RO-012

多品種対応ドメインを用いたモデル駆動開発の実現

A Realization Method of Model Driven Development using Domain with Variability

根路銘　崇†‡　　沼尾 雅之‡
Takashi Nerome　　Masayuki Numao

1．はじめに

近年の様々なビジネスの IT 化を背景に，多品種開発が可能なアプリケーション開発の柔軟性が求められている．多品種開発に対応可能なアプリケーション実行フレームワークに関しては，依存性注入技術[1]や，可変性に対応したルールエンジン[2]の技術が普及してきている．開発手法に関しては，組込み製品開発を主流に，派生開発技術を含むプロダクトライン開発の様々な技術が存在している[3].

しかしながら，アプリケーション実行フレームワークへの変換を可能とした，汎用的な多品種開発のためのアプリケーション開発手法が未だ存在していない．その理由としては，アプリケーション開発では，組込みのような物理的特性が存在しないために，設計開発対象の自由度が大きいことが主要な課題である．

我々は，この課題を解決するために，多品種開発を必要とする金融商品や保険商品に適用可能なアプリケーションのドメイン特性を定義し，多品種に対応するドメインモデル[4]の実現法を考案した．多品種開発に対応したモデリング設計のために，UML[5]を拡張したモデルを扱う．ドメインモデルの実現には，モデル駆動開発を取り込み多品種設計時の確認手法を提供する．また，モデル駆動技術[6]によって，依存性注入のための DI コンテナを保有するアプリケーション実行フレームワークのための変換を可能とする．これにより，本研究で定義するドメインモデルを用いて，PIM, PSM から実装コードまでの設計開発を可能とする．

以下，2 章で多品種開発のための要件と課題について述べ，3 章で我々の提案する可変性対応ドメインモデルの実現法について述べる． 4 章で金融商品ドメインでの実現例について述べし，最後に 5 章で本稿をまとめる．

2．多品種開発のための要件と課題

多品種開発のための主要な要件を以下に述べる．

- 要件 1. 設計の可視性
- 要件 2. 共通要素と可変点の抽出
- 要件 3. 派生要素の制約表現
- 要件 4. 資産の再利用
- 要件 5. アプリケーション実行環境

設計漏れや設計ミスの確認のためには要件 1.が必要であり，UML を用いたモデリング手法やドメイン駆動設計が多く普及している．要件 2.3.4.は，開発の冗長性を排除するために，プロダクトライン開発技術の特徴として有名である．要件 5.は，DI コンテナと呼ばれる依存性注入技術のためのフレームワークが存在する．

IT システムにおける多品種開発の課題について次に述べる．

- 課題 1. 設計開発の自由度が大きい

製造業向けの開発とは異なり，多品種開発のための物理制約がないために，設計開発の自由度が大きい．ドメインとしてアプリケーション構造を定義する動きがあるが，汎用的なアプローチは未だ存在していない．

- 課題 2. 従来型のプログラム開発への親和性

近年のグローバル開発を含む IT 系開発者の多様性により，多品種開発といえども，従来型のクラス図，シーケンス図を中心としたシンプルな開発スタイルが必要とされている．

- 課題 3. 多品種を同時に扱う実行環境用の開発手法

IT 系のアプリケーションでは，多品種が同時に同じ実行環境で実行される特徴を持つ．依存性注入技術のための DI コンテナがその基礎技術を支えられるものの，課題 1.と課題 2.により，DI コンテナを前提とした多品種対応のためのプロダクトライン開発手法が未だ存在していない．

2.1 関連研究

2.1.1 プロダクトライン開発

プロダクトライン開発は，再利用に基づく多品種開発のための開発方法論である．製造業における組込みシステムへの適用研究が特に盛んである[3].一方，IT システムへの適用についても近年関連研究が登場してきている[8][9]．プロダクトライン開発における再利用のための考え方としては，再利用資産を管理する解決空間と，派生のための可変性定義のための問題空間で区分けされる．問題空間における可変性は，フィーチャモデル[9]によって定義する．

2.1.2 依存性注入技術

依存性注入技術は，機能間の依存関係をプログラムから排除し，外部定義ファイルなどで依存関係を注入するためのソフトウェアパターンである[1]．依存性技術により，機能間の結合度を低下させることで，機能のコンポーネント化が促進できる．依存性注入技術を提供するフレームワークは DI(Dependency Injection)コンテナと呼ばれる．代表的な DI コンテナとしては，Spring[10]が挙げられる．

2.1.3 ドメイン駆動設計

ドメイン駆動設計は，"複雑なドメインの設計はモデルベースで行うべきであり、また大半のソフトウェアプロジェクトではシステムを実装するための特定の技術では

†日本アイ・ビー・エム（株）
‡電気通信大学大学院情報工学専攻

なくドメインそのものとドメインのロジックに焦点を置くべき"というパラダイムを持つ設計手法である[11]．特定のソフトウェアツールやフレームワークに依存せず，要件を定義する段階から，ドメインの構造と振舞いを定義していきドメインモデルを作成するためのデザイン・パターンである．

3. 可変性対応ドメインモデルの実現法

我々は，2 章で示す課題 1.に対して，派生付ドメイン特性を定義する事により，開発対象の基本構造アーキテクチャを提案することによる解決を提案する． 課題 2.に対しては，派生を UML オブジェクトの拡張で表現することで，従来の構造設計としてのクラス設計への影響を最小限とする．課題 3.に対しては，多品種へ対応した DI コンテナ用へのモデル変換技術を提案することで解決出来る．

実現のための開発ツールとは，IBM Rational Software Architect[12]を開発プラットフォームとして，実現のためのプラグインを開発した．

本章では，課題への解決するための可変性対応ドメインモデルの実現法について述べる．ドメインモデル自体の実行ロジックに関する実行環境については，本稿では省略する．

3.1 派生付ドメイン特性

本件研究で対象とする派生付ドメイン特性について，図 1 にメタモデルを示す．ドメインは，ドメイン属性と機能呼出しを保有する．機能は機能呼出しから保有され，機能自体も機能属性と機能呼出しを保有する．ドメイン自体は<派生>表現を持つ．ドメインに関連するドメイン属性は，<非派生>表現もしくは<派生>表現を持つ．機能と機能属性は，各々<派生>表現を持つ．ドメインの構造は，ドメインのドメイン属性によって定義する．ドメインの振舞いは，機能呼出しおよび機能呼出に関連する機能の組み合わせによって定義される．ここで全ての<派生>表現は，1 つ以上の派生を持つことを意味する．

3.1.1 ドメインインスタンス

ドメインの多品種への対応は，ドメインとドメイン属性に関する複数の派生で表すことになる．ここで，ドメインとドメイン属性の派生により，選択肢のない一意のドメインとドメイン属性の組合せを，ドメインインスタンスと呼ぶ．ドメインインスタンスはドメインインスタンス ID を保有し，実行時にはドメインインスタンス ID を用いてドメインロジック全体の実行が可能となる．これにより，実行時の多品種の種別計算を不要とする．

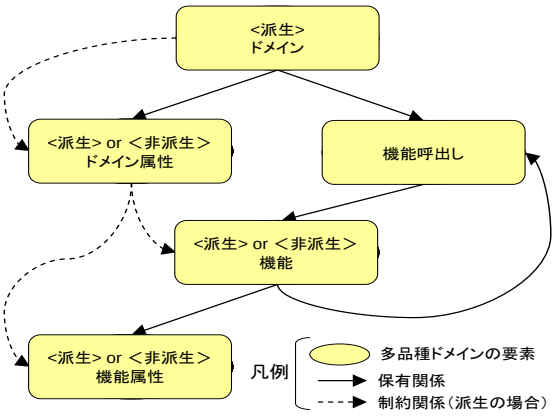

図 1．派生付ドメイン特性を示すメタモデル

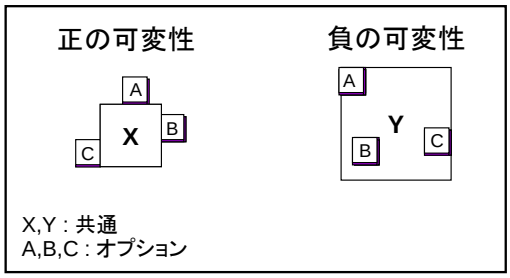

図 2．正の可変性と負の可変性

3.1.2 正の可変性と負の可変性との関連

ドメインの多品種対応に関する共通要素と派生要素の捉え方とし て，正の可変性と負の可変性がある[7]．これを図 2 に示す．

正の可変性は，共通 X を定めた後に，オプション A.B,C を加えていく捉え方である．負の可変性は，共通 Y を定めた後に，Y の範囲よりオプション A,B,C を定める事で共通部分を減らしていく捉え方である．

ドメインに関しては，最初の設計段階では負の可変性方式を採用する．これは設計初期段階で全ての可変性とその派生要素を設計する必要があることを意味する．よって機能と属性の多品種全体を設計し，ドメインインスタンスとして取り得る設計範囲を絞って行く進め方となる．また，設計変更時に関する負の可変性に関して，新規開発要素がないために，開発コストが少ない性質がある．一方，設計変更時の正の可変性に関しては，追加開発を必要とする設計要素を追加する必要があるため，開発コストが大きくなりやすい．

3.2 UML 表現拡張

我々は，派生付ドメイン特性メタモデルに関して設計及びモデル変換のために UML 拡張を行った．UML Profile を用いたステレオタイプ対応について図 3 に示す．

- ***Domain***

Domain は，DI コンテナのための beanClass と symbolName のステレオタイププロパティを持つクラス拡張である．Domain クラスは，機能呼出しをメソッドとして保有し，派生を伴うドメイン属性に関しては VariableProperty クラスとコンポジット関係を持つ．

- ***Function***

Function は，機能要素に対応するクラス拡張である．DI コンテナのための beanClass と symbolName をステレオタイププロパティに持つ．派生持つクラス拡張である．Function クラスは，振舞い呼出しをメソッドとして保有する．派生を伴う機能属性に関しては VariableProperty クラスとコンポジット関係を持つ．

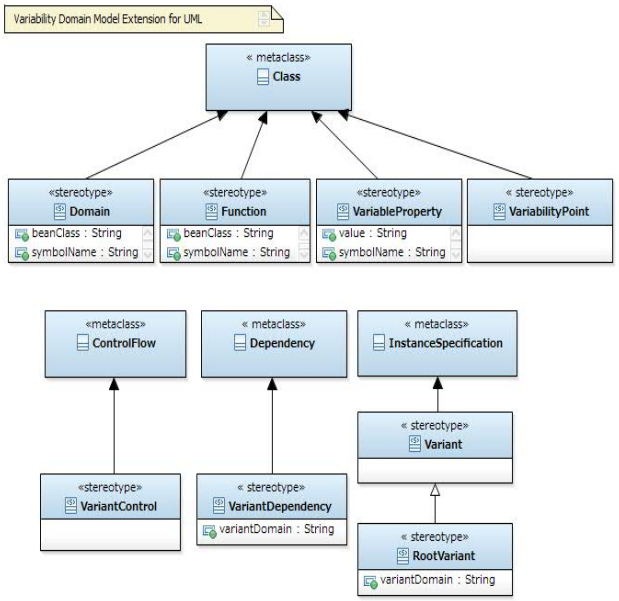

図 3. 派生付ドメイン特性用ステレオタイプ

- ***VariabilityProperty***

VariabilityPropery は，ドメイン属性と機能属性に対応するクラス拡張である． DI コンテナのための value と symbolName をステレオタイププロパティに持つ．

- ***VariabilityPoint***

VariabilityPoint は，派生を持つ可変要素に対応するクラス拡張である．VariabilityPoint に対するクラスは，Variant 要素を 1 つ以上保有しなければならない．

- ***Variant***

Variant は，派生に対応したオブジェクト拡張として InstanceSpecification の拡張で表される． ドメイン，ドメイン属性，機能属性の派生は全て Variant で表現される．

- ***RootVariant***

RootVariant は，Variant を継承し，Domain クラスの派生として表される．

- ***VariantDependency***

VariantDependency は，Variant 間の依存制約関係のための Dependency 拡張である．基本的な依存制約としては Variant が選択されるときに同時選択される Variant として VariantDependency の関係を定義する．依存制約の組合せについては，3 章 3 節で述べる．

- ***VariantControl***

VariantControl は，派生制約に依存する機能呼出し間の呼出しを表すための ControlFlow 拡張である．VariantControl の機能呼出し際には，派生間の依存制約を解決した上で呼出の可否を決定する．

3.3 変換分析

3.3.1 ドメインインスタンスの抽出分析

ドメインインスタンス抽出のための派生制約パターンについて，図 4 に示す．派生制約パターンを説明するためのドメインアーキテクチャのサンプルとして，ドメイン D とドメイン属性 P1 と P2 の関連定義を図 4 (a)に示す．図 4 (a)のドメインアーキテクチャの例に対する派生の例を図 4 (b)に示す．また，派生制約パターンと抽出されるドメインインスタンスについて図 4 (c)で示す．

ドメインインスタンスは，オブジェクト図上でドメインに対して依存関係を持つ派生を対象とし，派生制約パターンの解決によって抽出される．抽出の際には，複数のドメインインスタンスには ID が割り振られる．

OR の派生制約パターンは，同一の VariabilityPoint に対する複数の Variant の中から，Variant が択一選択されることを意味する．OR の例では，P1 のドメイン属性は派生 p11 と派生 p12 を持ち，この 2 つの派生に対するドメインインスタンスは，択一選択として 2 つ抽出される．

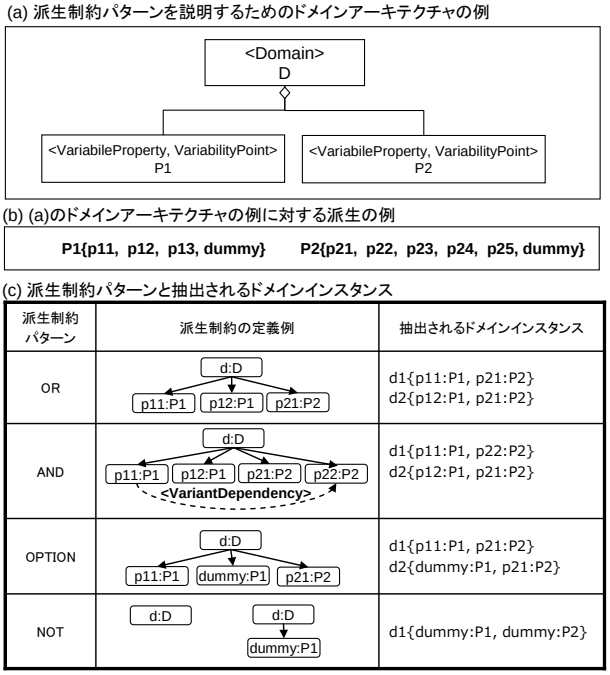

図 4. ドメインインスタンス抽出のための派生制約パターン

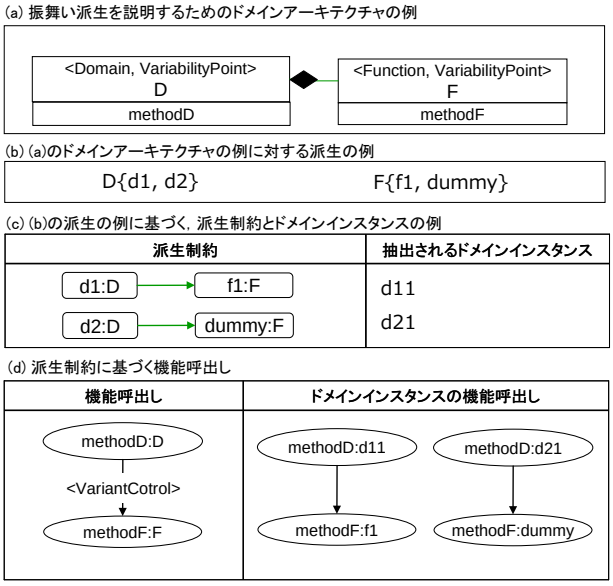

図 5. 派生制約に基づく振舞い派生

AND の派生制約パターンは，異なる VariabilityPoint からの派生する Variant の間を VariantDependency によって関係付ける．AND の例では，派生 p11 の選択の状態にあるときに，派生 p22 が選択されることを意味する．

OPTION の派生制約パターンは， 実体を伴わない dummy を持つ Variant との間の OR の派生制約パターンによって表される．OPTION の例では，派生 p1 が選択される場合とそうでない場合があるということを示す．ドメインインスタンスは，択一表現として 2 つ抽出されることになる．

NOT の派生制約パターンは，VariabilityPoint からの dummy の Variant が 1 つだけ関係が付いているか，関係が存在していないことで表される．

3.3.2 振舞いの派生分析

ドメイン振舞い定義では，アクティビティ図を用いて，機能呼出しとなる function のメソッド呼出アクション間を UML のコントロールフローによって繋げる．派生を持つ呼出しは，VariantControl フローで表現される．呼出される振舞いの派生分析が実施されて決定される．派生制約に基づく振舞い派生について図 5 に示す．

派生制約パターンを説明するためのドメインアーキテクチャのサンプルとして，ドメイン D と機能 F の関連定義を図 5 (a)に示す．図 5 (a)のドメインアーキテクチャの例に対する派生の例を図 5(b)に示す．また，派生制約パターンと抽出されるドメインインスタンスについて図 5 (c)で示している．図 5 (d)では，派生制約に基づく機能呼出しとして，ドメイン D の機能呼出し methodD から機能 F の機能呼出しである methodF へフローが定義されている．ここで，機能 F はドメイン D の派生 d1 からのみ呼びだされる派生制約が定義されているため，ドメインインスタンスの機能呼出しは，派生 d1 のインスタンスである d11 から呼出されることになる．この振舞いの派生分析は DI コンテナの定義ファイルへと変換される．

3.4 モデル駆動開発

派生付ドメイン特性に対するモデル駆動型開発で実現する開発アーキテクチャとして，図 6 に示す派生対応ドメイン駆動アーキテクチャを考案した．ここで，派生付きドメイン特性を表すモデルを派生対応ドメインモデルと呼ぶ．派生対応ドメインモデルは UML で表現され，下記 4 つのモデル的側面を持つ．

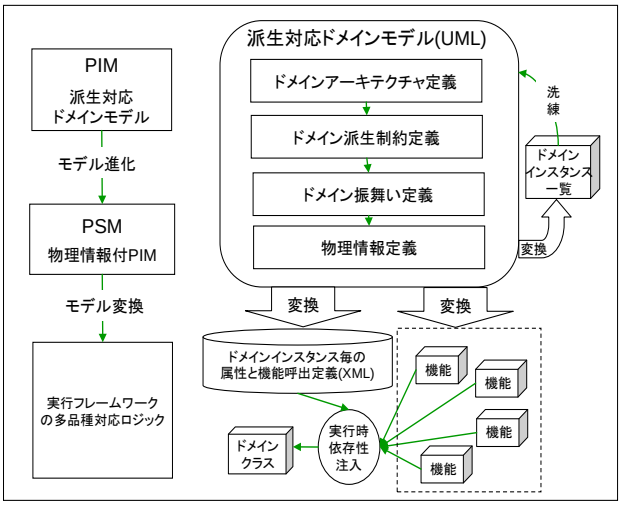

図 6．派生対応ドメイン駆動アーキテクチャ

(1)ドメインアーキテクチャ定義

ドメインアーキテクチャ定義は，派生付ドメイン特性に沿った多品種を含むドメインの全体構造を示す．ドメインとドメイン属性，機能と機能属性は UML クラスで表現され，各々の関連が UML クラス図にて定義される．機能呼出しはドメインが保有する振舞いメソッドとして定義される．派生要素に関しては，UML クラスに対する UML オブジェクトを拡張した要素として複数表現を可能とする．

(2) 派生制約定義

派生制約定義は，派生付ドメイン特性を持つモデルにおける派生要素間の制約を示す．制約については，派生間の依存性を表現することが可能であり，AND, OR の組合せ制約も対応する．これらの制約に基づきドメインインスタンス一覧が生成されるが，ドメインインスタンス一覧を参照しながら制約表現の妥当性を要件視点で確認することが出来る．

(3) ドメイン振舞い定義

ドメイン振舞い定義は，機能呼出しに関する振舞いとしてアクティビティ図とシーケンス図によって示される．アクティビティ図では，ドメインの機能呼出しの構造的側面を示す．機能呼出しの順序性および内部処理の含む詳細な振舞いは，設計者によってアクティビティ図を参考にしながらシーケンス図によって示される．また，振舞い定義に基づいて，ドメインアーキテクチャの洗練も必要に応じて実施する．

(4) 物理情報定義

物理情報定義は，派生対応ドメインモデルに対して実行フレームワークに変換が可能な物理情報が示される．モデル要素の英字名付与や，パッケージやクラス名が該当する．

3.4.1 モデル変換

図 6 を用いてモデル変換について述べる．モデル変換を特徴とする MDA[7]の考え方への対応比較としては，3 章 4 節の(1),(2),(3)が PIM に相当し，PSM が(4)に相当する．MDA との違いは，PIM から PSM へはモデル変換を行わずに，同一のモデルに物理情報を付加することである．これを，モデル進化と呼ぶ．モデル進化は，ドメイン駆動設計の考え方に沿って，PSM まで同一のモデルを育てていくことに意義を持つ．

派生対応ドメインモデルからのモデル変換について次の 3 種類がある．

- ドメインインスタンス一覧への変換

ドメインアーキテクチャ定義と派生制約定義によって，派生制約に基づく抽出分析を行い，ドメインインスタンスを生成後にドメインインスタンス一覧への変換する．

これにより，多品種開発の実行時インスタンスを実行前に確認することが出来る．

- ドメインインスタンス毎の属性と機能呼出定義へのモデル変換

物理情報定義の後は，派生制約定義を入力としたモデル変更機能によって，ドメインインスタンス毎の属性と機能呼出定義ファイルに変換される．DI コンテナは，ドメインインスタンス ID をキーとしてこのファイルを参照し依存性注入することで，アプリケーション実行時に必要なドメインのロジックを実行することが可能となる．

- Java クラス変換

ドメインモデル要素である依存性注入用のクラスは，そのまま Java クラスへと変換される．

3.4.2 派生対応ドメインモデルの位置付け

派生対応ドメインモデルは，アプリケーション全体を示すものではない．ドメインが持つ機能呼出しに対して，上位のアプリケーション側から呼び出される．派生対応ドメインモデルは，派生付ドメイン特性の要件を予め抽出しモデルの成熟度をあげておく必要がある．このように関心毎の分離として，派生対応ドメインモデルとアプリケーションの関係を疎結合に保つことは，開発の柔軟性を高めることに繋がる．

4. 金融商品ドメインでの実現例

本研究の実現例として，抜粋した金融商品モデルを用いた．

4.1 モデルとの対応

派生付ドメイン特性を持つモデル要素との対応について表 1 に示す．

表 1 モデル要素との対応

モデル要素	設計対象	派生要素
ドメイン	金融商品	商品 A, 商品 B,
ドメイン属性	委託者属性	個人，法人
	契約期間	1 年，2 年，5 年
機能呼出し	最高額を判定する	-
	最低額を判定する	-
機能	最高預入額判定条件	F11, dummy
	最低預入額判定条件	F12, dummy

4.2 設計した派生対応ドメインモデル

派生付きドメイン特性のための UML Profile を用い，金融商品のモデルを作成した．金融商品のドメインアーキテクチャを図 7 に示す．金融商品ドメインは，委託者属性と契約期間のドメイン属性を保有し，最高預入額判定条件と最低預入額判定条件を保有していることが表されている．図 8 に示すドメイン派生制約では，委託者属性の派生である”法人”と最低預入判定条件の機能が AND の派生制約を持つ．商品の振舞いアクティビティ図を図 9 に示す．”最高額を判定する”と”最低額を判定する”のメソッドのアクティビティは，<VariableControl>ステレオタイプのフローで呼出される．振舞いシーケンス図について図 10 に示す．振舞いシーケンス図は，モデルの振舞いと構造の洗練後の最後に定義した．

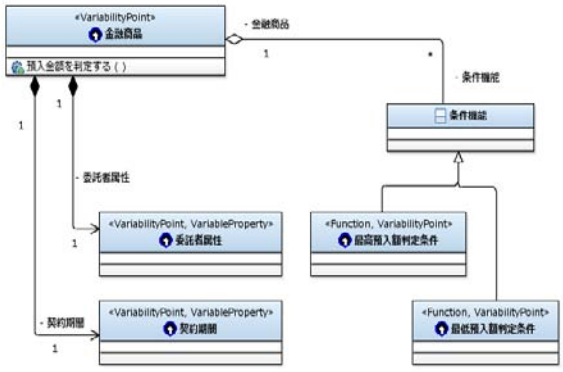

図 7 金融商品のドメインアーキテクチャ

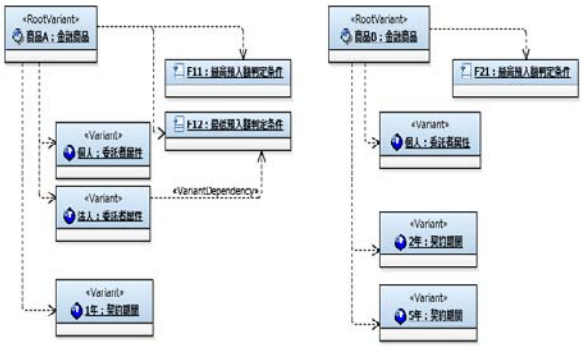

図 8 金融商品のドメイン派生制約（一部）

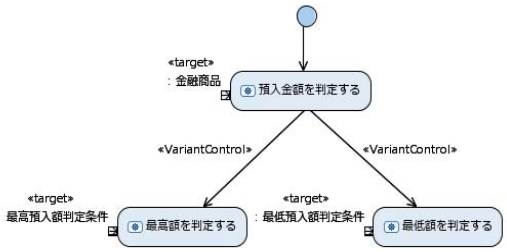

図 9.金融商品の振舞いアクティビティ図

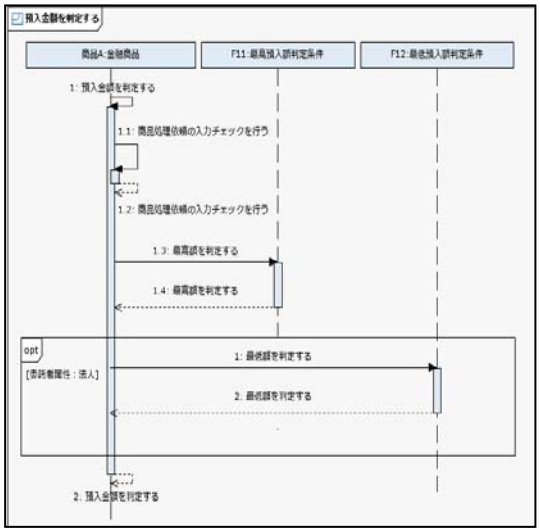

図 10.金融商品の振舞いシーケンス図

			ドメインインスタンスID			
モデル要素種別	要素の値	派生	商品A0001	商品A0002	商品B0001	商品B0002
ドメイン属性	委託者属性	個人	●		●	●
ドメイン属性	委託者属性	法人		●		
ドメイン属性	契約期間	1年	●			
ドメイン属性	契約期間	2年			●	
ドメイン属性	契約期間	5年				●
機能	最高預入額判定	F11	●		●	●
機能	最高預入額判定	dummy				
機能	最低預入額判定	F12		●		
機能	最低預入額判定	dummy			●	●

図 11.モデル変換後のドメインインスタンス一覧

```
<?xml version="1.0" encoding="UTF-8"?>
<p:beans xmlns:p="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.springframework.org/schema/beans ../../BankingProduct
UMLProfile/spring/spring-beans-3.1.xsd "></p:beans>
<beans>
<!-- ドメイン 商品A002 -->
  <bean id="ProductA001"   class="com.ibm.dde.sample.domain.ProductA">
    <property name="requesterTypeProperty">
          <value>Person</value> <!-- 個人 -->
    </property>
    <property name="termTypeProperty">
          <value>12</value> <!-- 1year by month -->
    </property>
    <property name="maxDepositValueJudgeConditionFunction"
          ref="F11_MaxDepositValueJudgeConditionFunction" />
    <property name="minDepositValueJudgeConditionFunction"
          ref="Dummy_DepositValueJudgeConditionFunction"/>
  </bean>
<!-- 機能 最高預入額判定条件 -->
  <bean id="F11_MaxDepositValueJudgeConditionFunction"
        class="com.ibm.dde.sample.function.MaxDepositValueJudgeConditionFunction">
  </bean>
 <!-- 機能 最高預入額判定条件 呼出なし-->
  <bean id="Dummy_MaxDepositValueJudgeConditionFunction"
        class="com.ibm.dde.sample.function.DummyDepositValueJudgeConditionFunction"
>
  </bean>
  (以下省略)
```

図 12.モデル変換後の Spring 定義ファイル

4.3 モデル変換

モデル変換のツールを開発し，モデル変換を実行した．生成されたドメインモデルインスタンス一覧について，図 11 で示す．ドメインインスタンスが抽出され，各々選択された派生に●印が付けられている．また，今回は Spring を対象にモデル変換機能を開発した．これによって生成された Spring の定義ファイルについて図 12 に示す．モデル変換される依存性注入に関わるクラスが呼出される．クラスは Spring フレームワークに従ってインタフェースクラスとして生成される．シーケンス図に沿った振舞いを実現するためには開発者が実装を行うことでドメインのロジック実装を完成させることが出来る．

5. あとがき

本稿では，DI コンテナに対応可能とする派生付ドメイン特性を述べ，UMLProfile を用いたステレオタイプ拡張を定義した．またドメインモデルによる派生制約の分析について提案した．本研究の特徴としては，ドメインアーキテクチャと派生の関係に対して，UML クラスとオブジェクトの関係を用いた点にある．プロダクトライン技術で扱われるフィーチャモデルを表現し，金融商品を実施例として Spring フレームワークを対象としたモデル変換を実現した．

負の可変性として，既存のドメインアーキテクチャを用いて派生制約関係のみを変更する場合には，プログラム開発が不要となり，Spring 定義ファイルの変換だけで対応可能である．正の可変性としては，派生付ドメイン特性に従って，機能が追加される場合はその開発だけで最小限で済むことが可能である．

また，開発アーキテクチャとして新規の依存性注入に関する実行フレームワークに対応する場合でも，PIM レベルのモデルは再利用可能となり，開発柔軟性も向上できる．今後は，機能の再利用性の視点からの開発柔軟性についても研究を進めて行きたい．

謝辞

本研究にあたって支援を頂いた IBM グローバルビジネスサービスの金融チームに感謝する．

参考文献

[1] Martin Fowler, ”Inversion of Control Containers and the Dependency Injection pattern”
http://www.kakutani.com/trans/fowler/injection.html, 2004
[2] ILog によるビジネスルール管理
http://www-06.ibm.com/software/jp/websphere/ilog/business-rule-management/
[3] 野田夏子. "2 プロダクトラインの可変性管理: 可変性のモデル化とアーキテクチャ設計 (< 特集> ソフトウェア再利用の新しい波-広がりを見せるプロダクトライン型ソフトウェア開発-)." 情報処理 50.4 (2009): 274-279.
[4] Evans, E. Domain-Driven Design, Tackling Complexity in the Heart of Software, Addison-Wesley (2003)
[5] UML – Unified Model Language
http://www.uml.org/
[6] MDA - Model Driven Architecture
http://www.omg.org/mda
[7] Pohl, Klaus, Gunter Bockle, and Frank Van Der Linden. , Software product line engineering: foundations, principles, and techniques. Springer, 2005.
[8] Product Line Implementation using Aspect-Oriented and Model-Driven Software Development M. Voelter and I. Groher, SPLC 2007, pp. 233-242, 2007
http://domaindrivendesign.org/
[9] K. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson. : Feature-Oriented Domain Analysis (FODA) Feasibility Study, Technical Report, Software Engineering Institute, Carnegie Mellon University, CMU/SEI-90-TR-222 (Nov. 1990).
[10] Spring Framework
http://projects.spring.io/spring-framework/
[11] Domain-Driven Design :
http://domaindrivendesign.org/
[12]Leorux Daniel, Martin Nally, and Kenneth Hussey. "Rational Software Architect: A tool for domain-specific modeling." IBM systems journal 45.3 (2006): 555-568.