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

野田 夏子　NEC 共通基盤ソフトウェア研究所

プロダクトライン開発では，可変性，つまりプロダクトラインに含まれるプロダクトの機能や品質特性等のバリエーションを，明確に捉え，そのバリエーションがソフトウェアとしてどのように実現されるかの対応を管理することが非常に重要である．このようなプロダクトラインの可変性管理のための技術について，中心となる可変性のモデル化とそれを踏まえたアーキテクチャ設計について解説する．

可変性とは

プロダクトラインにおける可変性（variability，バライアビリティ）とは，直感的に言えば，プロダクトラインに含まれるプロダクトごとに変化し得る（=variable）事柄，バリエーションである．これに対して，すべてのプロダクトに共通する事柄は共通性（commonality，コモナリティ）と呼ばれる．たとえば携帯電話を考えてみよう．通話機能，これは全機種が共通して持つ機能，つまり共通性である．一方，機種によっては，国内通話専用か，海外でも使用可能かといった違い，つまり可変性が生まれる．また，カメラ機能の有無，電子メール機能の有無等，さまざまな可変性が考えられるだろう．このような共通性と可変性を明確に捉えモデル化し，可変性を設計や実装上で実現し，またその対応関係を管理するといった一連の活動を可変性管理と呼ぶ．これはプロダクトライン開発において特徴的な活動であり，かつ開発の成否にかかわる重要なキーの１つと言える．

ところで，プロダクトラインにおける可変性をもう少し正確に定義しようとすると，少しずつ異なる複数の定義があるようである☆1．１つは，「プロダクトラインを構成するメンバ（プロダクト）がお互いにどのように異なっているかについての想定」[5]とするものである．これは，冒頭に述べた直感的な説明に非常に近いものであるが，違いに関しての「想定」が可変性であるとするものである．このような立場に立つ場合，違いとして要求，特にフィーチャ（feature）の違いを捉えることが多いようだ．定義としてではないが，「可変性をフィーチャで表現するのは自然であり直感的である」[2]とするものもあり，実際多くの研究・事例において可変性を後述のフィーチャモデルで記述している．ここでフィーチャとは，ユーザが認識するはっきりした特徴のことであり，要求管理の単位として用いられる．機能的なものも非機能的なものも含む．

これに対して，ユーザが認識できるものだけでなく，たとえば典型的には設計や実装上の違いのような，ユーザに見えないものも可変性とする立場もある．Pohl らは，実世界において可変であるものまたはその性質（可変性サブジェクト）とその実現例（可変性オブジェクト）のうち，特定のソフトウェアプロダクトラインの実現のために埋め込まれたものが，ソフトウェアプロダクトラインの可変性であるとする[4]．実世界における可変なものをソフトウェアとして実現するために埋め込まれたものであるから，ユーザに見えるものもあれば見えないものもある．また，要求にとどまらず，設計，実装等すべての工程において差異を可変性と捉えることになる．

ここまでの定義においては，差異そのものもしくは差異の想定が可変性とされているが，このような差異を持ち得るシステムや資産または開発環境の「能力（ability）」を可変性と呼ぶものもある[1]．この場合，プロダクトライン中のバリエーションをどう実現するかの仕掛けに対して，より多くの関心が払われるようだ．

定義自体の詳細は本質ではないが，このような定義の違いがモデルの違いや手法の違いをもたらすこともあるので，注意を払うことは必要であろう．

可変性のモデル化

◉フィーチャモデル

可変性を捉え管理することが重要であると述べたが，

☆1 一般的な意味での variability をそのまま使っているという意識のためか，特にこの言葉の定義を与えていない文献も多いため，正確な定義を列挙することは難しいが，以下に説明するいずれかの意味でこの用語を使っているものが多いようである．

この可変性管理において重要なのが可変性のモデル化である．モデル化の手法がいくつか提案されているが，本稿では代表的なものを2つ紹介する．まずフィーチャモデルによるモデル化を紹介しよう．

フィーチャモデルは，もともとはより一般的な意味でのドメイン分析の一手法Feature-Oriented Domain Analysis（FODA）の中で，ドメインにおいて共通なフィーチャと可変なフィーチャを明示的にモデル化するための記法として提案された．そして，プロダクトラインにおける可変性はフィーチャの差異として捉えるのが自然との考え方から，プロダクトライン開発における可変性のモデル化に用いられるようになった．**図-1**はフィーチャモデルの記述例である．フィーチャは木構造を作っており，親のフィーチャと子のフィーチャは，必須（子フィーチャが親フィーチャにとって必ず必要），選択(子フィーチャが親フィーチャにとって選択可能，子は必ずしも存在しなくてもよい)，代替(子フィーチャ群のいずれか1つが親フィーチャにとって必要）のいずれかの関係でつながれる．またフィーチャ間には依存関係を定義することができ，依存元フィーチャが存在するなら，依存先フィーチャが必ず必要であることを示している（依存関係は，図上には示さず，ルールとしてテキスト表現を与えることもある）．ルートにあたるフィーチャはプロダクトラインを示すため，フィーチャモデル全体としてはプロダクトライン全体に対する共通性・可変性を表すことになる．図-1は，簡単な携帯電話の例を示しており，すべてのプロダクトは通話機能を持つが，撮影機能は可変的なフィーチャであることを示す．また，撮影機能を持つ場合，それは高画素か低画素かのどちらであり，高画素の場合，外部メモリが必要であることが示されている．

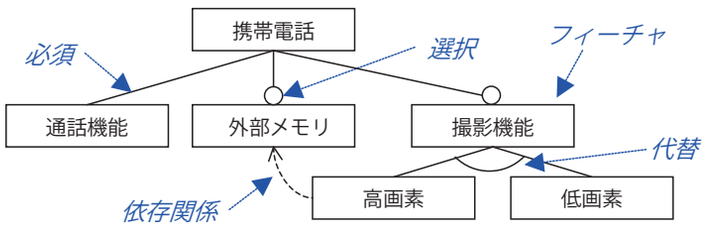

図-1　フィーチャモデルの例

フィーチャカテゴリ	内 容
能力 (Capability)	サービス（システムのサービス） 操作（システムの操作） 非機能特性（概観，コスト，制約，品質）
動作環境 (Operating environment)	SW/HWインタフェース（API，デバイスドライバ） SW/HWプラットフォーム（OS，CPU）
ドメイン技術 (Doman technology)	ドメイン特有の手法（勧告，ドメインでの理論，法規，標準）
実現技術 (Implementation technique)	設計判断（アーキテクチャスタイル，プロセス協調の方法） 実装判断（アルゴリズム，プロトコル，実装方法）

表-1　Kangらによるフィーチャカテゴリ

このように，フィーチャは必ずしも機能だけを表すものではなく，品質を表すものや，利用技術(実装技術)を表すものであってもよい．しかしこのような多様なフィーチャが混在すると，全体像が把握しにくくなる場合がある．Kangらは，フィーチャを書く際に，それをカテゴライズして記述することが有用だと指摘しており，そのためのフィーチャカテゴリを提案している[2]（**表-1**）．カテゴリごとに領域に分けて記述することで，効果的なフィーチャの整理・体系化ができる(**図-2**)．

図-1，2から分かるように，フィーチャモデルは非常にシンプルな図法であるため，さまざまなステークホルダが容易にプロダクトライン全体の共通性・可変性を俯瞰することができる．そのため，当初はコミュニケーションの手段として提案されたが，近年はそれを越えてさらに詳細なモデル化に用いられる例なども出てきている．

◉OVM

もう1つのモデル化手法が，Pohlらにより提案されたOrthogonal Variability Model（OVM）である．

Pohlらは，ユーザに見える可変性（外部的可変性と呼ぶ）だけではなく，それを徐々にリファインする過程で現れる，ユーザには見えない内部的なものも可変性（内部的可変性と呼ぶ）として明示的に捉えるとしている[4]．前者は顧客のニーズ，法律，標準などに関するものであり，後者は技術的な側面に関するものである．この2種類の可変性を模式的に示したものが**図-3**である．開発の当初は外部的可変性が支配的だが，開発の進展とともに内部的可変性が派生してくるイメージを示している．

このように，内部的なものも含め，要求段階だけでなく，設計段階，実装段階と開発を通じて現れるものを可変性として捉える場合，それぞれの段階で作成する成果物において可変性を表現する必要が生じる．そうした際に可変性がさまざまな成果物に分散すると分かりづらい，またさまざまな成果物（たとえばユースケース図やクラ

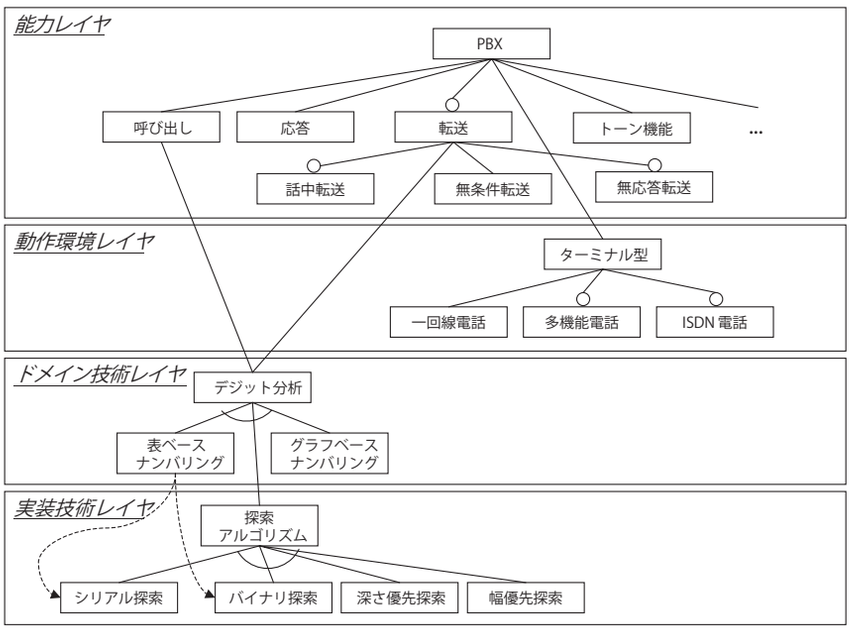

図 -2　PBX プロダクトラインのフィーチャモデル

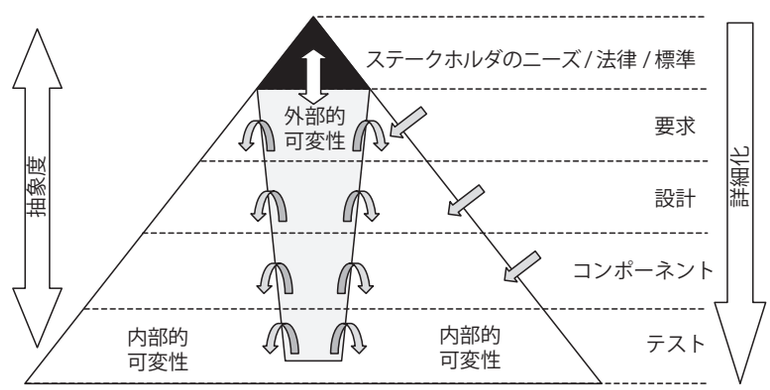

図 -3　外部的可変性と内部的可変性

ス図）に直接可変性を表現するための記法の拡張を行うのでは図が複雑になりすぎる場合もあるため，可変性を明示的に表現し他の成果物とは直行的に管理するモデルとして OVM を定義し，それと各成果物との対応をとることを提案している．

図 -4 に OVM を用いた可変性の記述の例を示す．OVM では，可変性図（図 -4 の右）で可変性のモデルを示す．OVM における基本的な構成要素は，バリエーションポイントとバリアントである．バリエーションポイントは可変性サブジェクトのモデルにおける表現，すなわち可変性概念を表すものであり，バリアントは可変性オブジェクトのモデルにおける表現，すなわちバリエーションポイントに対応づけられる選択肢である．バリエーションポイントとバリアントの関係には，必須（バリエーションポイントがアプリケーションの一部であればバリアントは必ず選ばれる），選択（バリアント群から 0 個以上複数個が対応づけられるかもしれない），代替（示された多重度の範囲でバリアントが対応づけられるかもしれない）がある．図 -4 の可変性図は，ドアロックのバリエーションポイントに対して，手動，電動のドアロックが対応づけられる可能性があり，ロック認証に対しては（ロック認証が）なし，キーパッド，指紋スキャナから少なくとも 1 つ，たかだか 2 つまでが対応づけられることを示す．OVM は，基本的にはバリエーションポイントとバリアントの対応関係を示すものであり，その記述はフラットである（木構造を作らない）．したがっ

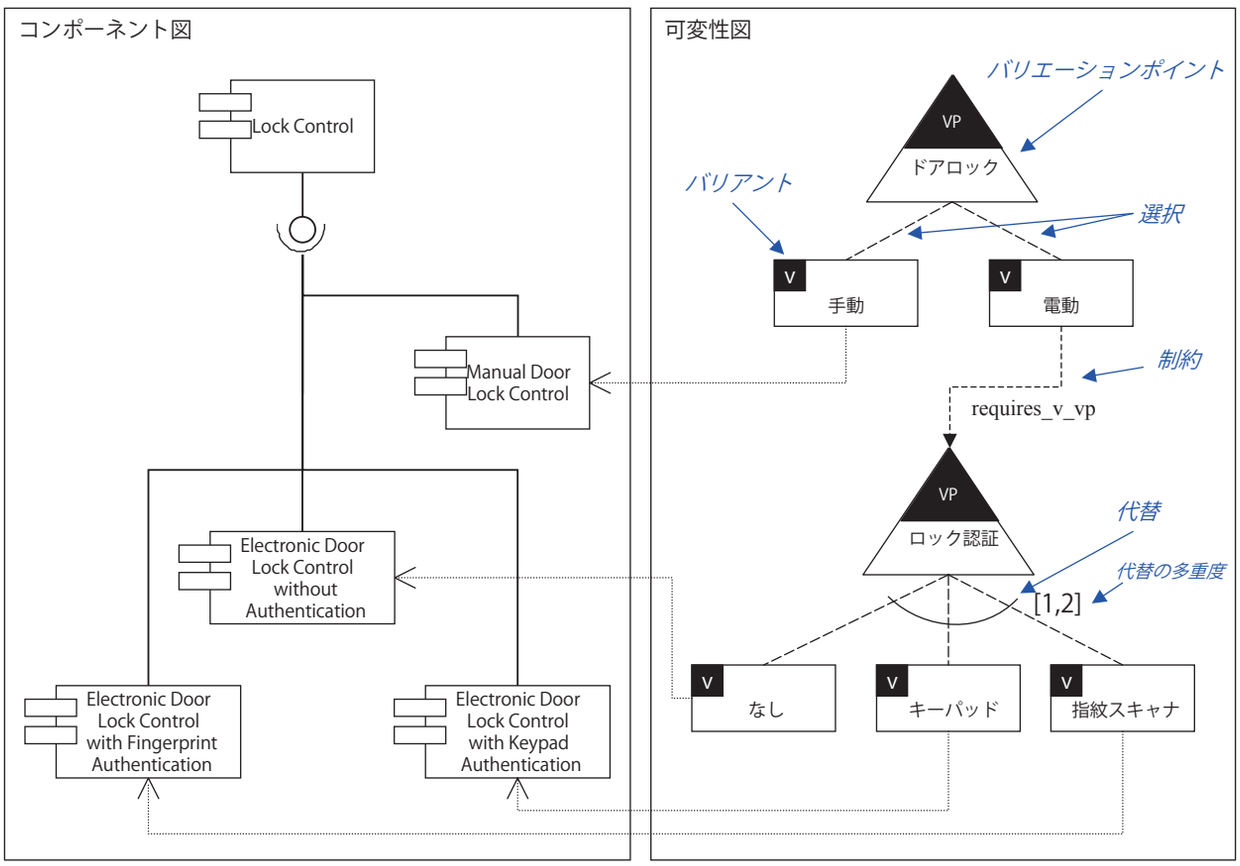

図 - 4　OVM によるモデル化例

て，可変性間の関係を示す階層化の概念は制約で表現する．制約はバリアント間（v_v），バリエーションポイント間（vp_vp），バリアント・バリエーションポイント間（v_vp）に定義でき，典型的には「必要とする（requires）」と「排除する（excludes）」の 2 種類を利用する．図 -4 では，ドアロックに電動が選ばれる場合には，ロック認証が必要とされることが示されている．

また，OVM では可変性図に示される可変性が他の成果物とどのように対応付くかを示す．図 -4 では，コンポーネント図と対応付けた例を示した．たとえば，手動のドアロックを選択するには Manual Door Lock Control コンポーネントが必要であり，キーパッドによるロック認証を選択するには Electronic Door Lock Control with Keypad Authentication コンポーネントが必要である等の対応関係が示されている．

フィーチャモデルは，特に要求段階において，プロダクトライン全体の共通性・可変性が捉えやすい一方，OVM では基本的にはフラットな表現であるので全体の俯瞰が場合によっては困難であるかもしれない．逆に，設計段階・実装段階における詳細な可変性を明示的に捉え，他の開発文書との対応付けを行う場合には，OVM が使いやすいかもしれない．それぞれの特徴を理解した上での使い分けが有効であろう．また，選択，代替等同じ用語が少しずつ違う意味で使われているものがあるので，注意が必要である．

プロダクトラインアーキテクチャの設計

プロダクトラインアーキテクチャは，プロダクトライン中のプロダクトが共有する（ソフトウェア）アーキテクチャである．ではソフトウェアアーキテクチャとは何か．狭義には，文字通りソフトウェアの構造ということであるが，具体的な構造だけではなく，開発の方針や方向付けに必要な構造化原則（抽象的な構造やガイドライン）を含めてソフトウェアアーキテクチャと呼ぶことが多い．なぜなら，このような構造化原則を含めた全体の構造を捉えることが，将来的な修正や拡張にも耐え得る適切なソフトウェアを開発するために重要だからである．またこの構造は，単一のモデルにより記述されるのではなく，考慮すべき品質特性のそれぞれに応じた複数の視点（ビ

実現手法	概要
継承	オブジェクト指向におけるクラスやインタフェースの継承.
拡張	異なり得る機能を固定的な機能から取り除き拡張可能に. 典型例はstrategyパターン.
構成	本体と分離されたリソースを利用. 定義ファイルなど.
パラメータ	複数のモジュールを内包させ，パラメータで選択. テキストプリプロセッサも含む.
テンプレート	特定の型に応じてコンポーネントの定義をインスタンス化.
生成	仕様や設計記述からコードを生成.
コンポーネント代替	開発時に複数の代替モジュールから選択して挿入.
プラグイン	実行時にコンポーネントを選択して挿入.
アスペクト	アスペクト指向プログラミング利用. 可変性をアスペクト化.
実行時条件	設定や定義ファイルによる実行時の切り替え.

表 -2　バリエーションポイントの実現手法

ュー)から捉えたモデル群により記述される．たとえば，性能を考慮する際には，タスク構造等の実行ビューからのモデル化が，拡張性を考慮する際にはクラス構造やファイル構造等の開発ビューからのモデル化が必要であるといった具合である．

このような観点で捉えると，プロダクトラインアーキテクチャとは，プロダクトライン中のすべてのプロダクトが共有する構造を考慮すべき品質特性に応じて複数のビューから捉えたものであり，プロダクトごとの差異，すなわち可変性を，構造中のどこでどのように実現するかについての決定や指針を含むものである．このようなプロダクトラインアーキテクチャを設計するための基本的な枠組みは，一般的なソフトウェアアーキテクチャの設計と同様である．すなわち，要求を明確にし，その中でアーキテクチャ的な重要性を踏まえながら，機能要求に基づき論理構造を，非機能要求に基づき対応するビューの構造を検討する．

ただし，プロダクトラインアーキテクチャ設計に特徴的な点も存在する．それは，複数のプロダクトが共有するアーキテクチャであるため，要求事項や将来的な修正や拡張に対する想定がより複雑になる点や，可変性に対する考慮が必要な点などである．特に，可変性をアーキテクチャ上のどこでどのように実現するか検討することは，プロダクトラインアーキテクチャ設計において最も重要なポイントの１つであろう．先に述べた可変性のモデル化を用い，特に要求事項の可変性を明示化し，それを踏まえてプロダクトラインアーキテクチャの設計を行う．すべてのプロダクトに共通する要求は，プロダクトラインアーキテクチャの固定的な部分として作り込むことができるが，プロダクトごとに変わり得る要求は，アーキテクチャ上では固定的に作り込まず，必要に応じて取捨選択できるように工夫しなければならない．こうした可変的なフィーチャに対応して，設計上で変更可能な部分をバリエーションポイント，バリエーションポイントに対応したフィーチャの実現をバリアントと呼ぶ．たとえば，あるコンポーネントをはめ込むと特定のフィーチャが実現できる場合，はめ込む個所がバリエーションポイント，はめ込まれるコンポーネントがバリアントとなる．１つの可変的なフィーチャに対して，設計上では複数のバリエーションポイントが作られることもあるし，１つのバリエーションポイントが複数のフィーチャの実現にかかわる場合もある（なお，バリエーションポイントは，コア資産において可変である部分，バリアントはその実現を意味する用語であり，アーキテクチャ設計に限ったものではない．先に述べた OVM でのバリエーションポイント，バリアントとほぼ同義である．ただし，これらが最も強く意識されなければならない工程の１つがアーキテクチャ設計であり，その際にはフィーチャと対応付けた説明がなされる）．

バリエーションポイントの具体的な実現には，既存のさまざまな設計技法が用いられる．**表 -2** に実現方法の例を示す．

こうした実現方法の違いは，結合タイミングと呼ばれる，プロダクトの違いが決定される時点の違いを生む．たとえば継承を用いてプロダクトごとの違いを作り込むとすると，それはプログラミング時点でプロダクトの違いを決定することになる．一方，パラメータでプロダクトごとの違いを作り込むとすると，それは実行時点でプロダクトの違いを決定することになる．この結合タイミングの違いは，開発の方法，配布の方法，あるいは販売形態などにもかかわってくる．たとえば，継承かパラメータかという選択によって，プロダクトごとに実行形式

ファイルが作られるのか，すべてのプロダクトが１つの実行形式に収まるのかといった違いが生じるからである．したがって，結合タイミングの検討は，要求定義の段階から複数のステークホルダを交えて検討を行うことが重要であり，それを踏まえて適切な可変性の実現方法をアーキテクチャ設計において検討することになる．

可変性にかかわるホットな話題

可変性管理は，プロダクトライン開発の成否にかかわるキーの１つであるため，常にさまざまな研究が行われてきたし，また現在も活発に続けられている．可変性のモデル化を中心テーマに可変性管理について議論する国際ワークショップが 2007 年から毎年行われているほどである☆2．本稿では，モデル化手法（記法）としてフィーチャモデルと OVM を紹介したが，多くの研究，開発事例において基本的な記法としてはフィーチャモデルが用いられているようである．その上で，より詳細かつ厳密な記述に用いるための記法の拡張や，形式的な定義の研究が行われている．また，フィーチャモデルによる記述に矛盾がないか形式手法により検証するといった研究も行われている．このようなアプローチにより，記述の正確性を高めプロダクトラインの信頼性を上げようとすることは，重要な試みであろう．一方で，フィーチャモデルが当初の意図を超えた広い範囲で使われるようになってきている現状において，フィーチャモデルによって記述しようとしている可変性がどのレベルのものであるのか（ユーザに見える特性であるのか，実現のための内部的な構造に現れるバリエーションであるのか）を考慮することなしに，闇雲に厳密さや詳細さを議論しても意味は薄いかもしれない．

可変性管理についてのもう１つのホットな話題は，アスペクト指向の有用性についてである．バリエーションポイントの実現方法の１つとして，アスペクト指向プログラミングの利用があることを紹介したが，その有用性は多くの研究で指摘されている．その一方で，アスペクト指向言語，たとえば AspectJ を単純に用いただけでは，フィーチャレベルの比較的な大きな単位の可変性が扱いにくい，フィーチャ間に依存関係がある場合に，フィーチャの組合せが変わると（フィーチャに対応する）アスペクトの変更が必要になり得るといった問題も指摘されている．このような問題に対しては，実装時のアスペクトの構造をフィーチャ分析を踏まえて行う実装手法の提案がある．さらに，プログラミングレベルだけで解決するのではなく，アーキテクチャ設計自体をアスペクト指向で行い，フィーチャをアーキテクチャ上のアスペクトとして分離することによって解決する手法の提案も行われている[3]．プロダクトライン開発へのアスペクト指向技術の適用の研究や実適用は今後も活発に行われると思われるが，このように多角的な，まさにさまざまなアスペクトからの検討が重要になるであろう．

また，実際のプロダクトラインにおける可変性について，その数は膨大なものになったり，また開発が進むにつれて増加したりして，扱いが困難になるといった問題が報告されている．このような場合，単純に可変性をモデル化すれば良いというものではない．こうした問題に対して，可変性自体を最適化するという手法の提案も行われている．プロダクトライン開発にかかわる研究・技術開発においては，このような現場の視点に根ざした工学的アプローチが，形式手法等の科学的アプローチとともに今後も非常に重要になるであろう．

以上，本稿ではプロダクトライン開発における可変性について，可変性管理の中心となるモデル化手法とそれを踏まえたアーキテクチャ設計について解説した．プロダクトライン開発は従来のアドホックな再利用とは異なり，対象の特性やソフトウェア構造の十分な理解に基づくコントロールされた再利用形態であり，可変性管理はその基盤となる技術である．今後実務と研究の両面から，より一層の進展を期待したい．

参考文献

1) Bachmann, F. and Clements, P. C. : Variability in Software Product Lines, CMU/SEI-2005-TR-012 (2005).
2) Lee, K., Kang, K. C. and Lee, J. : Concepts and Guidelines of Feature Modeling for Product Line Software Engineering, 7th International Conference on Software Reuse (ICSR-7) (2002).
3) Noda, N. and Kishi, T. : Aspect-oriented Modeling for Variability Management, 12th International Software Product Line Conference (SPLC 2008) (2008).
4) Pohl, K., Böckle, G. and Linden, F. v. d. : Software Product Line Engineering, Springer-Verlag Berlin Heidelberg (2005).
5) Weiss, D. M. and Lai, C. T. R. : Software Product-Line Engineering : A Family-Based Software Development Process. Addison-Wesley (1999).

（平成 21 年 2 月 18 日受付）

☆2 http://www.vamos-workshop.net/

野田 夏子（正会員）
n-noda@cw.jp.nec.com
東京女子大学大学院理学研究科修士課程修了，北陸先端科学技術大学院大学情報科学研究科博士課程修了，博士（情報科学）．日本電気（株）にてソフトウェア生産技術の研究に従事．ソフトウェア設計・検証，プロダクトライン開発に興味を持つ．