アーキテクチャパターンの構造化に基づくパターン候補の導出

中野 聡之† 中野 由貴† 鷲崎 弘宜† 深澤 良彰†

ソフトウェアアーキテクチャの設計においてアーキテクチャパターンを適用することで、品質要求や制約を満たすアーキテクチャを効率よく導出することができる。しかしながら、開発状況の軽微な相違や設計トレードオフから、既存のアーキテクチャパターンの適用が困難となる問題がある。そこで我々は、実現手法や品質要求が既存のパターンとは異なる複数のパターン候補を導出することで、アーキテクチャパターンの適用を支援する手法を提案する。

Deriving Candidate Patterns based on Structured Architecture Pattern

Toshiyuki Nakano† Yuki Nakano†
Hironori Washizaki† and Yoshiaki Fukazawa†

Applying architectural pattern can derive suited architecture efficiently at the software architecture design. However, it is often difficult to apply existed architectural patterns complied with developmental status and design trade-off. So we propose the method that support applying architectural pattern by deriving candidate architectural patterns differed from existed architectural pattern.

1. はじめに

アーキテクチャパターンとは、ソフトウェアシステムの構造組織化スキーマを表現するものであり、少数の主要な特徴と、それらを組み合わせてアーキテクチャの完全性を保持するための制約により構成されている[1]。ソフトウェアアーキテクチャの設計におけるアーキテクチャパターンの適用により、システムの有益なイメージや設計の指針を得ることできる。したがって、システムの品質要求や制約に適合するアーキテクチャパターンを選択することにより、品質や性能の保たれたソフトウェアを効率良く開発することができる。しかしながら、パターンには複数の実現手法が含まれており、それぞれが異なる制約や品質特性に対して影響を持つ。そのため、実際の開発環境との軽微な状況の相違や求められる品質要求の差異から、既存のアーキテクチャパターンの適用が困難となってしまう場合がある。

我々は、既存のアーキテクチャパターンを構造化（構造に関するモデル化）した上で構造を変化させることにより、実現手法や品質要求の異なる複数のパターン候補を導出する手法を提案する。開発状況や品質要求の軽微な相違により既存のパターンを適用できない場合において、複数の類似パターンを提示することによるアーキテクチャパターンの適用支援を目的とする。

2. アーキテクチャパターン適用の課題

2.1. ソフトウェアアーキテクチャ

ソフトウェアアーキテクチャは、システムに関する最も初期の設計方針を明示したものであり、システムの構造や要素間の連携を表現する。初期の決定であるため修正することが困難であり、後の開発プロセスで変更することも容易ではない。したがって、ここでの決定事項がシステムの残りの開発やその配置、および保守期間に対して重要な役割を果たす。また、システムの共通な抽象概念を表現しているため、ソフトウェアシステムの顧客、ユーザ、設計者、プログラマ等の利害関係者に対し、交渉や相互理解といったコミュニケーションの基礎として使用できる[2]。

2.2. アーキテクチャパターン

ソフトウェア開発におけるパターンとは、開発上で繰り返し現れる問題から得られる知識を抽象化、形式化することで再利用可能な形としてまとめ、記述されたものである[3]。パターンは主として文脈、問題、解法から構成されており、パターンが解決を与える問題やその問題が発生する状況、具体的な問題の解法や解決の指針について明記されている[4]。

アーキテクチャパターンとは、アーキテクチャの設計段階において適用されるソフトウェアパターンであり、一連の要素タイプとそれらの相互関係を表すレイアウト、および一連の意味上の制約により構成される。アーキテクチャパターンの適用により、レイアウトを通してシステムの有益なイメージを伝え、システムに有益な制約を課すことで設計の指針を効率よく得ることができる。アーキテクチャパターンの文書、および静的モデルの具体例を以下の図 1 に示す。このパターンは、Web アプリケーション開発の分野で用いられるアーキテクチャパターンであり、バックエンドおよびデータベースにリアルタイムのアクセスを提供することに適している。

適用するアーキテクチャパターンを選択する判断基準として、パターンが影響を与える品質特性が用いられる。ソフトウェアに求められる品質特性を持つアーキテクチ

†早稲田大学理工学術院基幹理工学研究科
Graduate school of Fundamental Science and Engineering，Faculty of Science and Engineering，Waseda University.

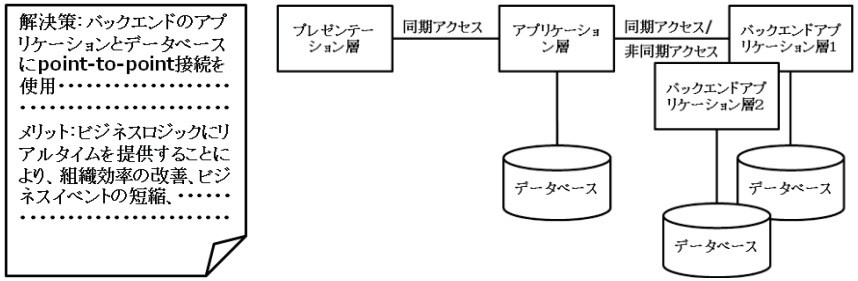

図 1. 直接統合単一チャネルパターン

ャパターンを適用することで、高品質、高信頼なソフトウェアの開発に繋がる。したがって、アーキテクチャパターンを選択することが、アーキテクチャ設計における最初の重要な設計選択となる。

2.3. アーキテクチャパターンの適用における問題点

どのようなパターンにも複数の実現手法が含まれており、それらは多くの場合異なる品質特性に影響を与える。しかしながら、実際に開発するシステムにおいて、パターンが影響を与える全ての品質特性が必要とされるとは限らない。また、実現手法の相互関係から生じる設計トレードオフ等の制約が、実際の開発状況に適さないために既存のパターンを適用できない場合がある。このように、求められる品質特性や開発状況の軽微な相違により、既存のアーキテクチャパターンの適用が困難、あるいは適用不可能となってしまう問題がある。

この問題に対し現状では、適用可能な他のパターンを探索することや、パターンマイニングにより新たなパターンの抽出を行うことで対処される。パターンマイニングとは、特定の範囲の具体的なプロダクト群や開発経験から、ノウハウや定石をパターンとして抽出する行為である[4][5]。しかし、パターンマイニングの従来手法はインタビューやワークショップにより、熟練者の実際の経験からパターンを抽出することが主である。したがって、従来手法により抽出されるパターンは開発状況や適用対象が限定的となってしまうため、上記の問題の本質的な解決にはならない。

3. 構造化に基づくパターン候補導出

我々は、既存のアーキテクチャパターンを構造化し、得られるパターン構造（以降、パターングラフと称する）を変化させることにより、性質の異なるパターングラフ（以降、パターン候補と称する）を導出する手法（以降、本手法）を提案する。パターンの構造化により、パターンに含まれる実現手法とその相互関係および影響する品質特性を明確化し、部分的に構造を変化させることで様々なバリエーションのパターン候補を導出する。こうして得られた数多くのパターン候補から特定の開発状況に合致するものを選択することで、アーキテクチャパターンの適用を支援することができる。

3.1. 概要

本手法は、既存のパターンを構造化するための「構造化ガイドライン」と、パターングラフを操作するための「グラフ操作システム」の 2 段階から構成される。本手法の全体像を図 2 に示す。

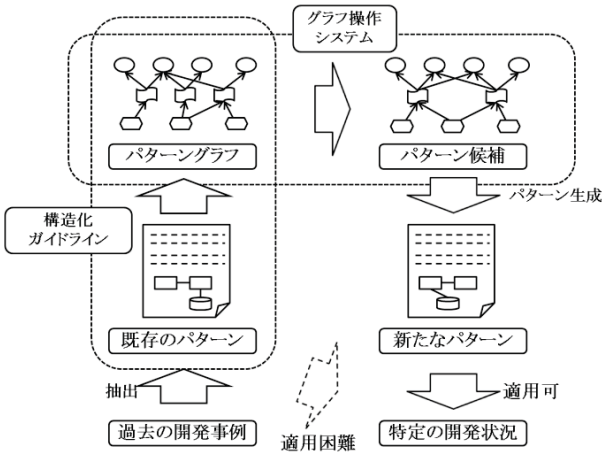

図 2. 本手法の全体像

まず、構造化ガイドラインを用いて既存のパターンを構造化する。パターンの文書を参照することでは、パターンに含まれる実現手法や影響する品質特性、およびそれらの相互関係を把握することは容易ではない。したがって、それらを要素として構造化することで、パターンの持つ性質および要素間の関係の把握が容易となる。また、自然言語記述であるパターンの文書に対し、各要素や要素間の関連の抽出をガイドラインに沿って行うことにより、いくらか属人性を排した構造化を行うことができる。

続いて、グラフ操作システムによりパターングラフの要素を部分的に操作することで、様々な性質のパターン候補を導出する。抽出したパターングラフの各要素は自然言語記述であるため、「HAZOP」の手法を利用し典型的な反意語を適用することで、機械的な要素の変化を実現した[6]。また、1 つの要素の変化が他の要素に与える影響を表現するために、影響度という値を各要素に与え、これを「ベイジアンネットワーク」の手法を用いて変化させる方法を実現した[7]。その結果、1 つのパターンから様々

な性質を持つ複数のパターン候補の導出に成功した。

以下、両手法の詳細について述べる。なお、本手法はパターンの文書から複数のパターン候補を導出するまでを対象とし、様々なパターン候補を提示することでアーキテクチャパターン適用の支援を可能とする。

3.2. 構造化ガイドライン

パターンの文書内から特定の要素を抽出し、有向エッジを用いて階層的に構造化することによりパターングラフを作成する。パターングラフを構成する各要素の名称、および役割の説明を以下の表 1 に示す。

表 1. パターングラフを構成する 3 要素

名称	要素の説明
実現手法	パターンの解決策を構成する具体的な実現方法
フォース	実現手法により与えられる影響や、実現手法の相互作用により発生する制約条件、トレードオフポイント等を示すもの
品質特性	機能性、保守性等のパターンが影響を与える品質

前述のように、パターンは複数の実現手法とそれらの相互作用により様々な品質特性に影響を与えている。したがって、上記の 3 要素を用いてパターングラフを表現することにより、各要素の関係性を容易に把握できることから、パターンの持つ構造的な側面を理解することができる[8]。その結果、パターンの持つ性質の理解のみならず、パターンを正しく適用できる状況などを明確に理解することができる[9]。

作成したガイドラインのおおまかな流れを以下の表 2 に示す（紙面の都合上、ガイドラインの全文については割愛する）。また、実際にガイドラインに基づいて抽出したパターングラフの具体例を以下の図 3 に示す。これは、図 1 で具体例として取り上げた直接統合単一チャネルパターンを構造化したパターングラフである。前述の通り、実現手法、フォース、品質特性の 3 要素でパターンの構造を表現している。品質特性の外形については、ポジティブなものとネガティブなもので表現の仕方を区別しており、要素内に D（ドメイン）や C（クラス）の記述を付加することで、品質特性が影響する分野や対象領域を明示している。また、「強く影響する」等のラベルや数値（影響度と称する）については次節において詳しく説明する。

表 2. ガイドラインの概要

各ステップの操作	操作の概要
1. 実現手法の列挙	文章内で用いられている具体的な実現手法を全て抽出する。　例. point-to-point 接続（図 1 より）
2. 下位フォースの抽出	実現手法により生じる具体的な影響や効果を抽出する。
3. 上位フォースの抽出	単一、または複数の下位フォースから生じる影響を抽出する。
4. メリット･デメリットの抽出	上位フォースにより生じるメリット、あるいはデメリットを抽出する。　例. 組織効率の改善（図 1 より）
5. 品質特性の特定	メリット・デメリットから影響する品質特性を特定する。
6. パターングラフの簡易化	メリット、デメリットおよび下位フォースを省略し、構成要素を前述の 3 要素に限定する。

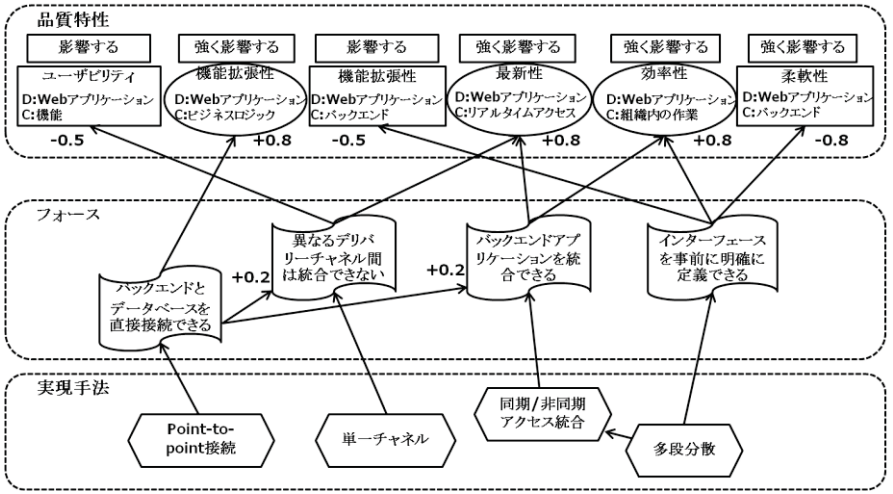

図 3. 構造化した直接統合単一チャネルパターン

3.3. グラフ操作システム

1 つのパターングラフから性質の異なる複数のパターン候補を導出するために、フォースを変化させる手法と要素間の影響の変化を表現する手法を考案し、それらを組み合わせることでグラフ操作システムを実現した。まず、パターンの持つ性質を変化させるために、与えられたパターングラフに含まれるフォースを変化させる。パターンにおいてフォースとは、実現手法やそれらの相互関係から生じる影響や制約を表現することで、パターンが影響を与える品質特性を特定するものである。したがって、フォースの変化により影響する実現手法や品質特性が変化することで、軽微な状況の相違や求められる品質特性の変化を表現することができる。続いて、パターングラフ内の 1 つのフォースを変化させた際の、パターングラフ全体への影響の変化を表現す

る。前述のように、フォースは実現手法の相互作用からなる影響や制約を表現するため、フォースの変化による影響は、変化させたフォースと直接関連する要素のみに留まらない。したがって、1 つのフォースの変化に伴うパターングラフ全体への影響を表現する必要がある。これら 2 つの操作を、パターングラフ内の全てのフォースに対して網羅的に適用することにより、1 つのパターングラフから複数のパターン候補を導出することができる。以降、各操作について具体例を交えた詳細な説明を行う。

(1) HAZOP を参考としたフォースの操作

自然言語で記述されているフォースの性質を変化させるために、一部の語を反意語に置き換える方法を用いる。しかしながら、反意語の解釈は人によって異なるため、属人性を排した機械的な操作を行うことは難しい。そこで我々は、ガイドワードを利用したリスク回避手法である HAZOP を本手法に応用することで、フォースを機械的に変化させる手法を実現した。ガイドワードとは、典型的な形容詞や副詞およびそれらの反意語をまとめた語のリストであり、ガイドワードと単語を組み合わせることにより、あらゆる状況を網羅的に想定することを目的とする手法が HAZOP である。HAZOP は主にリスク管理や安全性評価の分野で使用される手法であるが、しばしばソフトウェアアーキテクチャ設計の分野においても利用される[6]。

フォースを操作する具体的な手順について述べる。まず、フォースの記述を以下の表 3 に記載した 3 要素に分割する。このように分割することで、フォース内における反転操作（反意語の適用）可能な語を明示的に区別することができる。

表 3. フォースに含まれる 3 要素

名称	要素の説明	具体例
パラメータ	名詞や代名詞として扱われる語	ハードウェア、レイヤ
ワード	動詞として扱われる語	接続できる、定義できる
ガイドワード	修飾子のように扱われる語	早く、高く、直接

分割された各要素のうち、ワード、ガイドワードに対して反転操作を行う。このとき、1 つのフォース内に反転操作可能な要素が複数含まれている場合、各要素に対して操作を行った結果をそれぞれ保持する。操作の具体例を以下の図 4 に示す。ここで対象とするフォースは、3.2 節で示した直接統合単一チャネルパターンのパターングラフから得たものである。各フォースに含まれるワード、およびガイドワードに反意語を適用することにより、それぞれ異なる性質を持つフォースを得ることができる。図 4 において「バックエンドとデータベースを直接接続できる」というフォースでは、「直接」というガイドワード、および「接続できる」というワードに対してそれぞれ反意語を適用できる。したがって、このフォースから図 4 に示されている 2 種類のフォースを得ることができる。

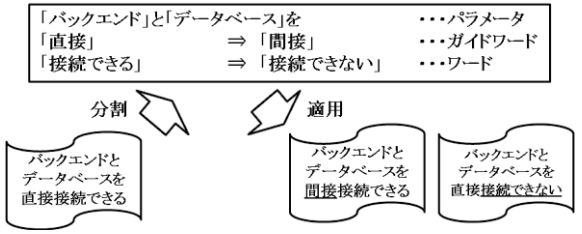

図 4. フォースの操作例

(2) ベイジアンネットワークに基づく影響度操作

前述のように、1 つのフォースの変化に伴うパターングラフ全体への影響を表現する必要があるため、要素間の値の推移を表現可能であるベイジアンネットワークを利用した。ベイジアンネットワークとは、グラフ全体の要素間における確率変数の値の推移を、確率伝播法に基づいて表現する手法である[7]。本手法では確率変数の値を影響度とすることで、フォースが品質特性に与える影響の大小、およびフォースの変化によるグラフ全体の要素への影響の推移を表現した。

影響度操作の詳細について述べる。まず、フォースから品質特性への影響度、およびフォースからフォースへの影響度を設定する。この設定はシステムの利用者が行うものであるが､連続的な値を利用者が判断して決めることは属人性が強く困難である。したがって、本手法では影響度の大小を表す離散的なラベルを設け、それに基づいた入出力を行えるものとした。影響度の値とラベルとの対応関係を以下の表 4 に示す。

表 4. ラベルと影響度の対応関係

ラベルの名称	入力時の影響度の値	値の範囲
強く影響する	+0.8、-0.8	± 0.7 ～ 1.0
影響する	+0.5、-0.5	± 0.31 ～ 0.69
弱く影響する	+0.2、-0.2	± 0 ～ 0.3

影響度が負の値となる場合は、パターンにおいてデメリットとなる品質特性に対してであり、フォースからフォースへの影響度やメリットとなる品質特性に対する影響度はすべて正の値となる。これらのラベルに基づき影響度を設定することにより、いくらか属人性を排した影響度操作を行うことができる。また、図 3 のように品質特性に対してラベルを付加することにより、パターンの品質特性に対する影響の度合いを明示することができる。

要素間の影響度を設定した上で、(1)で得たフォースをパターングラフ内のフォースと置き換える。ワードに対して反意語を適用させたフォースとの置き換えを行う場合、フォースの性質が反転されるため、影響度の値を正から負へと反転させる。ガイドワードに対して反意語を適用させたフォースとの置き換えを行う場合、フォースの性質の変化に即した値を入力者が与えるものとする。一例として、前例における「直接」を「間接」と置き換える場合においては、フォースによる影響が軽減されていると判断できるため、影響度の値を低減させるものとする。変化後の値に対してベイジアンネットワークを適用することで、グラフ内の全ての要素に対する影響度の値が変化し、品質特性への影響や実現手法が既存のパターンとは軽微に異なるパターン候補が生成される。この操作を(1)で得た全てのフォースに対して実行することにより、様々な性質のパターン候補を導出することができる。

実際に、直接統合単一チャネルパターンのパターングラフを操作した例を示す。図 3 において「異なるデリバリーチャネル間は統合できない」というフォースに対して置き換えを行い、得られたパターン候補を以下の図 5 に示す。

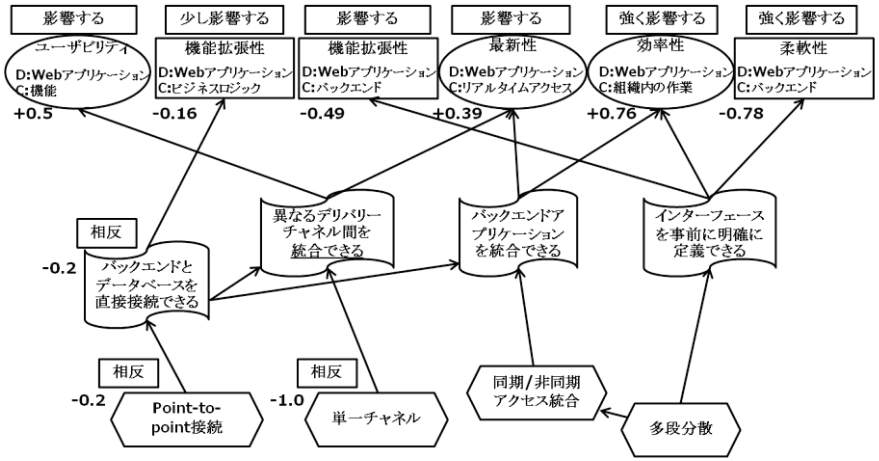

図 5. パターン候補の具体例

図 3 のパターングラフと比較すると、品質特性に対する影響度の値が変化していることから、品質特性への影響が異なるパターングラフであることが確認できる。また、一部のフォースや実現手法に付加されている「相反」というラベルについては、影響度の値が正から負へと反転した要素に対して与えたものであり、相反する要素と置き換える必要があることを示している。この例では、「point-to-point 接続」に対して「ハブアンドスポーク接続」を、「単一チャネル」に対して「複数チャネル」をそれぞれ置き換えることで、既存のパターンと異なる性質を持つパターン候補として完成されたものとなる。ここで、実在するアーキテクチャパターンである「スタンドアロンルータパターン」のパターングラフを以下の図 6 に示す。これは、図 3 と同様ガイドラインに沿ってパターンの文書から生成されたものである。

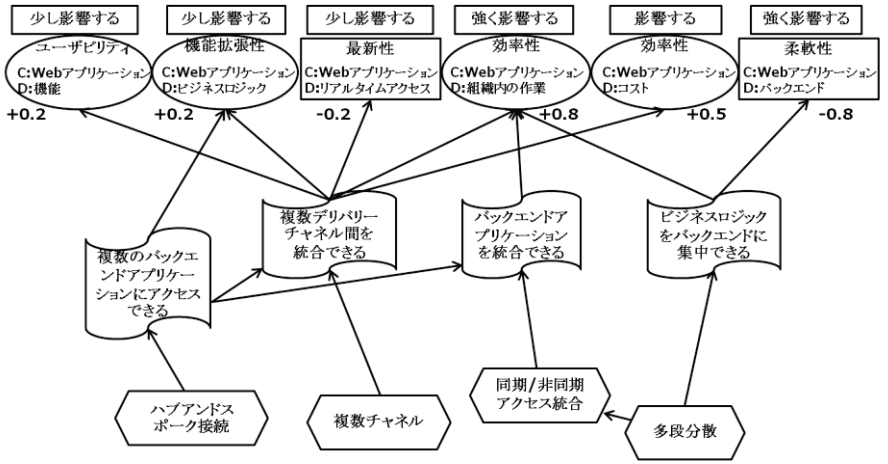

図 6. スタンドアロンルータパターンのパターングラフ

図 5 と図 6 を比較すると、細かな違いは見られるものの多くの類似点を確認することができるため、同一の性質を持つパターンであると言える。この結果から、既存のパターンの構造を変化させることで新たなパターンを導出できる可能性が示された。

4. 評価

4.1. 実験の方法

本手法の有用性を評価するために被験者実験を実施した。ソフトウェア工学を専攻する理工学生 2 名に同一のパターン文書を与え、一方は本手法を用いてパターン候補を導出し、他方は独自にパターン候補を提示する実験を行った。対象とするパターン文書は、具体例として取り上げた「直接統合単一チャネルパターン」を用いた。本手法を用いる被験者には前述の構造化ガイドライン、グラフ操作システムおよび各手法の説明を記述した用紙を与え、本手法を使用しない被験者に対しては、パターン候補についての説明を口頭で行うのみとした。

4.2. 結果と考察

本手法を用いた被験者は、2 時間程度で 2 種類のパターン候補の導出に成功した。導出したパターン候補のうち、一方を以下の図 7 に示す。

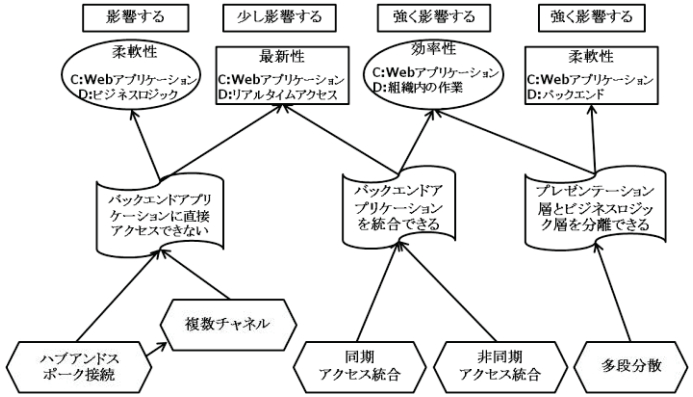

図 7. 被験者により導出されたパターン候補

この図から、図 6 に示したスタンドアロンルータパターンに近い性質を持つパターン候補を導出できたことが確認できる。構造にいくらか違いは見受けられるが、それぞれの要素に多くの共通点を見出すことができる。この結果から、ある程度の属人性は伴うものの、有用性のあるパターン候補を導出することができたと言える。また、他方のパターン候補は図 7 で示されているフォース「プレゼンテーション層とビジネスロジック層を分離できる」に対して反転操作を行ったものである。

本手法を使用しない被験者は同程度の時間を要したものの、パターン候補の可能性を示す抽象的な記述を得るにとどまり、具体的なパターン候補の提示には至らなかった。原因として、パターン候補の提示という目的が困難であった点のほか、パターンの持つ構造的な側面の理解に多くの時間を要してしまった点が挙げられる。

以上の結果から、本手法がある程度属人性を排したものであり、妥当性や実用性のほか、本手法を用いることによるパターンに対する理解容易性の向上が示された。

5. 関連研究

Weiss らは、ゴール指向要求言語（GRL）を用いてアーキテクチャパターンを形式化する手法を提案している[8]。GRL を用いてパターンに含まれるフォースやそれらの相互作用を形式的に表記することで、パターンの評価方法や適用する際の選択基準を明確化できる。本手法では、構造を変化させやすい形式を用いた構造化を行うことで、パターンに含まれる実現手法や影響する品質特性、およびそれらの相互関係を明確化するのみならず、グラフ操作により複数のパターン候補を導出することができる。

6. おわりに

アーキテクチャパターンの適用において、品質要求や開発状況の軽微な相違により既存のパターンを適用することが困難となる問題がある。これに対し我々は、既存のパターンを構造化した上で構造を変化させることにより、性質の異なる複数のパターン候補を導出する手法を提案した。本手法により、品質特性に対する影響や使用する実現手法が軽微に異なる複数のパターン候補を示すことで、アーキテクチャパターンの適用を支援することができる。

今後の課題として、本手法における成果物をより実用的なものとするために、パターン候補からパターン文書を生成する手法を考案する必要がある。また、本手法により得られたパターン候補の有用性を検証する方法についても考慮したい。

参考文献

1) Frank Buschmann, Regine Meunier et al. 著, 金澤 典子, 水野 貴之, 桜井 麻里 略, “PATTERN-ORIENTED SOFTWARE ARCHITECTURE”, 近代科学社, 2001

2) Len Bass, Paul Clements, Rick Kazman 著, 前田 卓雄, 佐々木 明博, 加藤 滋郎ほか 略, “実践ソフトウェアアーキテクチャ”, 日刊工業新聞社, 2005

3) Christopher Alexander, “Timeless Way of Building”, Oxford University Press, 1979

4) 久保 淳人, 鷲崎 弘宜, 深澤 良彰, “パターンマイニングによるソフトウェア要求の獲得知識の記述”, 第12回ソフトウェア工学の基礎ワークショップ(FOSE2005), pp.183-188, 2005

5) 鷲崎 弘宜, “協調作業型のパターンマイニング・ワークショップ”, ウィンターワークショップ2010・イン・倉敷 論文集, pp.91-92, 2010

6) Klaus Marius Hansen, Lisa Wells et al. “HAZOP Analysis of UML-Based Software Architecture Descriptions of Safety-Critical Systems”, Proceedings of NWUML 2004

7) 風戸 広史, 林晋平ほか, “ベイジアンネットワークを用いたソフトウェア実装技術の選択支援”, 情報処理学会論文誌, Vol.51, No.9, pp.1765-1776, 2010.

8) Gunter Mussbacher, Michael Weiss, Daniel Amyot, “Formalizing Architectural Patterns with the Goal-oriented Requirement Language”, Conference on Pattern Languages of Programs (PLoP2006), 2006

9) Ivan Araujo, Michael Weiss “Linking Patterns and Non-Functional Requirement”, Conference on Pattern Languages of Programs (PLoP2002), 2002