IntelligentPad における部品の抽象的仕様記述と開発への応用

平野亮太[†,††] 田中譲[†,††]

アプリケーションを部品の組み立てによって開発する技術体系としてコンポーネントウェアが注目されている．部品組み立て方式のアプリケーション開発を支援するには，ドメインに応じたアプリケーションフレームワークの構築と提供，そのカスタマイズによる再利用が効果的である．再利用性の高い有用なフレームワークの構築には，そのアーキテクチャの比較分析による洗練や，仕様を部品に切り出し，合成部品に組み立てることが必要である．本稿では，IntelligentPad を基盤に，部品のインタフェースと，インタフェース間の依存関係に着目した部品の抽象的な仕様記述法としてパターン記述法を提案する．また，それを開発に利用するために，パターン記述に有向グラフとしての表現形式を与え，それを利用したパターン記述の分解，合成法，比較法について述べる．これらの方法は，仕様の部品への切り出し，パターンマッチングによる部品検索，パターンの分解と部品再利用によるアプリケーション開発法に応用することができる．

Pattern Description in the IntelligentPad System, and Its Application to System Development

RYOTA HIRANO[†,††] and YUZURU TANAKA[†,††]

Componentware as a new software developent technology has been attracting much attention during the last decade. One possible approach to support application development in componentware systems is to provide an application framework for each application domain, and to allow us to reuse such application frameworks. In this paper, We introduce a semi formal method to describe the interface and the abstract behavior of components as pattern descriptions based on IntelligentPad system. This method provides a way to describe both existing applications and problem specifications as patterns. We will show composition and decomposition operators, using graph representation of patterns, and propose the method for pattern maching over applications. These methods can be used both to translate specifications to the composition with components, and to search a component library for required components.

1. はじめに

従来のソフトウェア部品化，再利用技術に欠けていた部品のパッケージ化，部品間インタフェースの標準化を実現し，ソフトウェアを部品の組み立てによって開発する新たな技術体系としてコンポーネントウェア[14]が注目されている．この種の開発ツールとしては Visual Basic, Delphi, Java Beans, OpenDoc[5], IntelligentPad[18] などが知られている．コンポーネントウェアでは，各部品はオブジェクトコードで提供され，システムに動的に組み込むことができるプラグ&プレイ型ソフトウェア部品によって容易に再利用することができる．

† 北海道大学大学院工学研究科電子情報工学専攻
Graduate School of Engineering, Hokkaido University
†† 北海道大学知識メディアラボラトリー
Meme Media Laboratory, Hokkaido University

コンポーネントウェアにおける開発方法論や開発プロセスは十分には明らかにされていない[17]．しかし有効な手段の1つが，ドメインに応じたアプリケーションフレームワークの提供と，そのカスタマイズによる再利用である．フレームワークは，ある問題領域を解決する基本的なアーキテクチャの提示と，その領域のサポート部品群の提供，組み立てルールの明示からなる[10),11)]．再利用性の高い部品やフレームワークを抽出するには，アーキテクチャの比較分析による洗練と，仕様の適切な部品組み立て構造への分解を行う必要がある．

本稿は，構築すべきシステムに対して，その部品組み立て構造への分解と，必要となる部品と既存部品の間の比較分析を可能にするために，部品の抽象的記述法を提案し，それを用いたシステム記述の分解・合成の方法と，システム相互ならびに部品相互の比較分析の方法を提案する．部品の組み立ては，部品のインタ

フェースの情報をもとに行われるため，インタフェースの振舞いの厳密な記述が必要である．また，部品を組み立てた合成部品によってアプリケーションを実現するため，単一部品や合成部品の振舞いの統一的な取扱いが必要である．本稿では，部品のインタフェースとその振舞いを，部品の接続点の集合と接続点間の参照/更新の依存関係の集合によって抽象記述化する．この記述を本稿ではパターン記述と呼ぶ．振舞いを厳密に記述するわけではないが，単一部品と合成部品との統一的な取扱いを目標とする．各部品や合成部品のパターン記述は一意に定まるが，逆は必ずしも成り立たない．パターン記述が部品や合成部品の振舞いを厳密に記述しているわけではないからである．

パターン記述に，有向グラフとしての表現を与え，グラフ理論における連結グラフの分解を応用することで，パターンの分解と合成方法を定義する．また，グラフの構造とノードの型に基づくパターンの比較方法を示す．パターンの分解と，比較の方法の確立は，与えられたシステム記述を実現可能な部分問題に分解し，既存部品を再利用することにより，段階的にアプリケーション開発を行うことを可能にする．

パターンの比較分析の方法の確立は，同じ問題領域のいくつかのアプリケーションアーキテクチャを抽象レベルで比較することを可能にし，その共通部分，変動部分の抽出を可能にする．共通部分はアプリケーションフレームワークの基本的なアーキテクチャとして活用され，その領域のアプリケーションの基本的な構造スキーマを提供する．このような構造スキーマを解析，記述し，設計の効率化を図る再利用技術にパターンがある．パターンは，ある繰り返し起こる設計上の問題と，それに対する解決策を記述したものであり，様々なアーキテクチャを設計するための部品となる．建築家の Alexander ら[1] が提案した概念であり，オブジェクト指向ソフトウェア設計にもこの考え方が適用されている[2),4),6),12)]．パターンはアーキテクチャの構成要素や各要素の役割を明確にし，その理解を容易にしたり，拡張や修正への柔軟な対処が期待できる[7)]．その有用性は IntelligentPad におけるアプリケーション開発でも報告されている[13),19)]．

本研究ではアプリケーション開発の基盤システムとして IntelligentPad[18)] を用いることを仮定する．IntelligentPad では部品のパッケージ化，部品間インタフェースの標準化，部品組み込みのためのソフトウェアアーキテクチャの標準化を実現しており，視覚的な部品「パッド」をユーザの直接操作で結び付け，合成パッドに組み立てることができる．木構造に沿った部品の組み立てのみが許され，組み立て構造において隣接する 2 つの部品間の機能連係結合は，各部品から 1 つずつ選ばれた接続点を結合して定義される．本稿で提案するパターン記述の分解方法は，IntelligentPad の機能連係モデルに基づく．機能連係の単純さは部品の組み合わせ方法に制約を与えるが，1 つの部品の機能や利用方法を単純化するという長所がある．部品の単純さは，エンドユーザによる合成部品の分解，部品の追加や変更によるカスタマイズを容易にすることが期待できる．また，対象とするアプリケーションはクライアントシステムとし，サーバシステムの開発は対象としない．DBMS などのサーバシステムと比較して，クライアントシステムは多様化しており，柔軟性の高さ，開発コストの低減，開発の即時性が，強く求められてきているからである．

以下，2 章では基盤システムとして用いる IntelligentPad について説明する．3 章では，本稿が提案するパターン記述法と，パターン記述のグラフ表現法について述べる．4 章ではグラフに基づくパターン記述の比較法について述べる．5 章では，パターン記述の分解法，合成法を示す．6 章では，パターン記述の分解を利用したアプリケーション開発法について述べ，7 章では具体的なアプリケーションの開発例と比較分析の例を示す．

2. IntelligentPad

IntelligentPad[18)] では紙のメタファを持つ可視的な部品を「パッド」と呼び，これらのパッドをユーザの直接操作で結び付けることで，合成部品（合成パッド）を木構造に沿って組み立てることができる（**図 1**）．各パッドは，AV 機器が複数のコネクションジャックを装備するように，外部インタフェースとなるスロットを複数個定義することができる．これをスロットリストと呼ぶ．各スロットには，外部からの書き込みと，読み出しアクセスに応じた関数を 2 つ定義することができ，それらの集合がパッドの振舞いを決定する．

また，各パッドには自身を別パッドのスロットに結合するためのピンプラグが 1 本だけ用意されている．

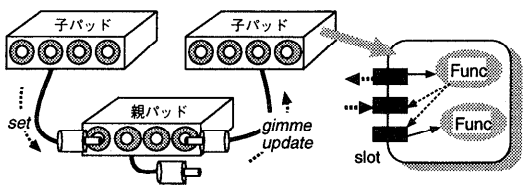


図 1 パッドの基本構造
Fig. 1 IntelligentPad architecture.

このピンプラグを通しての入出力は特別なスロットが行う．このスロットはプライマリスロットと呼ばれる．子パッドのプライマリスロットを親パッドが持つ1つのスロットに結合し，スロットデータの授受を可能にすることで，スロット結合によるパッド間の機能連携が可能になる．スロット結合する親パッド，子パッドの各スロットは結合スロットと呼ぶ．

3. 部品の記述方法

本章では，IntelligentPad を基盤としたパッド（部品）の抽象的記述法について述べる．

3.1 単一部品

部品を特徴づける要素として，部品のインタフェースと，その振舞いに注目する．ある部品 P は外部の他の部品とデータやシグナルのやりとりを行うための入出力口（スロット）を持つ．各スロットには，そのスロットがアクセスされたときに実行される関数（アクセス関数）が定義されており，その関数の実行が引き起こす他のスロットの変化を，その部品の抽象的な振舞いとしてとらえる．

単一部品が持つスロットの集合をスロットリストと呼び，S で表すことにする．各スロット $s \in S$ は名前（識別子）と，そのスロットに格納されるデータ型とからなるものとする（#slotName: type）．

部品の振舞いは，アクセス関数を，スロット間の参照/更新の関係のみに注目して抽象化する．アクセス関数の実行によって，更新される1つのスロット（更新スロット）に注目し，更新時に参照されるいくつかのスロット（参照スロット），更新のきっかけとなった1つのスロット（アクセススロット）を明らかにし，これらの関係を依存関係（dependency）として記述する．1つのアクセス関数から得られる依存関係の数は，そのアクセス関数の実行で更新されるスロットの数と同じである．すべてのアクセス関数について得られる依存関係の集合 D を，その部品の抽象的振舞い記述とする．各々の依存関係 $d \in D$ は式 (1) の形式に従って記述する．

$$\langle access\ slot \rangle \rightarrow \langle update\ slot \rangle\ (\langle ref\ slots \rangle) \tag{1}$$

たとえば，#x へのアクセスが #y を更新し，#y の更新時に $\#s_1, \#s_2, \cdots, \#s_n$ が参照されるとき，#x から #y に対して，$\#s_1, \#s_2, \cdots, \#s_n$ を参照とする依存関係があるという．#x がアクセススロット $\langle access\ slot \rangle$，#y が更新スロット $\langle update\ slot \rangle$，$\#s_1, \#s_2, \cdots, \#s_n$ が参照スロット $\langle ref\ slots \rangle$ である．参照スロットを無視したとき，依存関係は推移的

スロットリスト S：
- #index : *number*
- #current : α
- #collection : α *list*
- #size : *number*
- #addFirst : α
- #addLast : α

依存関係集合 D：
- #index → #current (#index, #collection)
- #collection → #current (#collection, #index)
- #collection → #size (#collection)
- #addFirst → #collection (#addFirst, #collection)
- #addLast → #collection (#addLast, #collection)

図 2 単一部品の記述例：CollectionPad
Fig. 2 An example description of a primitive pad.

である．2つの依存関係 $\#s_1 \rightarrow \#s_2$，$\#s_2 \rightarrow \#s_3$ が成立するとき，$\#s_1 \rightarrow \#s_3$ も成立する．

部品 P のパターン記述 $ptrn(P)$ を式 (2) のように，スロットリスト S と依存関係の集合 D の組で表す．

$$ptrn(P) = \langle S, D \rangle \tag{2}$$

パターン記述は部品の振舞いを抽象的に記述したものである．単一部品パッド CollectionPad の $\langle S, D \rangle$ によるパターン記述を図 2 に示す．

3.2 合成部品

2つの部品 A, B の合成は，各々の部品が持つ1つずつのスロットを結合することで実現される．これをスロット結合による合成部品の導出と呼ぶ．部品 A, B のパターン記述を各々 $ptrn(A) = \langle S_A, D_A \rangle$, $ptrn(B) = \langle S_B, D_B \rangle$ とし，結合スロットを #a, #b とすると合成部品 AB のパターン記述 $ptrn(AB)$ は式 (3) のようになる．

$$\begin{aligned} ptrn(AB) &= \langle S_A \cup S_B,\ D_A \cup D_B \cup D_C \rangle \\ &= \langle S_{AB},\ D_{AB} \rangle \end{aligned} \tag{3}$$

各スロットはユニークであり，重複はない（$S_A \cap S_B = \phi$, $D_A \cap D_B = \phi$）．D_C はスロット結合によって新たに追加される依存関係の集合である．A の結合スロット #a が B の結合スロット #b に結合するとき，結合状態 θ に違いよって次の依存関係が追加される．

$$\theta = \begin{cases} \rightarrow & D_C = \{\#a \rightarrow \#b(\phi)\} \\ \leftarrow & D_C = \{\#b \rightarrow \#a(\phi)\} \\ \leftrightarrow & D_C = \{\#a \rightarrow \#b(\phi), \\ & \qquad \#b \rightarrow \#a(\phi)\ \} \end{cases}$$

結合状態 θ はスロット値の送出される方向を表す．

#index → #current (#index, #collection)
#collection → #current (#collection, #index)
#collection → #size (#collection)
#addFirst → #collection (#addFirst, #collection)
#addLast → #collection (#addLast, #collection)
#down → #count (#count)
#up → #count (#count)
#count → #index (ϕ)

図 3 合成部品の記述例
Fig. 3 An example dependency set of a composite pad.

たとえば，θ が → であるとは，A のスロット #a の値が，B のスロット #b に送出されることを意味する．スロット結合の場合，#a の更新が無条件に #b に送出されるため，依存関係の参照スロット $\langle ref\ slots \rangle$ は ϕ となる．

合成部品 AB は，A を台紙（親）にしたとき $A[\#a\,\theta:\ B]$，もしくは θ や結合スロットを省略し $A[\#a:\ B]$ や $A[B]$ のように書くことにする．B を台紙としたときは $B[\#b\,\theta:\ A]$ と書く．

合成部品の記述例として，CollectionPad[#index ←: CounterPad] の依存関係の集合 D を図 3 に示す．この集合 D をよく見ると，図 2 の単一部品 CollectionPad の依存関係の集合と，CounterPad の 2 つの依存関係 #down → #count (#count)，#up → #count (#count) の集合との和がとられ，スロット結合により，D_C として #count → #index (ϕ) が追加されている．

3.3 有向グラフによる表現

スロットリスト S と依存関係の集合 D の組 $\langle S, D \rangle$ から，一意に定まるグラフ表現を得ることができる．このグラフ表現は，以下のような手順と条件により得られ，D において参照スロットを無視した依存関係に対するハッセ図を拡張した表現になる．

(1) 各依存関係の参照スロットを無視したものを，半順序関係と見なして，ハッセ図を作成する．$\#s_1 \rightarrow \#s_2$，$\#s_2 \rightarrow \#s_1$ が成り立つときは，$\#s_1$ と $\#s_2$ の間に双方向の有向辺を引き，便宜上，これらを 1 つの頂点として扱う．

(2) 各依存関係について，(1) で記述したハッセ図に参照スロットが存在していなければこれを追加し，その依存関係に含まれるアクセススロットと参照スロットをハッセ図の中で取り囲む．取り囲むスロットがアクセススロット 1 つの場合は省略する．

(3) スロットリスト S 中に，(2) で拡張したハッセ図に現れていないスロットがあれば，これらを孤立点として加える．

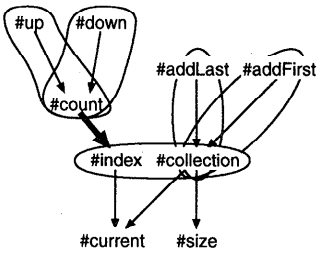


図 4 パターン記述のグラフ表現の例
Fig. 4 Example pattern graph.

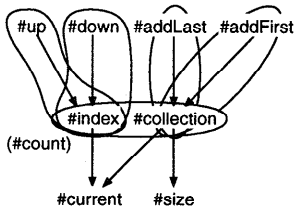


図 5 合成部品における結合辺の除去
Fig. 5 An extended pattern graph.

このような手順を経て得られる $ptrn(P)$ のグラフ $G(P)$ を式 (4) で表す．

$$G(P) = (S, U, R) \tag{4}$$

S はスロットリスト（頂点集合），U は上記の手順 (1) で得られるハッセ図の有向辺の集合である．R は手順 (2) で得られる各依存関係のアクセススロットと参照スロットを取り囲んだ特殊辺の集合とする．$G(P)$ を $ptrn(P)$ のグラフ表現と呼ぶ．

合成部品 AB の場合も同様に，式 (3) からグラフ表現を求めることができる．また，$ptrn(A)$，$ptrn(B)$ のグラフ $G(A)$，$G(B)$ を，スロット結合によって追加された依存関係から得られる辺 c で連結することでも同じグラフ表現を得ることができる．

$$G(AB) = (S_A \cup S_B, U_A \cup U_B \cup \{c\}, R_A \cup R_B)$$
$$= (S_{AB},\ U_{AB},\ R_{AB})$$

c を特に結合辺と呼ぶことにし，グラフ中では他の有向辺と区別して書く．図 4 に単一部品パッド CollectionPad と CounterPad の合成パッドのグラフの記述例を示す．スロット #count から #index への有向辺が結合辺である．また，本来ならば各頂点のスロットは名前と型を明記すべきだが，図の簡略化のため型の明記は省略する．

どんな合成パッドも明らかに単一部品パッドとして実現できる．単一部品パッドのグラフ中には，結合辺は存在しない．結合辺を含む合成パッドのグラフを，

結合辺の両端となるスロットを1つの頂点として融合し，縮退させることによって等価な単一部品パッドのグラフを得ることができる．たとえば，図4のグラフは**図5**のように書き換えることができる．このとき依存関係の集合 D から，スロット結合によって追加された依存関係は除去する．以降では，$P_1[\#s\ \theta : P_2]$ のグラフ表現として，結合辺の両端スロットを #s に縮退させたグラフ表現を用いることにする．

4. パターン記述の比較

本章ではパターン記述が拡張ハッセ図（パターングラフ）で表現できることを利用し，パターングラフの構造と頂点のスロットの型を用いて，パターン記述の比較を行う方法を示す．

4.1 同型グラフでの比較

ある部品のパターン記述のグラフ $G=(S,U,R)$ について，その頂点（スロット）の型と辺の接続関係を不変に保ち，頂点の名前を名前の衝突が起きないように変更することで得られるグラフ $G'=(S',U',R')$ はグラフ G と同型（isomorphic）であると定義する．

グラフ中のスロットの名前を変更する操作 σ を定義する．σ は1対1の写像であるとする．この操作 σ を用いることで2つの同型グラフ G, G' は式(5)のように表せる．

$$G' = \sigma\, G \tag{5}$$

また，スロットの名前の変更に加え，あるスロットの型を別の型に変更することでも，グラフの同型性を議論することができる．そこで，グラフ中のスロットの型の変更操作 ρ を定義する．この操作は，グラフ中の各スロットの型を別の型に変更する．G に型の変更操作 ρ を行って得られるグラフを ρG で表す．

4.2 包含関係での比較

2つのグラフ $G=(S,U,R)$, $G'=(S',U',R')$ について，S' が S の部分集合で，U に属する辺で S' に含まれる2頂点間を結ぶ辺の全体が U' となり，R に属する特殊辺で S' に含まれる頂点だけを囲む特殊辺の全体が R' となるとき，G' は G の部分グラフと定義する．

また，U' が U に属する辺で S' に含まれる2頂点間を結ぶ辺の全体でなく，単に部分集合であるとき，もしくは，R' が R に属する特殊辺で S' に含まれる頂点だけを囲む特殊辺でなく，S' に含まれる頂点を囲む特殊辺から，S' に含まれない頂点を除いた特殊辺の全体であるとき，G' は G の半部分グラフと定義する．

図6に部分グラフと半部分グラフの例を示す．図6中(A)を部分グラフに含むのは(B)である．(A)を半部分グラフに含むのは(C)，(D)である．

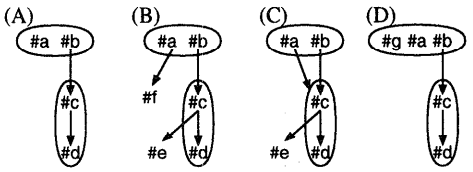


図6 部分グラフと半部分グラフの例
Fig. 6 Example subgraph and partial graph.

表1 比較の種類と関係
Table 1 Pattern matching operators.

prefix	R と式(8)右辺
iso	$G_q = \sigma G_l$
*iso**	$G_q = \rho\sigma G_l$
sub	$G_q \subseteq \sigma G_l$
*sub**	$G_q \subseteq \rho\sigma G_l$
par	$G_q \subseteq_p \sigma G_l$
*par**	$G_q \subseteq_p \rho\sigma G_l$

グラフ G' が，G にいくつかのスロットの名前の変更を行ったグラフの部分グラフになるとき，式(6)のように表す．同様に半部分グラフとなるとき，式(7)のように表す．スロット型の変更を行ったグラフについても式(6)，(7)と同様の関係を考えることができる．

$$G' \subseteq \sigma\, G \tag{6}$$

$$G' \subseteq_p \sigma\, G \tag{7}$$

4.3 比較オペレータ

グラフの同型，包含関係をもとに，比較オペレータ（マッチング）を定義する．2つのグラフ G_q, G_l を比較するための一般的な述語を式(8)で表す．R は2つのグラフを比較するための関係演算子とする．

$$match_{prefix}(G_q,\ G_l) = G_q\ R\ G_l \tag{8}$$

先程の結果から**表1**の6つの比較オペレータを定義することができる．表1には，同型，部分，半部分の3種類のグラフの構造の比較が，スロットの名前の変更，型の変更の2つの変換に対して列挙されている．

5. パターン記述の分解と合成

3章で部品の抽象的記述法としてパターン記述を導入した．これを用いることで，少なくともすでに存在する単一部品や合成部品の振舞いを抽象的に記述することができる．本章では，構築すべきシステムのパターン記述から，このシステムの分解の可能性と，既存部品を用いた合成の可能性について論じる．

5.1 パターン記述

新しい部品やアプリケーションを開発するには，それが最終的に単一部品で実現されるか，合成部品で実

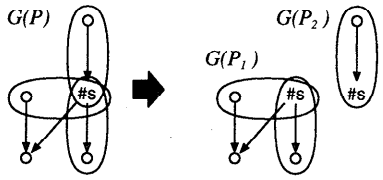


図 7 グラフの分解例
Fig. 7 An example decomposition of a pattern graph.

現されるかにかかわらず，どんなスロットが必要かをリストアップし，各スロットのアクセスに応じた関数を定義する必要がある．以降では，開発したいアプリケーションの仕様も $\langle S, D \rangle$ の形式で抽象的に記述されていることと仮定し，これらを統一的にパターンと呼び，そのグラフをパターングラフと呼ぶことにする．

5.2 パターンの分解

どんなパターンも単一部品として実現可能であるが，それを小さなサブパターンに分解し，各々のサブパターンのスロット結合による実現を考えた方が現実的である．パターンのグラフ表現を利用し，パターンを 2 つのサブパターンのスロット結合に分解する方法について考える．

パターングラフは連結な場合と，非連結な場合がある．非連結な場合，そのパターングラフは 2 つ以上の独立な連結成分を持つため，サブパターンは相互にスロット結合されることなく独立に存在しうる．このようなパターンは各々の連結成分ごとに独立した部品の単なる集まりとして実現することができる．したがって，以降では一般性を失うことなく，パターングラフは 1 つの連結グラフであると仮定して議論を進めることができる．

まず，通常のスロット結合による合成部品の導出を例に考える．2 つの部品 P_1，P_2 の合成部品 $P = P_1[\#\mathsf{s} : P_2]$ のパターングラフを式 (9) で表す．#s は P_1 の結合スロットである．$\circ_{\#\mathsf{s}}$ を合成オペレータと呼ぶ．

$$G(P) = G(P_1) \circ_{\#\mathsf{s}} G(P_2) \quad (9)$$

合成部品 P のパターングラフから，その構成部品 P_1 のパターングラフを取り出すと，**図 7** に示すように結合スロット #s を持つ 2 つの連結グラフに分解される．このとき，$G(P_1)$ を式 (10) で表す．$/_{\#\mathsf{s}}$ を分解オペレータと呼ぶ．

$$G(P_1) = G(P)/_{\#\mathsf{s}} G(P_2) \quad (10)$$

式 (10) は，合成部品を結合スロットで分解し，1 つの構成部品を取り出したものである．このことは，パターングラフ $G(P)$ に分解可能な頂点 #s が存在すれば，**図 8** に示すように $G(P)$ を部分グラフ $G(P')$ と，

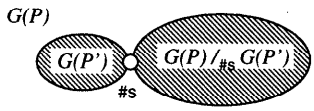


図 8 部分グラフと差分への分解
Fig. 8 Decompose a graph into subgraph and difference.

それを除いた残りの $G(P)/_{\#\mathsf{s}} G(P')$ に分解できることを意味する．$G(P)/_{\#\mathsf{s}} G(P')$ を $G(P)$ から $G(P')$ を除いた差分 (difference) と呼ぶ．また，特に $/_{\#\mathsf{s}}$ を差分オペレータと呼ぶ．このとき式 (11) が成り立つ．

$$G(P) = (G(P)/_{\#\mathsf{s}} G(P')) \circ_{\#\mathsf{s}} G(P') \quad (11)$$

分解可能な頂点の選び方は次節で説明する．

5.3 分解可能な頂点の導出

分解後の各連結グラフは，再度スロット結合による連結を行い，スロット結合辺を縮退すると分解前のグラフと同型にならなくてはならない．つまり，式 (11) が成立する必要がある．さらに，結合辺を縮退する場合，3.3 節で示すように，結合辺の 2 つの端点は 1 つの頂点に融合される．つまり，1 つに融合された頂点を 2 つの連結グラフが共有することになる．このような頂点はグラフ理論では関節点 (articulation point, 分断点) と呼ばれる．関節点は，その頂点を除去し，その頂点に接続するすべての辺を除去するとグラフが非連結になるような頂点である．

パターングラフの場合，以上の条件に加えて依存関係を考慮する必要がある．パターングラフの連結は，3.2 節で示すように，各パターンの依存関係の集合の和を意味する．つまり，分解時には依存関係の集合は，互いに重ならない 2 つの部分集合に分解される．各依存関係に対し，そのアクセススロットと参照スロットを囲む特殊辺や，アクセススロットから更新スロットへの有向辺は，パターンの分解によって分断されることはない．

以上のことから，パターングラフにおける拡張関節点を次のように定義する．各依存関係の特殊辺を，アクセススロット，参照スロットに加えて更新スロットも取り囲む拡張特殊辺としたとき，ある頂点と，その頂点に接続する有向辺の除去に加えて，その頂点を取り囲む拡張特殊辺から，その頂点のみを除去することによってグラフが非連結になるような頂点を拡張関節点と定義する．

図 9 (a) の頂点 #A2 に接続する有向辺と，拡張特殊辺から #A2 を除去してもパターングラフは連結である．したがって，#A2 は拡張関節点ではない．一方，図 9 (b) の頂点 #B3 に接続する有向辺と，拡張特殊辺から #B3 を除去するとパターングラフは非連結

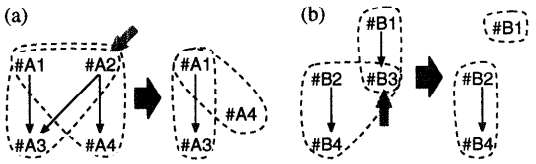
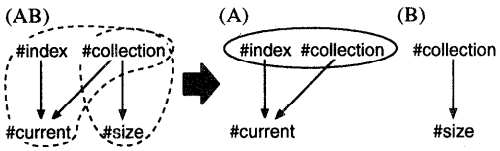


図 9 拡張関節点とそうでない頂点との違い
Fig. 9 A difference between division node and no division node.



(AB)
#index → #current (#index, #collection)
#collection → #current (#collection, #index)
#collection → #size (#collection)
(A)
#index → #current (#index, #collection)
#collection → #current (#collection, #index)
(B)
#collection → #size (#collection)

図 10 グラフと依存関係の集合の分解例
Fig. 10 An example decomposition of pattern graph and dependencies.

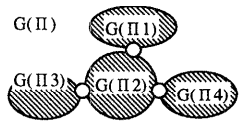
になる．したがって，#B3 は拡張関節点であり，頂点 #B1，#B2 からなる連結グラフと，頂点 #B2，#B3，#B4 からなる連結グラフに分解することができる．

2 つのパターングラフをスロット結合し，結合辺を縮退させることで得られるパターングラフにおいて，結合辺を縮退させることによって 1 つに融合された頂点は，必ず拡張関節点となる．一方，拡張関節点で分解すれば，各依存関係中のすべてのスロットを囲む拡張特殊辺は分断されず，依存関係の集合を互いに重ならない 2 つの部分集合（サブパターン）に分解することができる．つまり，以下の定理が成立する．

定理 1 あるパターン Π とそのスロット #s に対して，Π のサブパターン Π_1，Π_2 が存在して，$comp(\Pi) = comp(\Pi_1)[\#s : comp(\Pi_2)]$ または $comp(\Pi) = comp(\Pi_2)[\#s : comp(\Pi_1)]$ と分解できるための必要十分条件は #s が $G(\Pi)$ の拡張関節点であることである．

ここで，$comp(\Pi)$，$comp(\Pi_1)$，$comp(\Pi_2)$ は各々のパターンを実現する部品とする．

分解例を**図 10** に示す．図 10 のグラフ (AB) の破線で囲まれたものは，各依存関係の拡張特殊辺であり，



comp(Π1)[comp(Π2)[comp(Π3), comp(Π4)]]
comp(Π2)[comp(Π1), comp(Π3), comp(Π4)]
comp(Π3)[comp(Π2)[comp(Π1), comp(Π4)]]
comp(Π4)[comp(Π2)[comp(Π1), comp(Π3)]]

図 11 分解例と考えられる部品構造
Fig. 11 Decomposition of a pattern, and its available composition.

これにより拡張関節点 #collection が導出されている．分解後の各依存関係の集合は（図 10 (A)，(B)），分解前の依存関係の集合（図 10 (AB)）の，互いに重ならない 2 つの部分集合となっている．

また，単一部品のパターンでも拡張関節点が存在すれば，それをサブパターンに分解し，各サブパターンを実現する部品の合成で，抽象レベルで等価な合成部品を構成することができる．

5.4 合成方法

定理 1 から，パターン Π に拡張関節点 #s が存在すれば，これを $comp(\Pi) = comp(\Pi_1)[\#s : comp(\Pi_2)]$ または $comp(\Pi) = comp(\Pi_2)[\#s : comp(\Pi_1)]$ へと分解することができる．さらにサブパターン Π_1，Π_2 に拡張関節点が存在すれば，これらも同様に分解することができる．分解されたサブパターンどうしの結合構造を考えると，Π_1，Π_2 の結合構造は，閉路が存在しないため木となる．さらに，Π_1 や Π_2 を分解したときも同様に，サブパターンどうしの結合構造は木となる．つまり，パターン Π の拡張関節点での分解は，Π を木構造に構成することを意味し，IntelligentPad における木構造の部品の組み立てを保障する．たとえば，あるパターンが**図 11** の 4 つのサブパターンに分解され，各サブパターンを実現する部品の組み立て構造を考えると図 11 の 4 通り考えることができる．これらはすべて木構造となる．

6. パターンの分解と部品の再利用

5 章で示したように，パターンはパターングラフ中に存在する拡張関節点で分解することができる．このことは，抽象的システム記述として与えられたパターンを，より小さなサブパターンへと分解し，各々のサブパターンを満たす部品を用意して，それらの合成によって元々のパターンを満たすアプリケーションを開発することが可能なことを意味する．本章では，与えられたパターンに対して，差分を用いた分解によって，アプリケーション開発を行うプロセスを示す．

既存部品の再利用には，必要な部品のパターンと既存部品のパターンのマッチングが必要である．ここではパターン記述を各部品のシグニチャとして利用する．パターングラフは，シグニチャの一意な標準的表現として用いることができる．問合せのパターンと，検索対象のパターンの一致/不一致は4章で示した比較オペレータで議論できる．

構築すべきシステムのパターングラフ G を，最初に G_1 と差分 G/G_1 に分解し，さらに G/G_1 を G_2 と差分 $(G/G_1)/G_2$ へと分解する．これを幾度か繰り返すことで，最終的に n 個の連結グラフに分解されたとすると，式(12)のように表す．

$$\begin{aligned} G &= (G/G_1) \circ G_1 \\ &= (((G/G_1)/G_2) \circ G_2) \circ G_1 \\ &\quad \vdots \\ &= (((\cdots(G_n \circ G_{n-1}) \circ \cdots) \circ G_2) \circ G_1 \quad (12) \end{aligned}$$

各連結グラフ G_i に対し，パターン間のマッチングで検索された部品 P_i' のグラフ G_i' が，$G_i \subseteq op\ G_i'$ の関係を満たすとすると，G は式(13)を満たす．op は4章で示した σ または $\rho\,\sigma$ を意味するものとする．

$$\begin{aligned} G \subseteq (((\cdots(&op_n G_n' \circ op_{n-1} G_{n-1}') \circ \\ &\cdots) \circ op_2 G_2') \circ op_1 G_1' \end{aligned} \quad (13)$$

したがって，構築すべきシステムは，どの連結グラフを根とするかで幾通りかの組み合わせ方があるが，たとえば式(14)のような再利用部品のスロット結合で実現できる（スロットの明記は省略）．ただし，式(13)と式(14)は，部品のシグニチャ・マッチングのみに基づいているため，構築すべきシステムと，得られた合成部品が機能的に一致しているとは限らない．

$$P \subseteq P_1'[P_2'[\cdots[\cdots P_n']]] \quad (14)$$

分解したすべてのパターンごとにマッチする部品が見つかる必要はない．部品ライブラリの中にないものは，そのパターン記述を用いて新たな部品の開発を行う必要がある．差分の取り方を変更することにより，部品が見つかる場合もある．

図12 に差分を用いたアプリケーション開発プロセスを示す．最初に，与えられたパターンから拡張関節点を選びだし，あるサブパターンと差分とに分解する（図12左）．得られたサブパターンに対し，これにマッチする部品を部品ライブラリから探し出す（図12(1)）．マッチする部品がないときは，差分の取り方を変更して検索するか（図12(2)，(1)），新しい部品を開発する（図12(1)）．残った差分を新たなパターンとし，そのパターンのサブパターンと差分への分解と，サブパターンにマッチする部品の検索や開発を繰り返すことで最初のパターンを満たすアプリケーションを開発することができる（図12(3)）．

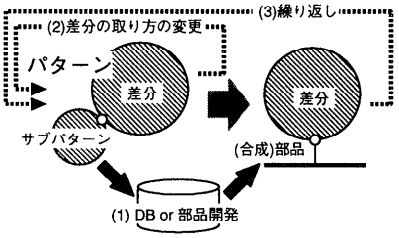


図12 差分によるアプリケーション開発プロセス
Fig. 12 Development process based on specification division.

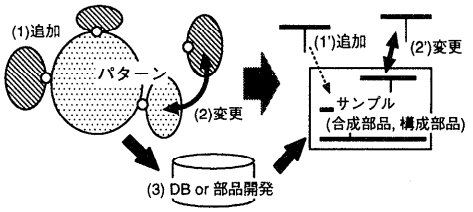


図13 カスタマイズによるアプリケーション開発プロセス
Fig. 13 Development process based on pattern reuse.

また，既存の合成部品アプリケーションとそのパターンをサンプルとして再利用するアプリケーション開発を考えることができる．**図13** に既存の合成部品とそのパターンのカスタマイズによるアプリケーション開発プロセスを示す．まず，パターンに対して，機能拡張や変更にともなう新たなスロットや依存関係の追加（図13(1)），一部の変更（図13(2)）や，削除を行うことでカスタマイズが行われる．追加や変更されるパターンを満たす部品を部品ライブラリから探し出すか，新たな部品を開発し（図13(3)），それらの部品を元々サンプルとして与えられた合成部品に追加したり（図13(1')），その合成部品の構成部品と変更したりすることによりシステムが実現される（図13(2')）．サンプルとして再利用するものが，完成度の高いアプリケーションである必要はない．既存のものから参考になるような部分を取り出し，それをカスタマイズすることも考えられる．

7. 開発例と比較分析の例

本章では，アプリケーションの開発例と，パターングラフを用いたアプリケーションの比較分析の例を示す．

図書管理システムのクライアントアプリケーションを考える．ユーザの操作インタフェースと，途中の各処理のインタフェースに注目し，以下に示すスロット

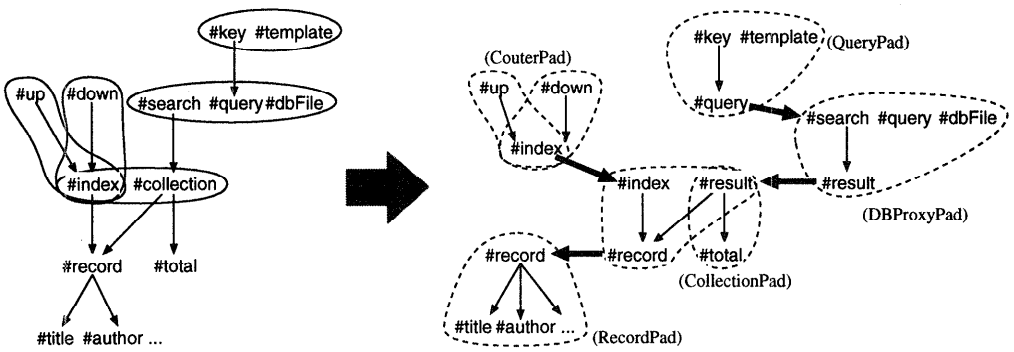


図 14 パターン記述の分解例
Fig. 14 Division of specification pattern graph.

と依存関係の抽出を行う.

- 問合せ用インタフェース
  検索に必要な検索文の入力を行う. ユーザが与えたキーをもとにSQL文を生成する. 次のような依存関係を得ることができる. スロット #template はSQL文のテンプレートである.
  #key → #query (#key, #template)
- 検索の実行
  ユーザの要求を契機に, 検索文と遠隔地DBの情報から, 検索を実行してもらい, その結果と件数を受け取る. 次の2つの依存関係が得られる.
  #search → #result (#query, #dbFile)
  #result → #total (#result)
- 検索結果の選択
  複数ある検索結果の中から, 1つのレコードをインデックスを与えることで取り出す. また, ユーザがインデックス値を増減させる操作も必要である. スロットと依存関係は次のものが得られる.
  #index → #current (#result, #index)
  #up → #index (#index)
  #down → #index (#index)
- レコードの表示インタフェース
  レコードの各フィールドの情報をユーザに示す. 各フィールドの情報を各スロットに分配する必要がある. n 個のフィールドを $\#\text{field}_i$ とすれば, 次の n 個の依存関係が得られる $(i = 1, \cdots, n)$.
  $\#\text{record} \rightarrow \#\text{field}_i$ (#record)

このようにして得られたパターングラフを**図14**(左)に示す(各スロットの型の明記は省略). また, このパターンを拡張関節点をもとに, 5つのサブパターンに分解したものを図14(右)に示す.

#up, #down, #index からなるパターンは, スロット #index の名前を #count に変更することで CounterPad のパターンと同型グラフで一致する. また, #index, #record, #result, #total からなるパターンは CollectionPad のパターンの部分グラフとなる. #search, #result, #query, #dbFile からなるパターンはデータベースアクセスを実現する部品, #key, #template, #query からなるパターンは検索文を生成する部品のパターン記述となる.

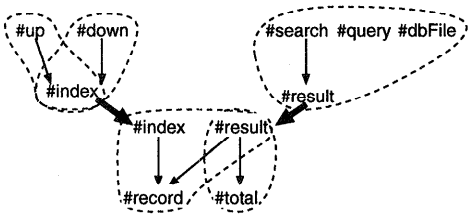


図 15 複数のアプリケーションに共通するパターン
Fig. 15 Common pattern graph over same applications.

この例は, 図書の情報に限らず, 製品情報, 顧客情報など, サーバのデータベースに格納されている情報の検索, 閲覧を行ういくつかのアプリケーションに応用することができる. それらのパターングラフを比較分析することで, 図14から**図15**の共通部分を抽出することができる. 問合せ入力や検索結果のレコードの表示には様々なバリエーションの部品が考えられるため, これらの部分はアプリケーションに依存する変動部分となる.

また, 図15の共通部分パターンの実装では, CollectionPad とデータベースアクセス部品が, それぞれ独立した部品で実現される場合と, それらを1つにした単一部品で実現される場合がある. パターンによる比較分析は, このような部品の組み立て構造の違いも吸収することができる.

8. おわりに

本稿では，IntelligentPadを基盤にして部品の振舞いの抽象的記述法をパターン記述として提案し，パターン記述の比較法，分解法，合成法を明らかにした．これにより，構築すべきアプリケーションの部品への切り出しと，合成部品での組み立てに対し，ヒントを与えることが可能となった．さらに，部品ライブラリと，パターンの分解法を利用したアプリケーション開発法を提案した．

部品の抽象的な仕様記述を開発や検索に利用した研究はいくつかある．文献15)では，MLの関数やモジュールの型情報の記述をシグニチャとして用い，その比較による部品検索を実現している．文献8)，16)では，事前/事後条件を付加した厳密な振舞いの記述と，それを用いた部品検索を実現している．本稿では，パターンをシグニチャとして用いた抽象レベルでの比較ではあるが，単一部品と合成部品の統一的な取扱いを実現している．このため，必要な部品を含む合成部品も検索対象とすることができる．文献3)では，限定された部品とコネクタによってソフトウェアアーキテクチャを表現し，パターンマッチングによるアーキテクチャの比較分析や分類を行っている．本稿で提案した方法も，アーキテクチャの比較分析が可能であるが，加えて，スロットリストと依存関係で記述されるあらゆる部品を取り扱うことができる．

本稿で示した仕様の分解手法は，ソフトウェアシステムやプログラムを，サブシステムや機能モジュール，手続きなどの部品に分割するという意味でプログラム・スライシングやAspect-Oriented Programming[9]の考え方に類似する．本稿では，仕様の部品への切り出しの実現に加え，パターン記述を用いた比較方法を示すことで，再利用性の高い部品やフレームワーク・アーキテクチャの抽出を可能にしている．

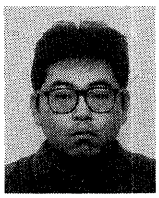
現状では，既存部品の再利用に重点を置いているため，まったく新しいアプリケーションを構築する際，どのようにスロットをリストアップし，依存関係を導出するのかについては考慮されていない．たとえば，問題領域の分析モデルから，本稿で提案したパターン記述に変換する方法が必要となるであろう．また，パターン記述をシグニチャとした部品検索の実現法，部品の振舞いの厳密な記述法についても今後さらに研究を進める必要がある．

参考文献

1) Alexander, C., Ishikawa, S. and Silverstein, M.: *A Pattern Language*, Oxford University Press (1977).
2) Coad, P., North, D. and Mayfield, M.: *Object Models: Strategies, Patterns, and Applications*, Prentice-Hall (1995).
3) Dean, T.R. and Cody, J.R.: A Syntactic Theory of Software Architecture, *IEEE Trans. SE*, Vol.21, No.4, pp.302–313 (1995).
4) Buschmann, F., Meunier, R., Rohnert, H. and Sommerland, M.P.: *Pattern-Oriented Software Architecture A System of Patterns*, Wiley (1996).
5) Feiler, J. and Meadow, A.: *Essential OpenDoc*, Addison-Wesley (1996).
6) Gamma, E., Helm, R., Johnson, R. and Vlissides, J.: *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley (1995).
7) Johnson, R.E.: Documenting Frameworks using Patterns, *Proc. OOPSLA '92, ACM SIGPLAN Notices*, pp.63–76 (1992).
8) Jun-Jang Jeng, B.H.C.: Specification Matching for Software Reuse: A Foundation, *Proc. ACM Symposium on Software Reuse*, pp.97–105 (1995).
9) Kiczales, G., et al.: Aspect-Oriented Prigramming, Technical Report, Xerox PARC (1996).
10) Nierstrasz, O. and Meijler, T.D.: Requirements for a Composition Language, *Object Based Models and Languages for Concurrent Systems*, pp.147–161, Springer-Verlag (1995).
11) Nierstrasz, O., Schneider, J.-G. and Lumpe, M.: Formalizing Composable Software Systems – A Research Agenda, *1st IPIF Workshop on Formal Methods for Open Object-based Distrubuted Systems*, pp.271–282 (1996).
12) Pree, W.: *Design Patterns for Object-Oriented Software Development*, Addison-Wesley (1994). 佐藤啓太，金澤典子（訳）：デザインパターンプログラミング，トッパン (1996).
13) Tanaka, Y.: A Meme Media Architecture for Fine-Grain Component Software, *2nd International Symposium on Object Technologies for Advanced Software*, Kanazawa, Japan (1996).
14) Udell, J.: Componentware, *Byte*, Vol.19, No.5, pp.46–56 (1994).
15) Zaremski, A.M. and Wing, J.M.: Signature Matching, a Tool for Using Software Libraries, *ACM Trans. Software Engineering and Methodology*, Vol.4, No.2 (1995).
16) Zaremski, A.M. and Wing, J.M.: Specifica-

tion Matching of Software Components, *ACM Trans. Software Engineering and Methodology*, Vol.6, No.4, pp.333–369 (1997).
17) 青山幹雄：コンポーネントウェア：部品組み立て型ソフトウェア開発技術，情報処理，Vol.37, No.1, pp.71–79 (1996).
18) 長崎 祥，田中 譲：シンセティック・メディアシステム：IntelligentPad，コンピュータソフトウェア，Vol.11, No.1, pp.36–48 (1994).
19) 平野亮太，田中 譲：IntellgentPad における部品設計パターン，ソフトウェア科学会第 13 回全国大会論文集，pp.185–188 (1996).

(平成 10 年 6 月 15 日受付)
(平成 11 年 3 月 5 日採録)



平野 亮太（正会員）

1971 年生．1994 年北海道大学工学部電気工学科卒業．1996 年同大学院電気工学専攻修士課程修了．現在，同大学院工学研究科電子情報工学専攻博士後期課程に在学中．ソフトウェア開発手法，ソフトウェア再利用技術等の研究に従事．ソフトウェア科学会会員．

田中 譲（正会員）

1972 年京都大学工学部電気工学科卒業．1974 年同大学院電子工学専攻修士課程修了．工学博士．1974 年北海道大学工学部助手．1977 年同講師．1985 年同助教授を経て，1990 年同教授，現在に至る．1996 年北海道大学知識メディアラボラトリー長．この間，1985 年 10 月より 1 年間，IBM 社 T.J. ワトソン研究所客員研究員．ソフトウェア科学会，人工知能学会，米国 IEEE 各会員．データベース理論，データベース・マシン，並列処理アーキテクチャ，メディア・アーキテクチャ等の研究に従事．「コンピュータ・アーキテクチャ」（雨宮氏との共著，オーム社）等の著書あり．1994 年に IntelligentPad の開発に関して日経 BP 技術賞大賞受賞．