化機構を用いてデータ永続化を伴う機能型プロトタイプを自動生成する事で実現可能性を示す．

従来手法では曖昧に定義されるシステムパーティション(以降，システム内部)の業務ロジックに対し，本稿の系統的な洗練手法を用いてデータライフサイクルの観点から曖昧性を排除し実現可能性を保証する．

5. 従来モデルの系統的な洗練による解決方法

図 1 の(Ⅰ)-(Ⅲ)に示す中間表現を経てフレームワークと従

Business logic modeling of CURD analytics and automatically generating functional prototype
† Hirotaka Okuda †† Shinpei Ogata †† Saeko Matsuura
†Department of Electronic Information System, collage of System Engineering, Shibaura Institute of Technology
††Division of Functional Control Systems, Graduate School of Engineering, Shibaura Institute of Technology

Copyright ©2011 Information Processing Society of Japan. All Rights Reserved.

来モデルとの差異を埋める事を目指す．その為，図 1 に示す系統的な洗練を行う必要がある．従来モデルでは，ユーザへの入出力 UI を意図する「入出力データ定義」，業務ロジック実行の為の「入力の有無や値の形式・範囲の正しい入力データ」，業務ロジック実行後の「出力データ」が夫々オブジェクトノードとして定義される．

5.1. クラス図からの Model 変換（Ⅰ）

クラス図からの Model 変換(図 1 のⅠ)は，MVC の Model との対応関係を構築する．特にフレームワークでの Model はデータが妥当な値の保持や指定された要件を満たす事の保証とオブジェクトの永続化の為に JPA(Java Persistence API)を通して Hibernate[3]を使用する為の用意が必要である． システム内部でアクション中に出現するクラスをエンティティとして永続化対象とする．その為に従来モデルの各属性に定義された制約事項の定義を play!framework の制約(数の範囲，最大値，正規表現に一致するなど)にマッピングし，クラス間の多重度から JPA のアノテーションを自動的に付加し Model 要素に変換する．

5.2. 表示データモデルの定義（Ⅱ）

表示データモデル(図 1のⅡ)は，MVC の View に対して動的に出力が変化する箇所を明確に定義する為，対応関係を構築する．特にフレームワークでの View は，クラス図とアクティビティ図で定義される入出力データとシステム内部のエンティティとの対応関係を持つ事で動的な変化に対応した View の生成ができる．

そこで，表示データモデルでは，入出力データとして定義されたオブジェクトノードから動的な Web ページ生成の為にクラスを次の様に分類して属性として再定義する．

- 「インプット」―従来手法で HTML の<input>タグとしてプロトタイプに出力されるもの．従来手法による定義でアクタパーティションのアクション記述と対応する．
- 「表示データ」―システム内部に存在するエンティティとの対応関係を持つデータである．入出力データはシステム内部で使用されるエンティティと必ずしも一致しない為，対応関係を構築する為に属性単位に OCL(Object Constraint Language)の派生関係の定義を利用して記述する．複数個データが存在する場合は List 型を用いる．

例として図 2 のデータ入力の定義である[個人情報修正画面]に対して上記の定義を用いた例を示す(図 4)．属性の[名前]，[学籍番号]，[電話番号]，[メールアドレス]は表示データに相当し，[個人情報修正]，[キャンセル]はインプットに相当する．

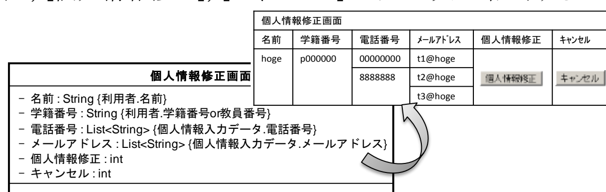

図 4:表示データ定義による[個人情報修正画面]の再定義と生成画面イメージ

5.3. CRUD モデルの定義（Ⅲ）

CRUD モデル(図 1 のⅢ)とは，MVC の Controller と対応しており，システムパーティション内部のロジックアクション群を対象とし，データのライフサイクルを明確にモデル化する為，基本的なデータの処理単位である CRUD の観点からフレームワークに存在する永続化機構との対応関係を構築出来るように業務ロジックの振舞いの曖昧性を排除するモデルである．

CRUD モデル定義は表示データモデルの定義と同様の前提である．CRUD モデルでは，システム内部に存在するロジックアクションの末尾の動詞の役割に注目して次の様に分類を与える．(以下，基本 3 アクションと呼ぶ)

- 「CRUD」 - アクション末尾の動詞が「～を作成する」，「～を更新する」，「～を更新する」，「～を削除する」で終わるもので， 永続化動作を明示的行うアクションのことである．
- 「状態変化」 −業務ロジックのルールに与える変化をオブジェクトノードに及ぼすアクションのことである．
- 「条件判定」 - アクション末尾の動詞が「～を判定する」で終わるもので，業務ロジックの条件判定となるアクション，すなわち業務ルールに相当するアクションのことである．

詳細化の対象は，図 2 中の点線内の様にシステム内部に出入りするフローに注目し，それらのアクション群とオブジェクトノード群の接続関係を持つノードをメソッド単位として切り出す．そして，基本 3 アクションに対してアクションが実行前に定義されたオブジェクトノードの持つ属性を使用して，アクションが実行できるかを確認する．それを繰り返しアクションの分割，オブジェクトノードの再定義を行い，洗練を加えた結果以下のようになる．(図 5)

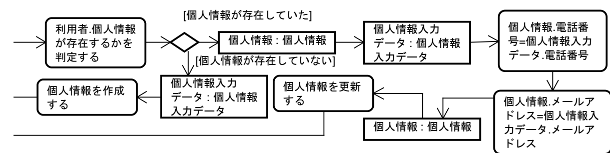

図 5:CRUD モデルによって洗練化されたアクティビティ図

例えば，洗練化されたモデルから[個人情報を作成する]アクションは，「new 個人情報(個人情報入力データ．メールアドレス，個人情報入力データ．電話番号)．save();」の様にソースコードを生成する．

5.4. CRUD に関わらない機能

CRUD 以外の機能とは，従来モデルのアクティビティ図中のアクションでは表現困難な機能を指す．特に「メール送信機能」等の業務ロジックから独立可能な機能や複雑なアルゴリズムを要する機能に分類されるものである．

モジュール的に提供される機能は，業務ロジックに影響を及ぼさない為，機能型プロトタイプの適応範囲外とする．一方，複雑なアルゴリズムを要する機能は，入力値に関わらず出力に固定値を定義する事によって機能をブラックボックス化し，機能型プロトタイプに盛り込む．

6. 機能型プロトタイプを使用した CRUD モデルの検証

CRUD モデルを定義過程では，システム内部のモデルをオブジェクトノードとアクションとの対応関係を形式的に定義する事で曖昧性が排除され，コード化を念頭に詳細化を行うため粒度が統一される．

機能型プロトタイプでは，テストデータを使用して想定される入力を行い，意図通りに業務ロジックが動作しているかを目視確認できる．エンティティに対して永続化を行い，実現可能性の検証を行う．その為，ロジック部分に不備が存在すれば，業務ロジックが意図通りに実行されないことが予想されアクティビティ図の業務ルールの誤解や記述の曖昧さに気付く．実際にグループワーク支援システムの「掲示板」サービスでは， 出力データとして投稿時間の表示を定義したにも関わらず，システム内部で「投稿時間を付加する」という事を行わなかった為，View に反映されずに誤りに気づいた．

7. まとめ

本稿では従来モデルの系統的な洗練によって従来手法の問題点である業務ロジック要件の曖昧性の排除を行った．その結果，機能型プロトタイプを自動生成し，データライフサイクルから見る業務ロジックの実現可能性を示せた．

今後は， CRUD モデルの「検索」時の検索条件の定義法を考えていく．また，より高い実現可能性の保証を複数のフレームワークに対応していく．

8. 参考文献

[1] 小形真平，松浦佐江子: UML 要求分析モデルからの段階的な従来手法，コンピュータソフトウェア，Vol. 27(2010)，No. 2，pp. 14-32

[2] play!framework (http://www. playframework. org/)

[3] Hibernate (http://www. hibernate. org/)

Copyright ©2011 Information Processing Society of Japan.
All Rights Reserved.