Vol.2013-SE-182 No.13
2013/10/24

ソフトウェアプロダクトラインにおける非機能特性を考慮した製品導出支援手法の提案

永野寛丸[†1]　　岸知二[†1]

ソフトウェアプロダクトライン（SPL）開発は，ソフトウェア資産の戦略的な再利用を促進するために近年注目を集めている開発アプローチである．SPL では，対象とする製品群の可変性を管理するために主にフィーチャモデルが用いられるが，規模が大きくなるにつれてフィーチャモデルから顧客の要求を満たした製品を導出することは必ずしも容易ではなくなってしまう．その理由として，フィーチャモデルは数百のフィーチャと数百万の製品バリエーションを持ち得るためである．この課題を解決するための研究として，セレクティビティ駆動製品導出（SDPD）[1]というアプローチが存在するが，このアプローチでは機能特性のみを扱っており非機能特性を考慮していない．本研究ではこの課題を解決するために，非機能特性を考慮できるように SDPD アプローチを拡張することにより，SPL における製品導出支援手法を提案することを目的とする．

A product derivation support method considering the non-functional properties in software product line

HIROMARU NAGANO[†1]　TOMOJI KISHI[†1]

Software Product Line (SPL) is one of promising approaches to support a systematic software reuse. In SPL, feature model is used to manage the variability among products in the SPL, however selecting required features from the feature model is not easy. The reason is feature model may have several hundred features and this makes millions of possible configurations. In order to support the feature selection, Selectivity Driven Product Derivation (SDPD) [1] approach is proposed, but they just focus on functional features, and non-functional properties are not considered. In this paper, we propose an extension of SDPD to be able to handle non-functional properties.

1. はじめに

ソフトウェアプロダクトライン（SPL: Software Product Line）開発は，ソフトウェア資産の再利用を支援し特定のニーズに対する製品のカスタマイズを容易にすることのできる開発アプローチであり，特に組込みソフトウェア開発分野で近年注目を集めている．SPL は 2 つの活動で構成されており，SPL の対象とする製品群（ドメイン）を定義するドメインエンジニアリングと，定義されたドメインから個別製品を導出するアプリケーションエンジニアリングである. しかしアプリケーションエンジニアリングにおいて，ドメインの規模が大きくなるにつれて顧客の要求を満たした製品を導出することは必ずしも容易ではなくなってしまう．なぜなら，製品導出はドメインエンジニアリングで作成されたフィーチャモデル等の可変性モデルから行うが，数百というフィーチャと数百万という製品バリエーションを持つフィーチャモデルの中から 1 つの製品を選択し導出しなければならないからである．

この課題を解決する研究として，セレクティビティ駆動製品導出（SDPD: Selectivity Driven Product Derivation）[1]というアプローチに注目する．SDPD アプローチはフィーチャモデルからの製品導出を最小の決定シーケンスで行うことを目的としており，フィーチャ毎の選択性としてセレクティビティという値を定義し，そのセレクティビティを利用した製品導出アルゴリズムを提案することでこの目的を達成している．

しかし，一般的に要求には機能要求と非機能要求が存在するが，この研究では機能要求のみを考慮しており，顧客の持つ非機能要求を満たした製品を導出することは難しい. 非機能要求とは，機能とは別に性能やメモリ使用量等のシステムの品質に関わる特性（非機能特性）に対する要求のことで，非機能特性はソフトウェア開発で非常に重要視される特性であり，その達成は機能要求同様に重要である[2]. 例えば組込みソフトウェア開発ではメモリ使用量の特性が重要視されるが，その理由として，デバイスが搭載したメモリ量以上のメモリを使うソフトウェアは動かないからである．そのため，SDPD アプローチによって製品が導出されたとしても，非機能特性を考慮していないため，その製品は実際には利用できない製品である恐れがある．

本研究ではこの SDPD アプローチの課題を解決するために，非機能特性を考慮できるように SDPD アプローチを拡張することによる，SPL における自動的に非機能特性を考慮できる製品導出支援手法を提案することを目的とする．

2. ソフトウェアプロダクトライン（SPL）

2.1 SPL の概要

ソフトウェアプロダクトライン（SPL）開発は同種同系

†1 早稲田大学
Waseda University

列のソフトウェア製品を効率的に生産するために，再利用可能なソフトウェア資産（コア資産）を整備し，これを用いて個別製品を作り出す開発アプローチである[3]．図 1 にその全体像を示す．

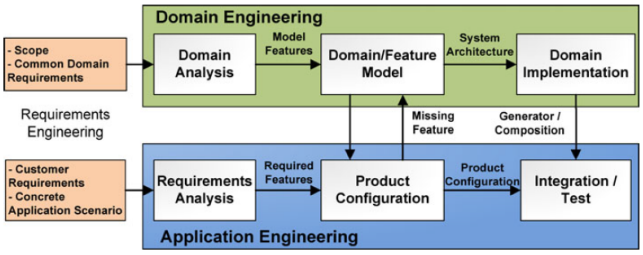


図 1　SPL 開発の全体像[4]

SPL 開発は大きく 2 つの活動から構成される．1 つ目はドメインエンジニアリングと呼ばれ，その SPL が対象とするドメインを定義し，製品バリエーション間の共通性と可変性を明らかにするために製品バリエーションをフィーチャモデル等の可変性モデルで表現する．2 つ目はアプリケーションエンジニアリングと呼ばれ，ドメインエンジニアリングで定義されたドメイン内，すなわちドメインエンジニアリングで作成された可変性モデルで定義された製品バリエーションの中から，顧客の要求に従って個別製品を導出するプロセスである．

アプリケーションエンジニアリングでの製品導出は基本的に可変性モデルから個別製品の構成を決定することである．この決定は，具体的なアプリケーションの要求をインプットとして行われる．個別製品の構成を決定してから，個々のフィーチャのコア資産を用いてシステムとして結合をする．本研究ではアプリケーションエンジニアリング，特に図 1 の Product Configuration にフォーカスする．

2.2 フィーチャモデル

本研究では，ドメインエンジニアリングで作成された可変性モデルとしてフィーチャモデルを利用する．

フィーチャモデルとは，K.Kang らが提案したフィーチャ指向ドメイン分析（FODA: Feature-Oriented Domain Analysis）[5]で定義されたモデルであり，製品バリエーション内で選択され得るフィーチャ（ユーザが観測可能な特徴）を階層的に表現したモデルである．フィーチャモデルの記法は様々存在するが，本研究では図 2 のように定義する．またフィーチャモデルの各要素の定義は以下である．

(1) **Mandatory**
フィーチャ a はフィーチャ b を必須とする．

(2) **Optional**
フィーチャ a はフィーチャ b を任意で選択できる．

(3) **Alternative**
フィーチャ a はフィーチャ b とフィーチャ c のどちらか 1 つを選択しなければならない．

(4) **OR**
フィーチャ a はフィーチャ b とフィーチャ c の内，1 つ以上を選択しなければならない．

(5) **requires**
フィーチャ a はフィーチャ b を必要とする．

(6) **excludes**
フィーチャ a とフィーチャ b は同時に選択されない．

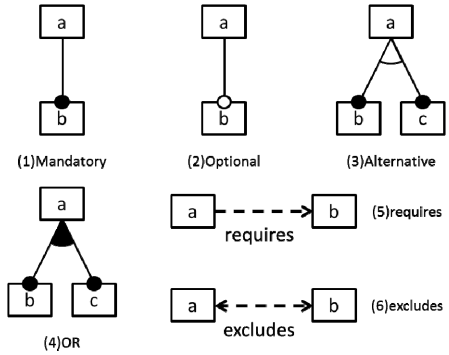


図 2　フィーチャモデルの記法

例として

図 3 に携帯電話 SPL のフィーチャモデルの簡単な例を示す．

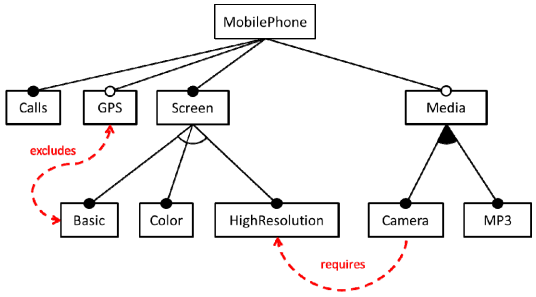


図 3　携帯電話 SPL のフィーチャモデル[1]

フィーチャ Calls は製品バリエーション全てに共通の Mandatory フィーチャであり，フィーチャ GPS は製品バリエーションの中の一部の製品が持ち得る Optional フィーチャである．また，フィーチャ Camera とフィーチャ MP3 の内 1 つ以上選択可能である OR 関係，フィーチャ Screen 以下の 3 つのフィーチャの内どれか 1 つのみ選択可能である Alternative 関係といった複数フィーチャ間の関係や，requires や excludes といったクロスツリー制約も表現することができる．

3. 製品導出における課題

本研究では，SPL のアプリケーションエンジニアリングでの製品導出における課題として，膨大なフィーチャを含むフィーチャモデル中で意思決定を行わなければならないという課題を解決することを目的とする．ここでいう意思決定はフィーチャモデル内でのフィーチャの選択のことを指す．産業レベルの SPL は典型的に数百のフィーチャを持ち得るが，その SPL から導出することのできる製品バリエーションは数百万に上ると言われている[1][4]．そのため，フィーチャ間の依存関係を正確に理解し，その中で真に必

要なフィーチャを持つ要求通りの製品を人手で決定することは時間がかかり，効率も悪い[6].

本研究では，この課題を解決するためのアプローチとして，S.Chen らが提案したセレクティビティ駆動製品導出（SDPD: Selectivity Driven Product Derivation）アプローチ[1]に注目する．

4. SDPD アプローチ[1]

4.1 SDPD アプローチの概要

S.Chen らは SPL のアプリケーションエンジニアリングにおいて最小の決定シーケンスで製品導出を行うことを目的としている．その目的を達成するために，彼らはフィーチャにセレクティビティと呼ばれる製品バリエーション全体に対するそのフィーチャを使用した製品の数の割合の値を付加し，最大のセレクティビティを持つフィーチャから意思決定を行う SDPD アプローチを提案している．

4.2 SDPD アルゴリズム

SDPD アプローチにおける製品導出アルゴリズムを図 4 に示す．

INPUT: Choice description(e, c) and intended product p
OUTPUT: A product derivation s
METHOD:
(1)$s \coloneqq []$
(2)Find most selective feature f of e
(3)If $f \notin p$ then $f \coloneqq \overline{f}$
(4)$s \coloneqq s \cdot f \cdot inclusion_f(e, c)$
(5)$e \coloneqq e|f$
(6)If $\varphi(p) \not\subseteq \varphi(s)$ then goto (2)
(7)Return s

図 4　SDPD アルゴリズム[1]

まず入力と出力について説明する．選択記述(e,c)はフィーチャモデルの構造を代数式で表現した記述である． e は選択式と呼ばれフィーチャモデルの木構造を表現し， c は制約と呼ばれフィーチャ間の制約を表現する．フィーチャモデルから選択記述への変換については後述する．意図された製品 p はユーザの機能要求とほぼ同義であり，最終的に導出される製品にユーザが要求しているフィーチャのセットである．SDPD アルゴリズムではこれに基づいて出力として導出される製品 s を決定する．

次に図 4 の METHOD 部分について説明する．まず(1)導出される製品 s を用意する．次に(2)セレクティビティを計算し（後述），選択式 e 中で最もセレクティビティの高いフィーチャ f を探す．(3) f が意図された製品 p に含まれない場合は $\overline{f}$ を f とする． $\overline{f}$ は除去されるフィーチャを表す．(4)導出される製品 s と f と，f によって選択記述(e,c)から自動選択（もしくは自動除去）されたフィーチャ（$inclusion_f(e,c)$）を s とする．それから(5)選択式 e に対して限定操作 $e|f$ を行う．限定操作 $e|f$ とは選択式 e から f 及び f によって自動選択（もしくは自動除去）されるフィーチャを選択（もしくは除去）する操作のことである．限定操作に関する定義については後述する．そして(6)もし $\varphi(p)$ が $\varphi(s)$ に含まれていなければ(2)に戻り，含まれていれば(7) s を返す．ここで関数 $\varphi(p)$ とは p 中に存在するフィーチャを示す関数である．

このアルゴリズムを見るとわかるようにセレクティビティは 1 つのフィーチャ（それに加え，そのフィーチャにより自動選択，自動除去されるフィーチャ）が選択（もしくは除去）されるたびに再計算される必要がある．

4.3 フィーチャモデルから選択記述への変換

SDPD アルゴリズムでは，入力としてフィーチャモデルの構造を代数式化した選択記述を用いる．そのため，まず与えられたフィーチャモデルを選択記述の形式に変換する必要がある．ここではフィーチャモデルから選択記述への変換方法について述べる．

図 5 はフィーチャモデルの各要素の選択記述への変換を示している．

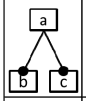
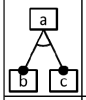
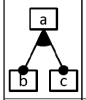
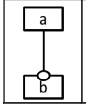
Feature Tree		Choice Description
	a requires both b and c	$(abc, \emptyset)$
	a requires either b or c, but not both	$(a(b+c), \emptyset)$
	a requires b or c or both	$(a(1+b)(1+c), \{b \vee c\})$
	b is an optional feature of a	$(a(1+b), \emptyset)$

図 5　フィーチャモデルから選択記述への変換[1]

選択記述中の“＋”は選択を示しており，例えば図 5 の Alternative の選択記述は$(a(b+c),\phi)$となっているが，$(b+c)$は選択なので b と c は同時に存在できず，この Alternative では ab と ac の 2 つの製品が導出され得る．

図 6 にフィーチャモデルから選択記述への変換例を示した．

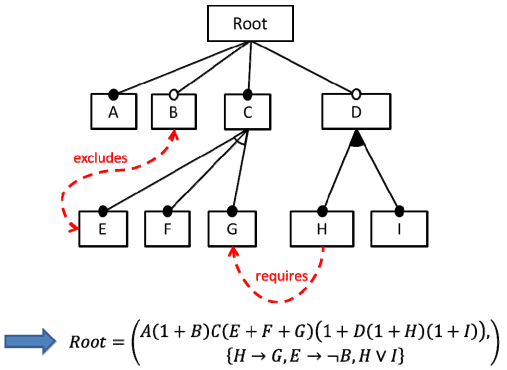


図 6　選択記述への変換例

基本的な要素の変換は図 5 で示したが，requires や excludes といったクロスツリー制約は $H \to G$， $E \to \neg B$ のように制約式で表現することができる．

4.4 限定操作

限定操作とは，制約や選択（もしくは除去）されたフィーチャを選択式に反映させる操作のことである．セレクティビティを計算する前に限定操作を行うことで，選択式に制約を反映させることができ，制約を考慮したセレクティビティを計算することができる．

限定操作を行うために，制約式をリテラル化する必要がある．リテラル化を行うための関数 $tru(c)$， $fls(c)$ は以下のように再帰的に定義させる．

$$tru(c) = \begin{cases} \{\{f\}\} & c = f \\ \{\{\bar{f}\}\} & c = \neg f \\ tru(c_1) \bowtie tru\,(c_2) & c = c_1 \wedge c_2 \\ tru(c_1) \cup fls(c_1) \bowtie tru(c_2) & c = c_1 \vee c_2 \\ tru(c_1) \bowtie tru(c_2) \cup fls(c_1) & c = c_1 \to c_2 \end{cases}$$

$$fls(c) = \begin{cases} \{\{\bar{f}\}\} & c = f \\ \{\{f\}\} & c = \neg f \\ fls(c_1) \cup tru(c_1) \bowtie fls\,(c_2) & c = c_1 \wedge c_2 \\ fls(c_1) \bowtie fls(c_2) & c = c_1 \vee c_2 \\ tru(c_1) \bowtie fls(c_2) & c = c_1 \to c_2 \end{cases}$$

また，

$$S \bowtie S' = \{s \cup s' | s \in S \wedge s' \in S'\}$$

である．例えば，制約 $a \to b \vee c \vee d$ をリテラル化するために $tru(a \to b \vee c \vee d)$ を考えると，上記の定義より，

$$tru(a \to b \vee c \vee d) = \{\{\bar{a}\}, \{a, b\}, \{a, \bar{b}, c\}, \{a, \bar{b}, \bar{c}, d\}\}$$

となり，制約が 4 つのリテラルに分解される．

以上のようにリテラル化された制約を選択式に反映する限定操作のための関数 $e \uparrow f$， $e \downarrow f$ は以下のように再帰的に定義される．

$$e \uparrow f = \begin{cases} e_1 \cdot \cdots \cdot (e_i \uparrow f) \cdot \cdots \cdot e_n & f \in \varphi(e_i) \wedge e = e_1 \cdot \cdots \cdot e_n \\ e_i \uparrow f & f \in \varphi(e_i) \wedge e = e_1 + \cdots + e_n \\ f & f = e \\ 1 & f \notin \varphi(e) \end{cases}$$

$$e \downarrow f = \begin{cases} e_1 \cdot \cdots \cdot (e_i \downarrow f) \cdot \cdots \cdot e_n & f \in \varphi(e_i) \wedge e = e_1 \cdot \cdots \cdot e_n \\ e_1 + \cdots + e_i \downarrow f + \cdots + e_n & f \in \varphi(e_i) \wedge e = e_1 + \cdots + e_n \\ 1 & f = e \\ e & otherwise \end{cases}$$

この 2 つの関数を使って定義された限定操作 $e|l$ は次のようになる．

$$e|l = \begin{cases} e \downarrow f & l = \bar{f} \\ e \uparrow l & otherwise \end{cases}$$

例えば，選択式 $(a+b)(c+d+e)$ に $a\bar{e}$ というリテラル化された制約を加えることを考える．その場合の限定操作は以下のようになる．

$$\begin{aligned} &(a+b)(c+d+e) \,|\, a\bar{e} \\ &= ((a+b)(c+d+e) \,|\, a) \,|\, \bar{e} \\ &= a(c+d+e) \,|\, \bar{e} \\ &= a(c+d) \end{aligned}$$

4.5 セレクティビティの計算

セレクティビティとは製品バリエーション全体に対する，対象のフィーチャを含む製品の数の割合のことである．

選択式 e 中のフィーチャ f のセレクティビティ $\sigma_f(e)$ は以下のように定義される．

$$\sigma_f(e) = \gamma_f(e) / \#(e)$$

ここで，$\gamma_f(e)$ は選択式 e 中のフィーチャ f の共通性であり，以下のように再帰的に定義される．

$$\gamma_f(e) = \begin{cases} \gamma_f(e_j) \prod_{i \neq j} \#(e_i) & f \in \varphi(e_j) \wedge e = e_1 \cdot \cdots \cdot e_n \\ \gamma_f(e_j) & f \in \varphi(e_j) \wedge e = e_1 + \cdots + e_n \\ 1 & f = e \\ 0 & e \text{ is a feature and } f \neq e \end{cases}$$

また，$\#(e)$ は選択式 e から導出される製品の数であり，以下のように再帰的に定義される．

$$\#(e) = \begin{cases} \prod_{i=1}^{n} \#(e_i) & e = e_1 \cdot \cdots \cdot e_n \\ \sum_{i=1}^{n} \#(e_i) & e = e_1 + \cdots + e_n \\ 1 & e \text{ is a feature} \end{cases}$$

例えば，選択式 $e = a + b(c+d)$ と e 中の部分式 $e' = b(c+d)$ と $e'' = (c+d)$ があり，選択式 e 中のフィーチャ b のセレクティビティを計算することを考える．まず $\#(e)$ を計算する．

$$\begin{aligned} \#(e) &= \#(a) + \#(e') \\ &= \#(a) + \#(b) \times \#(e'') \\ &= 1 + 1 \times 2 = 3 \end{aligned}$$

次に $\gamma_b(e)$ を計算する．

$$\gamma_b(e) = \gamma_b(e') = \gamma_b(b) \times \#(e'') = 2$$

これらより，フィーチャ b のセレクティビティは次のようになる．

$$\sigma_b(e) = \gamma_b(e) / \#(e) = 2/3$$

セレクティビティは，製品バリエーション全体の中のそのフィーチャを含む製品の数の割合である．そのためこの数

値は，選択式 e から導出されることのできる製品バリエーション全体のうち，フィーチャ b を含む製品が 2/3 の割合で存在するという意味となる．

4.6 SDPD アプローチの特徴と課題

SDPD アプローチはフィーチャモデルからの最小の決定シーケンスでの製品導出を目的としており，フィーチャモデルの構造により計算された，選択されやすさを表すセレクティビティという数値をフィーチャに付加することでこれを実現している．

SDPD アプローチの特徴として，意図される製品が製品バリエーションの中に存在するならば，どのようなフィーチャモデルに対しても必ず 1 つ製品が導出されるということである．つまり，意図される製品が妥当であれば SDPD アルゴリズムは製品導出が完了するまで動き続け，必ず 1 つの製品を出力するということである．ここでいう妥当な製品とは，フィーチャモデルの構造と制約に対して矛盾のないフィーチャのセットのことである．ただし製品導出アルゴリズム中でのフィーチャの選択順序はセレクティビティに依存して決定されるが，セレクティビティはあくまでフィーチャモデルの構造や制約から計算された選択の優先度であるため，最大のセレクティビティを持つフィーチャが必ずしも製品の機能として最も重要なフィーチャであるというわけではない．つまり，SDPD アプローチはフィーチャ毎の意味や重要性を考慮した製品導出ではないということである．

SDPD アプローチの課題として，入力とする情報が機能に特化したフィーチャモデルの構造や制約を表現した選択記述と意図される製品（ユーザの要求するフィーチャのセット）のみなので，各フィーチャの持つ非機能特性を考慮できない点が挙げられる．そのため，SDPD アプローチで製品を導出したとしても，非機能特性を考慮していないためにその製品を利用できない可能性がある．

5. 研究目的と研究アプローチ

本研究では SDPD アプローチを，非機能特性を考慮できるように拡張した拡張 SDPD アプローチを提案することを目的とする．また本研究で扱う非機能特性はメモリ使用量とする．その理由は，メモリ使用量への要求が特にデバイスの制約が強い組込みソフトウェア開発分野で非常に重要視されているからであり，またこの特性は定量化可能な特性であるからである．

本研究で提案する拡張 SDPD アプローチは SDPD アプローチに基づいた拡張を行っているが，製品バリエーション中で妥当な製品を要求として与えたとしても必ず 1 つの製品が導出されるというわけではない．つまり，要求された製品が製品バリエーション中に確実に存在するとしても，本アプローチでその要求を満たした製品が確実に 1 つ導出されることは保証できないということである．そのため，本アプローチはあくまで製品導出を支援する手法であり，従来の SDPD アプローチと同様の状況で使用されることは想定していない．この問題については 8 章で詳しく述べる．

拡張 SDPD アプローチを提案するにあたって解決しなければならない課題は主に 3 点挙げられる．1 点目は入力とするモデルの決定である．これは従来の SDPD アプローチの入力が機能に特化したフィーチャモデルを表現した選択記述であるため，このフィーチャモデルのみで非機能特性を考慮するのは難しいからである．2 点目は非機能特性を考慮するためのアルゴリズムの拡張である．これは，本アプローチではアルゴリズム中で要素の非機能特性を考慮するためである．3 点目は入力とするモデルの選択記述への変換，及び限定操作の拡張である．これは入力とするモデルを変更するため，それに伴い選択記述や限定操作の定義を拡張する必要があるからである．

6. 提案手法

6.1 入力とするモデル

本研究では提案する拡張 SDPD アプローチの入力とするモデルとして N.Siengmund らが提案した統合ソフトウェアプロダクトラインモデル（ISPLM: Integrated Software Product Line Model）[7]を利用する．

6.1.1 ISPLM の概要

N.Siegmund らは，非機能特性は実装に密接に関連しているため，フィーチャモデルのみで非機能特性を考慮した個別製品の構成の決定は困難としている．そこで彼らは，フィーチャモデルと実装モデル，そして非機能特性の情報を統合した統合ソフトウェアプロダクトラインモデル（ISPLM: Integrated Software Product Line Model）を提案している．ISPLM では様々な非機能特性の表現を考慮しているが，本研究ではメモリ使用量の表現に ISPLM を活用する．

図 7 はデータベース管理システム（DBMS）の ISPLM である．

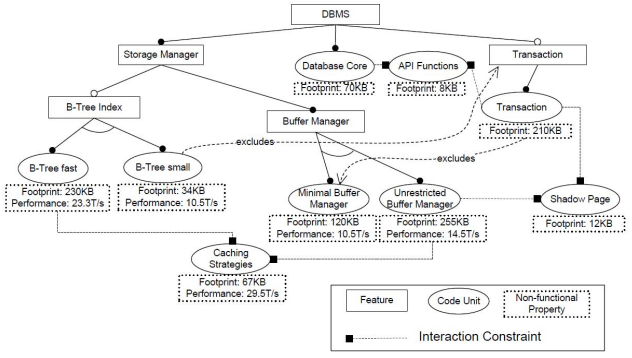


図 7　DBMS の ISPLM[7]

ISPLM はフィーチャモデルにコードユニット記述を拡張したモデルである．楕円形の要素は実装のコードユニットを表し，黒塗りの四角で繋がれている要素はコードユニット間のインタラクションを表す．例えば，2 つのコードユ

ニットのメモリ使用量を考えた際，コードユニット間で同じコードを用いている場合等にインタラクションが発生し，メモリ使用量が単純な和にならないことがある．図 7 の例でいうと，コードユニット B-Tree fast とコードユニット Unrestricted Buffer Manager 間にインタラクションが発生し，この 2 つのコードユニットを使用した場合にそれぞれのメモリ使用量に加え，67KB のインタラクションが発生する．

図 8 に ISPLM のメタモデルを示す．図 8 を見ると，基本的な構造はフィーチャモデルと変わらないが，そこにインタラクション，コードユニット，非機能特性が加えられていることがわかる．

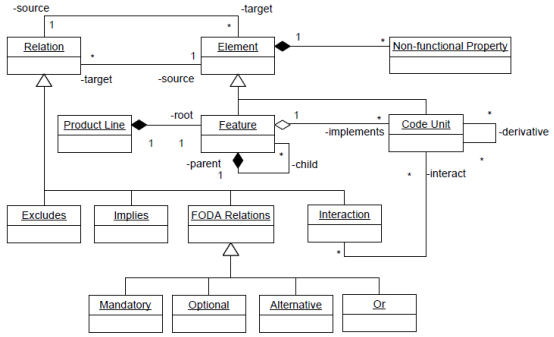


図 8　ISPLM のメタモデル[7]

6.1.2 ISPLM を入力とする理由

ISPLM は前述した通り，フィーチャモデルと実装モデルと非機能特性の情報を統合したモデルである．ISPLM は非機能特性の情報が付加されたモデルであるため，ISPLM を SDPD アプローチの入力とすることで非機能特性を製品導出時に考慮できるようになる．また図 8 を見ると ISPLM では FODA で定義された関係（FODA Relations）をそのまま利用しているため，SDPD アプローチにおけるフィーチャモデルから選択記述への変換のアイデアを利用しやすい．そのため，本研究では ISPLM をインプットとした拡張 SDPD アプローチを提案する．

6.2 SDPD アルゴリズムの拡張

本研究で提案する拡張 SDPD アプローチでは非機能特性をアルゴリズム中で考慮できるようにするために，要素の選択の際，次回以降の選択で使用されると予想される潜在的なメモリ使用量を計算し，その値に基づいて対象の要素を選択するかどうかの決定を行う．そのために本アプローチでは，確定メモリ使用量 *dfootprint* と潜在的最小メモリ使用量 *pfootprint* を導入した．確定メモリ使用量とは現時点で選択された要素のメモリ使用量の総和のことを指し，潜在的最小メモリ使用量とは現時点で選択された要素に加え，現時点で選択された要素の直下の要素の内最もメモリ使用量の小さい要素を選択した場合のメモリ使用量の総和のことを指す．

本研究で拡張した SDPD アルゴリズムを図 9 に示す．まずアルゴリズムの入出力であるが，入力に非機能制約を追加している．非機能制約は全体のメモリ使用量をどのくらいまでの値に収めるかという制約である．また選択記述 (e,c) は ISPLM を選択記述形式に変換したものであり，その変換方法については後述する．

次に図 9 の METHOD 部分について述べる．既存の SDPD アルゴリズムからの主な拡張点は(5)，(6)，(10)である．まず(5)について， *pfootprint* から最もセレクティビティの高い要素 f のある Alternative 中で最小のメモリ使用量を持つ要素のメモリ使用量を引いている．この操作をする理由は，前回の選択時に計算された *pfootprint* にこの Alternative 中で最小のメモリ使用量を持つ要素のメモリ使用量が加えられているため，この要素のメモリ使用量を引かずに次の(6)の操作を行うと，この Alternative 中から 2 つの要素の選択（もしくは同一の要素を二重に選択）することを考慮することになってしまい，正常な製品導出を行えなくなってしまうからである．次に(6)であるが，最もセレクティビティの高い要素 f のメモリ使用量 *f.footprint* と *pfootprint* を足している．この値が非機能制約である x を超えてしまうということは，この要素を選択すると非機能制約を満たせないということなので，この値が x を超える場合は $\overline{f}$ を f とする．最後に(10)であるが，現時点での選択が終了したときの *pfootprint* が x を超えてしまう場合，意図された製品 p では要求された非機能制約を満たすことができないということなので，SDPD アルゴリズムを終了させる．

INPUT: 選択記述(e,c)，意図された製品p，
非機能制約$total\ footprint < x$
OUTPUT: 導出される製品s
METHOD:
(1) $s \coloneqq []$とする
(2) $dfootprint = 0$, $pfootprint = 0$とする
(3) e中で最もセレクティビティの高い要素fを見つける
(4) $f \notin p$ならば，$f \coloneqq \overline{f}$とし(7)へ
(5) $f.footprint > 0$かつfがAlternative中にあるならば，
$pfootprint$からそのAlternative中で
最小のメモリ使用量を持つ要素のメモリ使用量を引き，
それを$pfootprint$とする
(6) $f.footprint + pfootprint > x$ならば$f \coloneqq \overline{f}$とする
(7) $s \coloneqq s \cdot f \cdot inclusion_f(e,c)$とする
(8) $e \coloneqq e|f$とする
(9) $dfootprint$及び$pfootprint$を計算する
(10) $pfootprint > x$ならば製品導出を終了する
(11) $\varphi(p) \not\subseteq \varphi(s)$ならば(3)へ
(12) sを返す

図 9　拡張 SDPD アルゴリズム

6.3 選択記述と限定操作の拡張

ISPLM から選択記述への変換に関して，基本的にはフィーチャモデルから選択記述への変換と同様に行う．つまり，ISPLM 中のフィーチャもコードユニットも，フィーチャモデル中のフィーチャと同様に扱うということである．しかし，ISPLM ではフィーチャモデルとは異なり，コードユニット間のインタラクションが存在する．そのため，ISPLM を選択記述へ変換するための拡張を行う必要がある．例えば ISPLM 中のインタラクションが図 10 のように存在している場合を考える．

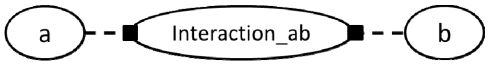


図 10　ISPLM 中のインタラクション

インタラクションは選択記述中の選択式には現れず，制約のみに現れ，制約は次のようになる．

$$a \wedge b \rightarrow Interaction\_ab$$

これは，コードユニット a とコードユニット b が同時に選択された場合，インタラクション $Interaction\_ab$ も選択されなければならないという制約である．

インタラクションは選択式には存在しない要素であるため，限定操作の拡張も必要となる．なぜなら既存の SDPD アプローチの限定操作はフィーチャモデルの構造上，選択式に存在しない要素は制約式にも存在しないことを前提としているからである．本研究で行った拡張は，リテラル化された制約の中にインタラクションである要素が含まれている場合のみ，選択式にその要素を追加するというものである．例として以下の限定操作を考える．

$$a(b+c)(d+e)\,|\,be\alpha$$

ここで α 以外の要素はコードユニットであり，α は b と e の間のインタラクションであることを想定する．この限定操作は以下のようになる．

$$\begin{aligned} &a(b+c)(d+e)\,|\,be\alpha \\ &=(a(b+c)(d+e)\,|\,be)\,|\,\alpha \\ &=abe\,|\,\alpha \\ &=abe\alpha \end{aligned}$$

コードユニット b とコードユニット e は選択式に存在するため，従来の限定操作と同様の操作を行っている．インタラクション α は選択式に存在していないが，そのまま選択式に追加することとしている．

7. 拡張 SDPD アプローチの適用例

拡張 SDPD アプローチの適用例として図 11 の ISPLM を考える．入力とする要求は，“フィーチャ C の使用”，“メモリ使用量が 400KB 未満” の 2 つである．

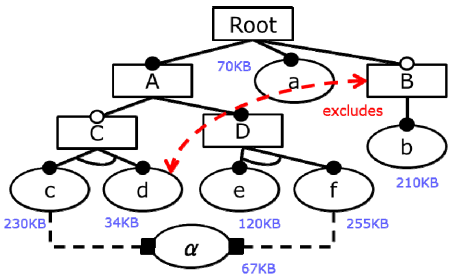


図 11　適用例で用いる ISPLM

まず製品導出の前に ISPLM の選択記述への変換を行う必要がある．図 11 の ISPLM を選択記述へ変換すると次のようになる．

$$Root = \begin{pmatrix} A(1+C(c+d))D(e+f)a(1+Bb) \\ ,\{d \rightarrow \neg B, c \wedge f \rightarrow \alpha\} \end{pmatrix}$$

次に制約のリテラル化を行う．

$$tru(d \rightarrow \neg B) = \{\{d, \overline{B}\}, \{\overline{d}\}\}$$

$$tru(c \wedge f \rightarrow \alpha) = \{\{c, f, \alpha\}, \{\overline{c}\}, \{c, \overline{f}\}\}$$

制約のリテラル化の後，そのリテラルを用いて限定操作を行う．2 つ以上制約が存在する場合，限定操作はそれぞれのリテラルの組み合わせ全てで行う必要がある．つまりこの例の場合，6 通りの限定操作を行う必要がある．限定操作を行った選択記述が以下である．

$$\begin{aligned} Root = (&ACdD(e+f)a + ACcDfa(1+Bb)\alpha \\ &+ AD(e+f)a(1+Bb) + ACcDea(1+Bb), \phi) \end{aligned}$$

限定操作により制約を選択式に反映させたこの選択記述と，前述の要求を入力とし製品導出を行う．

表 1 は製品導出の前半 3 回の選択の結果である．

表 1　適用例の製品導出（No.1～3）

No.1	選択式	$ACdD(e+f)a+ACcDfa(1+Bb)\alpha$ $+AD(e+f)a(1+Bb)+ACcDea(1+Bb)$					選択の結果
	要素	**A**	B	C	D	a	A を選択 →D を自動選択
	セレクティビティ	**10/10**	4/10	6/10	10/10	10/10	
	要素	b	c	d	e	f	
	セレクティビティ	4/10	4/10	2/10	5/10	5/10	
	dfootprint	0KB					
	pfootprint	120KB					
No.2	選択式	$ACdD(e+f)a+ACcDfa(1+Bb)\alpha$ $+AD(e+f)a(1+Bb)+ACcDea(1+Bb)$					選択の結果
	要素	A	B	C	D	**a**	a を選択
	セレクティビティ		4/10	6/10		**10/10**	
	要素	b	c	d	e	f	
	セレクティビティ	4/10	4/10	2/10	5/10	5/10	
	dfootprint	70KB					
	pfootprint	190KB					
No.3	選択式	$ACdD(e+f)a+ACcDfa(1+Bb)\alpha$ $+AD(e+f)a(1+Bb)+ACcDea(1+Bb)$					選択の結果
	要素	A	B	**C**	D	a	C を選択（要求より）
	セレクティビティ		4/10	**6/10**			
	要素	b	c	d	e	f	
	セレクティビティ	4/10	4/10	2/10	5/10	5/10	
	dfootprint	70KB					
	pfootprint	224KB					

太枠で囲まれている要素はその時点で最大のセレクティビティを持つ要素である．No.1 では A の選択に伴い，D が自動選択されており，*pfootprint* は 120KB となっている．これは D の直下の要素の内最もメモリ使用量の小さい要素（e）が選択されたと仮定した場合のメモリ使用量である．No.2 では a が選択されているため，*dfootprint* は 70KB，*pfootprint* は 190KB となっている．No.3 では C が選択されているが，これは要求“フィーチャ C の使用”に基づく選択である．

表 2 は製品導出の後半 2 回の選択の結果である． No.4 で最大のセレクティビティを持つ要素は c であるが， c は 230KB のメモリ使用量を持つ要素である．No.3 終了時点で

の *pfootprint* は 224KB であり，*c* を選択すると要求“メモリ使用量が 400KB 未満”に違反してしまうため，*c* は選択されず除去される．これにより *d* が自動選択され，それに伴い *B* が自動除去，さらに *b* も自動除去される．No.5 では *e* が選択され，それに伴い *f* が自動除去される．この時点で，全ての要素が選択，もしくは除去されたため導出を終了する．今回の適用例では 5 回の決定シーケンスで導出が完了し，導出された製品は *ADaCde* となり，この製品のメモリ使用量は 224KB となる．この製品は要求を満たしているため，導出は成功であると言える．

表 2　適用例の製品導出（No.4～5）

No.4	選択式	$ACdD(e+f)a+ACcDfa(1+Bb)\alpha$ $+ACcDea(1+Bb)$					選択の結果
	要素	*A*	*B*	*C*	*D*	*a*	
	セレクティビティ		2/6				*c*を除去（制約違反）
	要素	*b*	*c*	*d*	*e*	*f*	
	セレクティビティ	3/6	4/6	2/6	3/6	3/6	→*d*を自動選択 →*B*を自動除去 →*b*を自動除去
	dfootprint	104*KB*					
	pfootprint	224*KB*					
No.5	選択式	$ACdD(e+f)a$					選択の結果
	要素	*A*	*B*	*C*	*D*	*a*	
	セレクティビティ						*e*を選択
	要素	*b*	*c*	*d*	*e*	*f*	
	セレクティビティ				1/2	1/2	→*f*を自動除去 →製品導出終了
	dfootprint	224*KB*					
	pfootprint	224*KB*					

8. 考察

拡張 SDPD アプローチは，非機能特性を考慮できるように従来の SDPD アプローチを拡張したアプローチである．従来の SDPD アプローチでは前述の通り，どのようなフィーチャモデルからも必ず 1 つ製品を導出することができる．これはアルゴリズムにおいてフィーチャモデルの構造と制約から計算されたセレクティビティのみを考慮しているためである．

本研究で提案した拡張 SDPD アプローチでは，意図された製品が製品バリエーションの中で妥当であっても，最小の決定シーケンスで必ずしも 1 つの製品を導出できるとは限らない．なぜなら，本アプローチでは ISPLM の構造や制約のみでなく，メモリ使用量も考慮した要素の選択を行っているため，最小の決定シーケンスで非機能制約を満たした製品を導出できるとは限らないからである．そのため，本アプローチの有効性の検証を行う必要があり，それに関しては数学的に証明する，もしくはツールを実装し，様々なパターンの SPL に対してシミュレーションを行う方法が考えられる．本研究ではシミュレーションを行って検証することを考えており，シミュレーションの結果として要求を満たした製品を導出できるパターンと導出できないパターンが特定できれば，本アプローチを有効に利用できる状況について考察することができる．今後本提案に基づきツールを実装し，シミュレーションを行うことで，本アプローチの有効性を確認していきたい．

9. おわりに

本研究では，自動的で非機能特性を考慮できる製品導出支援手法の提案を目的とし，ISPLM を入力とする拡張 SDPD アプローチを提案した．適用例では実際に ISPLM を選択記述に変換し，それを入力として提案した拡張 SDPD アルゴリズムにより要求を満足した製品を導出した．

今後の課題として以下の 3 点が挙げられる．まず 1 点目は拡張 SDPD アプローチの有効性の確認である．本アプローチをツール化することで実際に様々な SPL に対して本アプローチを適用することができ，それを評価することで考察でも述べたように本研究の有効性の検証や，本研究が用いられるシチュエーションを考察することができる．2 点目として挙げられるのが計算量の大きさの問題である．本研究で提案したアプローチではセレクティビティや *dfootprint*，*pfootprint* を選択の度に再計算する必要があるため，入力とする ISPLM の規模が大きくなるにつれ計算量も膨大になってしまう恐れがある．最後に 3 点目として，考慮できる非機能特性の種類が挙げられる．本研究では，考慮する非機能特性はメモリ使用量に限定したが，その他にも多くの重要な非機能特性が存在する．これら 3 点の課題を中心に，本研究で提案したアプローチを今後改良していく予定である．

参考文献

1) Sheng Chen, Martin Erwig, “Optimizing the Product Derivation Process”, In Proceedings of 15th International Software Product Line Conference(SPLC2011), pp.35-44, 2011
2) Norbert Siegmund, Marko Rosenmüller, Christian Kästner, Sven Apel, Sergiy S. Kolesnikov, “Scalable Prediction of Non-functional Properties in Software Product Lines”, In Proceedings of 15th International Software Product Line Conference(SPLC2011), pp.160-169, 2011
3) Klaus Pohl, Gunter Bockle, Frank van der Linden 著，林好一，吉村健太郎，今関剛訳，『ソフトウェアプロダクトラインエンジニアリング』，株式会社エスアイビー・アクセス, p.14, 2009
4) Norbert Siegmund, Marko Rosenmüller, Martin Kuhlemann, Christian Kästner, Sven Apel, Gunter Saake, “SPL Conqueror: Toward optimization of non-functional properties in software product lines”, In Proceedings of Software Quality Journal Volume 20, pp.487-517, 2012
5) K. Kang, S. Cohen, J. Hess, W. Novak, A.Peterson, “Feature-Oriented Domain Analysis (FODA) Feasibility Study”, In Proceedings of Technical Report CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon University, 1990
6) Norbert Siegmund, Marko Rosenmüller, Martin Kuhlemann, Christian Kästner, Sven Apel, Gunter Saake, “Measuring Non-functional Properties in Software Product Lines for Product Derivation”, In Proceedings of the Asia-Pasific Software Engineering Conference(APSEC), pp.187-194, 2008
7) Norbert Siegmund, Martin Kuhlemann, Marko Rosenmüller, Christian Kästner, Gunter Saake, “Integrated Product Line Model for Semi-Automated Product Derivation Using Non-Functional Properties”, In Proceedings of the International Workshop of Variability Modelling of Software-Intensive Systems(VaMoS), pp.25-31, 2008