財務会計システムにおける
アプリケーションの自動生成に関する研究

M2013SE007 村上 晃輝
指導教員：野呂 昌満

1　はじめに

アーキテクチャに基づくソフトウェア開発において，アプリケーションフレームワークを利用した開発が一般的である．ドメインを限定した際，アーキテクチャ上には不変部分と変動部分が存在する．アプリケーションフレームワークには，アーキテクチャ上の不変部分と変動部分に対応する箇所があり，それぞれフローズンスポットとホットスポットと呼ばれている．フローズンスポットは，さらに 2 種類に分類される．1 つ目は対象ドメインのアプリケーション全てに共通する部分で，本研究では完全フローズンスポットと呼ぶ．2 つ目は仕様上の共通部分で，実現レベルでは対象ドメインのアプリケーションごとに異なる部分だが抽象化した対象ドメインのアプリケーションの共通部分である．本研究ではこの部分をコード自動生成可能フローズンスポットと呼ぶ．本研究では，Service Oriented Architecture(以下，SOA) に基づくアプリケーション開発を前提としており，ビジネスロジックを実現している Action 記述がホットスポットとなる．

アプリケーションの完全自動生成を目標とした場合，ビジネスロジックのコード自動生成は一般的には困難とされている．その理由としては，ホットスポットはアプリケーションごとに異なることからコード自動生成の手法が一般化されていないということが挙げられる．

本研究の目的は，ホットスポットであるビジネスロジックの標準化によるコード自動生成の支援である．ビジネスロジックを組み立てる基準を提案してそれぞれを部品化することで，定型コードとの対応付けが可能となり，コード自動生成の支援に繋がる．

本研究で取り扱う問題点の解決策として，ロジックパターンを提案する．ロジックパターンは，頻出するロジックをパターンとして定義したものである．パターンを用いてロジックを表現することにより，ビジネスロジックの標準化を行なうことができると考える．

ロジックパターンの抽出は，アプリケーションドメインを財務会計システムに限定して行なう．ドメインを限定することで，抽象度の低いロジックパターンを定義することができ，具体的なロジックのコード自動生成が可能となる．財務会計処理は簡潔な処理が多いことからロジックパターンを抽出しやすいと考え，本研究では対象ドメインを財務会計システムに限定する．ロジックパターンは，既存システムのコードの分析により抽出する．抽出されたロジックパターンを抽象化し，デザインパターン [1] を適用する．デザインパターンを適用することで関心事を分離することができ，ロジックの部品化が可能となる．事例検証として既存システムの分析を行ない，ロジックパターンの適用可能性の分析を行なう．

2　SOA に基づくアプリケーション開発におけるアプリケーションフレームワーク

アプリケーションフレームワークとは，オブジェクト指向に基づいたアプリケーションの再利用技術である．開発者はホットスポットと呼ばれるアプリケーション固有の部分のみを作成することで，アプリケーションを完成させることができる．

SOA に基づくアプリケーション開発におけるアプリケーションフレームワークの例を図 1 に示す．各 Abstract クラスは対象ドメイン固有の部分であるので，完全フローズンスポットである．各 Concrete クラスは，実現レベルでは異なるが仕様上はこの構造は不変なので，自動生成可能フローズンスポットである．ConcreteAction クラスのメソッド内の Action 記述は，ビジネスロジックを実現する箇所でありアプリケーションごとに異なる部分なので，ホットスポットである．

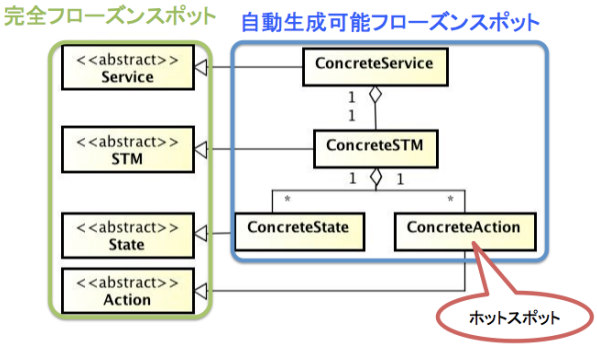

図 1　SOA に基づくアプリケーション開発におけるアプリケーションフレームワーク

3　財務会計処理のロジックパターンの定義

3.1　ロジックパターンの定義

本研究ではロジックパターンを「成功事例のロジックを集めて共通部分を抽象化して名前を与えた解法」と定義する．ここで言う「共通部分」は，特定のデータを対象とした処理の流れや処理の順序が相当し，「解法」は，特定のロジックを組み立てたいという状況において用いられる方法である．ロジックパターンの記述テンプレート [3] は，名前・状況・解決・フォース・アルゴリズム (表現方法) とする．ロジックパターンを表現する方法として，Unified Modeling Language(以下，UML) で定義されているアクティビティ図を採用する．アクティビティ図は UML の中で振舞いを表す図であり，ロジックを表すのに最適な図である．シーケンス図とは異なり通信相手のオブジェクト生成を明記する必要がないので，より粒度の

高いレベルでロジックを取り扱うことができる．ロジックパターンは，処理内容や処理データという付加情報を与えて具体化することによりビジネスロジックとなる．

3.2 アプリケーションドメインの限定

ホットスポットのコード自動生成の支援を行なうためには，アプリケーションドメインを限定する必要がある．アプリケーションドメインを限定せずにロジックパターンを定義すると，抽象度の高いロジックパターンとなってしまう．抽象度が高いということは，どのような場面でも適用できる反面，詳細なロジックを扱うことができなくなってしまう．出来る限り多くのビジネスロジックコードを取り扱う必要があるので，アプリケーションドメインを限定してロジックパターンを定義する．

本研究では，アプリケーションドメインを財務会計システムに限定してロジックパターンの定義を行なう．財務会計処理は簡潔な処理が多いことからロジックパターンを抽出しやすいと考え，本研究では対象ドメインを財務会計システムに限定する．本研究では Case Study の一例としてアプリケーションドメインを財務会計システムに限定する．本研究で実施した手順でロジックパターンを定義することで，他のアプリケーションドメインにおけるホットスポットのコード自動生成も可能になると考える．

3.3 事務処理のロジックパターンの定義

第 3.1 節の定義に基づき，本節では事務処理のロジックパターンについて定義する．このステップは，ビジネスロジックを自動生成するためのキーとなるロジックパターンをボトムアップにより定義するステップである．現行システムを分析し，定型コードを抽出する．抽出結果を記述テンプレートに従って記述し，事務処理のロジックパターンの定義を行なう．財務会計処理より抽象度の高い事務処理を先に対象とすることで，いきなり財務会計処理を対象とするよりもロジックパターンの共通部分を抽出しやすいと考える．

現行システムを分析した結果，事務処理のロジックは 3 種類に分類できることがわかった．データベース (以下，DB) への Query を行なうロジック，DB へデータを登録する Query を行なうロジック，DB にあるデータを更新する Query を行なうロジックの 3 種類である．事務処理ロジックの詳細を表 1 に示す．DBQuery は，DB 登録 Query と DB 更新 Query の抽象化したロジックである．

抽出した事務処理のロジックをアクティビティ図で表現する．DB 登録 Query をアクティビティ図で表現した結果を図 2 に示す．以下では，DB 登録 Query を対象としてロジックパターンの定義を行なう．DB 更新 Query と DBQuery のロジックパターンに関しては，紙面の都合上，省略する．

DB 登録 Query は主に 4 つのステップで構成される．メタデータの取得とは，DB に登録するデータのメタデータを取得する処理ステップである．DB に登録するデータが必要とするメタデータの数だけメタデータ取得という処理ステップを重ね，登録対象のデータを生成する処理ステップを実行する．最後に，生成された対象データを DB に登録する処理ステップを実行する．また，ロジックというアクティビティは，開発者が独自に記述する処理ステップであり，ロジックパターンのホットスポットである．本研究では，これを事務処理の DB 登録 Query のロジックパターンと定義する．

表 1 事務処理ロジックまとめ

ロジック名	項目	説明
DBQuery	概要	DB にアクセスする
	事前条件	特定の目的を持って DB へアクセスしようとしている
	事後条件	特定の目的を達成し，DB へのアクセスが完了している
DB 登録 Query	概要	対象データを DB へ登録する
	事前条件	対象データが DB に存在しない
	事後条件	対象データが DB に存在する
DB 更新 Query	概要	DB に存在する対象を更新する
	事前条件	対象データが DB に存在する
	事後条件	対象データが書き換えられる

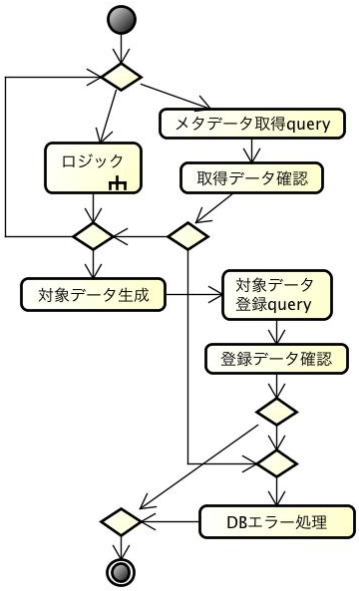

図 2 事務処理のロジックパターン (DB 登録 Query)

3.4 財務会計処理のロジックパターンの定義

本節では，図 2 の事務処理のロジックパターンを基に，財務会計処理のロジックパターンを定義する．事務処理のロジックパターンに財務会計特有の処理を付加することにより，財務会計処理のロジックパターンの定義を行なう．現行システムの分析の結果，財務会計システムの主な処理は，申請・承認・変更・取消・転送・確認の 6 種類であることがわかった．これらの処理に着目してロジックパターンの定義を行なう． 申請は，事務処理の登録の具象化された処理である．承認・変更・取消 / 削除・確認は，事務処理の更新の具象化された処理である．確認は，事務処理の DBQuery(DB アクセス) の具象化され

た処理である．以上のことから，各財務会計処理に当てはまる事務処理のロジックパターンを具象化することで，財務会計処理のロジックパターンの定義を行なう．

財務会計処理の 申請に関するロジックパターンを図 3 に示す．事務処理の DB 登録 Query パターンに基づき，再び現行システム分析することで，申請のロジックパターンを定義した．事務処理のロジックパターンと異なる部分は，実行処理の登録が申請になった点，申請が実行される前に，実行処理を確認する点である．大まかな処理の流れは事務処理の DB 登録 Query パターンと変わらない．本研究では，図 3 を「データ申請」ロジックパターンと定義する．記述テンプレートに従って定義したロジックパターンを表 2 に示す．

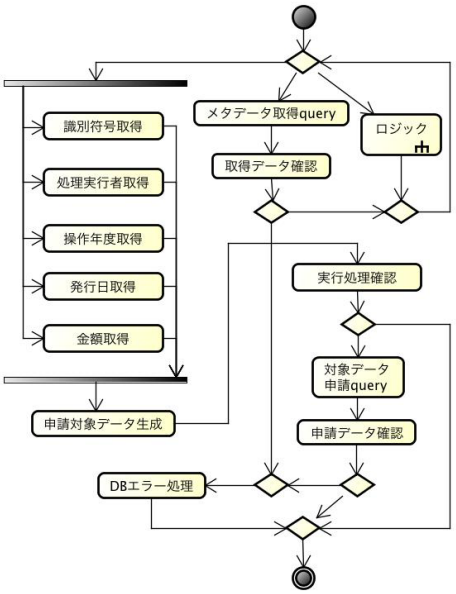

図 3 財務会計処理のロジックパターン (データ申請)

表 2 財務会計処理のロジックパターン (データ申請)

名前	データ申請
状況	指定データの申請を行なうロジックを組み立てたい
	申請の取消を考慮したロジックとしたい
	柔軟に変更可能な Action コードを作りたい
解決	指定データのメタデータを取得し，指定データを生成，申請を行なう
	適用デザインパターンを選択し，主要ロジックから特定の関心事を分離
フォース	申請というロジックは各アプリケーションに依存しており，同一の Action クラスにて申請を行なうと，申請処理自体の変更が困難となる
	申請の取消をする際，復元する情報を Action クラスで設定してしまうと，復元したい情報が固定となってしまう．
	申請対象データのメタデータ取得を Action クラスで行なうと，メタデータの取得方法を変更したい場合は Action コード自体を変更しなければならなくなってしまう
アルゴリズム	図 3 参照

3.5 ロジックパターンに対するデザインパターンの適用

本研究で定義したデータ申請ロジックパターンに適用可能なデザインパターンの一部を表 3 に示す．各デザインパターンが表現するロジックと，ロジックパターンの一部が表現するロジックが一致する場合，適用可能だと考える．

表 3 データ申請ロジックパターンに適用可能なデザインパターン

分類	パターン名	ロジック
生成	Abstract Factory	種類や部品の数が固定のオブジェクトの生成
	Factory Method	サブクラスを用いたインスタンスの生成
構造	Composite	木構造のオブジェクトの利用 (処理実行)
振舞い	Command	要求に見合った処理
	Memento	オブジェクトのスナップショットの保存，再利用
	Visitor	オペレーション委託によるオペレーションの実行

データ申請ロジックパターンにデザインパターンを適用した結果を図 4 に示す．図 4 では，Command パターンと Memento パターンの適用を行なった．Command パターン適用により分離されるロジックは，要求に応じた処理で，図 4 では「対象データ申請」と「申請取消の対象データの設定 (申請データの取消)」である．Memento パターンにより分離できるロジックは,「memento の生成」と「memento の取得」である．それぞれの分離できるロジックはビジネスロジックから分離されたので，ビジネスロジックに依存せずに変更することができる．

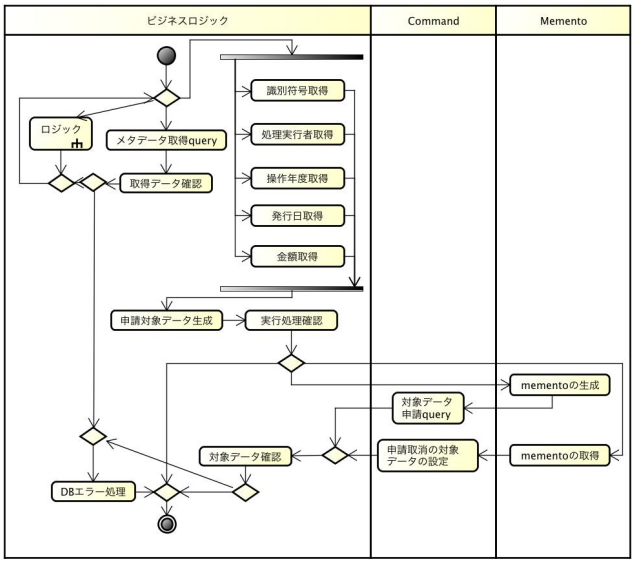

図 4 データ申請ロジックパターンに対するデザインパターンの適用

4　事例検証

本章では，定義したロジックパターンの適用可能性について検証する．現行システムに対してロジックパターンを適用し，ロジックパターンの適用可能性を分析する．現行システムのビジネスロジックを担うコードに注目し，コードをアクティビティ図で表現した際にロジックパターンが網羅可能なコード数について分析した．分析結果の一部を表 4 に示す．ここで言う適用可能性とは,「パターンの適用可能行数」を「ロジック部分のコードの行数」で割った値の小数点第二位を四捨五入したものである．

表 4　ロジックパターンの適用可能性に関する検証

ロジックパターン	ファイル名	ロジック部分のコードの行数	パターンの適用可能行数	適用可能性 (%)
データ申請	振替伝票申請	158	115	72.8
	支払伝票申請	260	161	61.9
	小目的科目申請	109	73	67.0
データ更新	振替伝票承認	82	54	65.9
	伝票承認	95	60	63.2
	目的予算一括承認	338	53	15.6
	単位間振替更新	77	71	92.2
	予算補正	105	55	52.4
	小目的科目承認解除	83	38	45.8
	管理階層更新	156	32	20.5

5　考察

5.1　ロジックパターンに対するデザインパターンの適用に関する考察

第 3.5 節で一般的な技術であるデザインパターン適用により，ロジックパターンの関心事を分離することができた．生成，振舞いに関するパターンを適用する場合，分離された関心事とデザインパターンの定型コードとの対応付けが可能となる．また，主要のロジックから関心事を分離できる．以上のことから，新たなアクティビティを部品として作成し，ロジックパターンに対して付け替えが可能となる．

本研究では全てのデザインパターンの適用をしていない．よって今後の課題として，適用可能な全てのデザインパターンを適用する必要がある．また，適用可能なデザインパターンを組み換え可能にしてロジックパターンを洗練することで，より柔軟なロジックパターンとする必要がある．

5.2　ロジックパターンの適用可能性に関する考察

第 4 章での検証の結果，本研究で定義したロジックパターンは現行システムのビジネスロジックコードの 50%程度適用可能ということがわかった．適用可能性の値が明らかに低いものがあるが，これはビジネスロジックに SQL に関するロジックが含まれていたからである．他のロジックでは，SQL に関するロジックはモデル内のメソッドに記述されている．このことから，本研究で定義したロジックパターンは SQL に関して考慮されていないことがわかった．以上のことから，アプリケーションドメインを財務会計システムに限定したときに本研究で定義したロジックパターンはビジネスロジックの 50%程度の適用可能性を秘めていることがわかった．また，ロジックパターンはビジネスロジックのコード自動生成の手がかりになるということがわかった．今後の課題として，今回対象とした既存システム以外の財務会計システムを対象としてロジックを抽出し，本研究で定義したロジックパターンを洗練する必要がある．

5.3　ビジネスロジックのコード自動生成に関する考察

本研究では，ビジネスロジックのコード自動生成の手がかりとなるロジックパターンの定義を行なった．デザインパターンを適用することで，ロジックパターンの構成要素と定型コードとの対応付けが可能となり，ロジックパターンからコード自動生成が可能であるということがわかった．ロジックパターンを用いてコード自動生成を実現することで，本来の目的であるアプリケーションの完全自動生成が達成できると考える．

ロジックパターンを用いたコード自動生成を実現するには，以下の項目を明確にする必要がある．これらの項目は，本研究の今後の課題となる．

- 各アクティビティ・制御フローとデザインパターンの定型コードとの関連付け
- コード自動生成ツールに対して必要となる入力
  - 適用ロジックパターン
  - 適用デザインパターン

6　おわりに

本研究では，ロジックパターンを定義することにより，アプリケーションの完全自動生成の支援を試みた．既存システムからロジックパターンを抽出してデザインパターンを適用することで，ロジックパターンにおける関心事を分離することができた．既存システムに対してロジックパターンを適用することにより，ロジックパターンの適用可能性について分析した．本研究の成果として，本研究で定義したロジックパターンがビジネスロジックのコード自動生成の手がかりとなることがわかった．今後の課題として，ロジックパターンの可変部分を分析し，定型コードとの対応関係を明確化することでビジネスロジックのコード自動生成を可能にすることが挙げられる．

参考文献

[1] E. Gamma, J. Vlissides, R. Helm, and R. Johnson, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995.

[2] 岡村美和，菊島靖弘，青山幹雄,“ 業務プロセスのパターン化による大規模業務システム開発の効率化とその評価，”ソフトウェア・エンジニアリングシンポジウム 2014 論文集，pp.9-14，2014.

[3] 鷲崎弘宜，丸山勝久，山本里枝子，久保淳人，ソフトウェアパターン， 近代科学社，2007.