情報処理学会論文誌　Vol.58　No.4　885–897 (Apr. 2017)

多層システムに対する横断的な機能捜索

風戸 広史[1,a)]　林 晋平[2]　大島 剛志[3]　小林 隆志[2]　夏川 勝行[3]　星野 隆[3]　佐伯 元司[2]

受付日 2016年8月8日, 採録日 2017年1月10日

概要：複数のレイヤで構成されたソフトウェアでは，レイヤ間に分散したプログラム要素が協調動作して1つの機能を実現するために，機能とプログラム要素群を対応付ける作業である機能捜索が難しい．そこで，本論文では機能とレイヤ間に分散したプログラム要素群の対応関係を半自動的に抽出する機能捜索手法を提案する．提案手法ではレイヤごとの実行プロファイルを併合して形式概念分析への入力として用いることにより，異なるレイヤに属するプログラム要素を形式概念としてグループ化し，機能の集合と対応付ける．たとえば，提案手法をWebアプリケーションに対して適用することにより，アプリケーション層に属するモジュールだけでなく，プレゼンテーション層やデータ層に属するWebページやデータベースのテーブルアクセスといった要素を同時に機能に紐付けられる．Webアプリケーションの例題に提案手法を適用し，3つのレイヤに分散したプログラム要素と機能の対応関係を抽出した事例を示すことにより，手法の実現可能性を示すとともに，現実的なプログラム理解の支援に向けた応用可能性について議論する．

キーワード：機能捜索，多層システム，動的解析，形式概念分析

Cross-layer Feature Location

Hiroshi Kazato[1,a)]　Shinpei Hayashi[2]　Tsuyoshi Oshima[3]　Takashi Kobayashi[2]
Katsuyuki Natsukawa[3]　Takashi Hoshino[3]　Motoshi Saeki[2]

Received: August 8, 2016, Accepted: January 10, 2017

Abstract: In multi-layer systems such as web applications, locating features is a challenging problem because each feature is often realized through a collaboration of program elements belonging to different layers. This paper proposes a semi-automatic technique to extract correspondence between features and program elements among layers, by merging execution traces of every layer to feed into formal concept analysis. By applying this technique to a web application, not only modules in the application layer but also web pages in the presentation layer and table accesses in the data layer can be associated with features at once. To show the feasibility of our technique, we applied it to a web application which conforms to the typical three-layer architecture of Java EE and discuss its applicability to other layer systems in the real world.

Keywords: feature location, multi-layer system, dynamic analysis, formal concept analysis

1.　はじめに

ソフトウェアが提供する機能（feature）[2] とソースコード中のプログラム要素の対応関係を特定することは，プログラムを変更する前段階の作業として頻繁に行われている．この作業は機能捜索（feature location）もしくは概念捜索（concept location）[3], [4][*1]と呼ばれ，これまでに静的解析，動的解析，情報検索などの技術を応用した機能捜索の支援手法（以下，機能捜索手法）が数多く提案されて

1　株式会社 NTT データ
NTT DATA CORPORATION, Tokyo 135–8671, Japan
2　東京工業大学
Tokyo Institute of Technology, Tokyo 152–8552, Japan
3　日本電信電話株式会社ソフトウェアイノベーションセンタ
NTT Software Innovation Center, Tokyo 108–0075, Japan
a)　kazatoh@nttdata.co.jp

本論文は，CSMR 2012 での我々の発表 [1] の内容を発展させたものである．

*1　概念（concept）という用語は形式概念分析と概念捜索の両方の文脈で使用される．混乱を避けるため，本論文では，捜索対象を機能，また手法を機能捜索として定義し，以降では概念を捜索対象としては扱わない．本論文の以降で概念とあった場合，これは形式概念分析における形式概念を指す．

いる [5]（詳細は 7 章）．

これらの機能捜索手法を，Web アプリケーションをはじめとする，複数のレイヤで構成された多層システムに適用することを考える．エンタープライズ分野では多層アーキテクチャ [6] が主流となっており，多くの企業は継続的に保守すべき多層システムをかかえている．これらの多層システムでは各レイヤをそれに適した記述言語や実行基盤を用いて実装し，複数のレイヤが協調動作して 1 つの機能を実現しているため，その保守では，複数のレイヤに横断した機能捜索が必要である [7], [8]．たとえば，Web アプリケーションの機能は，Web ブラウザが実行する Web ページへのアクセスに始まり，アプリケーションサーバ内のプログラムの実行を経て，データベース管理システム内のテーブルの読み書きを行うといった一連の流れによって実現される．この際には，プログラムのモジュールだけでなく，関連している Web ページやテーブルアクセスなども一種のプログラム要素として機能するため，これらも機能捜索により特定できると都合が良い．

しかし，多くの機能捜索手法では，オブジェクト指向言語のメソッドや手続き型言語の関数などのモジュールを機能の構成単位とし，機能捜索の結果としてこれらのような単一のプログラミング言語で定義される要素の集合を抽出する．前述したような複数の言語や実行基盤を併用する多層システムでは，このような単一のレイヤに属するプログラム要素のみを抽出する機能捜索では不十分である．各レイヤでは異なる種類のプログラム要素が扱われるため，既存の機能捜索手法を適用する場合には機能の実装箇所として捜索を受けないレイヤが発生するという問題点がある．レイヤごとに繰り返し既存手法を適用し，それらの結果を統合することによってすべてのレイヤが捜索可能であるが，この繰返しに手作業の部分を含むと，レイヤ数が増加するほど多くの労力を必要とする．各レイヤは情報隠蔽原則 [9] に従って密結合を避けるよう構成されるため，機能を塊として発見しがたいことや，関連している実装技術がレイヤごとに異なり統一的なプログラム解析が実現しがたいことなども，こういった解析を難しくしている．さらに，システムによって扱うレイヤの数や役割，実装に用いる技術といった構成が異なることもあることから，なるべく特定のレイヤに依存しないような解析手法が望まれる．

本論文ではレイヤごとに異なる記述言語を用いて開発された多層システムにおいて，機能と各レイヤに分散したプログラム要素群の間の対応関係を抽出する機能捜索手法を提案する．提案手法ではまず，機能を実行するための複数のシナリオを用意し，シナリオの実行中に動作したプログラム要素の対応関係を動的解析でレイヤごとに取得する．次に，それら対応関係を 1 つの形式文脈として併合して形式概念分析を適用し，シナリオの部分集合とプログラム要素の部分集合を形式概念として抽出する．この形式概念に含まれるプログラム要素間の依存関係を分析することにより，機能を実装しているプログラム要素の集合の抽出が可能となる．提案手法は，レイヤごとの独立した動的解析結果を用いるため，対象システムを動作させることさえできれば導入が容易であるという利点がある．また，分析対象のレイヤは自由に選択できるため，詳細な分析に先立ってプログラム要素数の少ないレイヤのみによる解析を行い，システムの概要を大雑把に把握するといった利用法も可能である．形式概念分析結果の理解のしやすさは分析結果の概念数に強く依存する [10] ことが知られており，たとえば 3 層 Web システムのアプリケーション層から得られる概念数が膨大になることは多いため，複雑なシステムを分析したい際にはこの特徴は利点につながる．

本論文の主要な貢献を以下に示す．

- Web アプリケーションなどのレイヤごとに異なる記述言語で開発された多層システムにおいて，機能に対応するプログラム要素群を複数のレイヤに横断的に抽出するための機能捜索手法を提案したこと．
- 実際に Web アプリケーションの例題に対して提案手法を適用することにより，機能に対応する Web ページやテーブルアクセスなどの要素を特定できること，また分析が困難な規模のシステムに対して分析対象のレイヤの調整が有効であることを確かめたこと．
- 提案手法の現実的な有効性に関する議論を与えたこと．

本論文の以降の構成を以下に示す．まず，2 章で準備として提案手法の説明に必要となる形式概念分析を導入し，それを用いた既存の機能捜索手法を紹介する．3 章で提案手法について述べ，4 章ではその実現法を与える．5 章で予備評価について述べ，6 章で提案手法の有効性を議論する．7 章で関連研究を示し，最後に，8 章で本論文の結論を述べ，今後の展望を示す．

2. 準備

文献 [11] などにも詳しく，よく知られたことであるが，提案手法の説明で必要となるため，本章で形式概念分析を簡潔に導入しておく．また，既存の形式概念分析に基づく機能捜索手法を紹介する．

2.1 形式概念分析

形式概念分析 [11] はデータ分析手法の一種であり，束論に基づいて二項関係から概念構造を抽出することを特徴とする．

$\mathcal{O}$ がオブジェクトの集合，$\mathcal{A}$ が属性の集合であり，$\mathcal{I}$ が二項関係 $\mathcal{I} \subseteq \mathcal{O} \times \mathcal{A}$ であるとき，三つ組 $C = (\mathcal{O}, \mathcal{A}, \mathcal{I})$ を形式文脈（formal context）という．形式文脈は文脈表と呼ばれる $|\mathcal{O}|$ 行 $|\mathcal{A}|$ 列の表を用いて，$(o_i, a_j) \in \mathcal{I}$ のとき i 行 j 列のセルに $\times$ を記入することにより表現できる．

オブジェクトの集合 $O \subseteq \mathcal{O}$ に対して O が共通に持つ属

性の集合 $\sigma(O)$ を

$$\sigma(O) = \{\, a \in A \mid (o, a) \in \mathcal{I} \text{ for all } o \in O \,\}$$

とおく．同様に，属性の集合 $A \subseteq \mathcal{A}$ に対して A をすべて持っているオブジェクトの集合 $\gamma(A)$ を

$$\gamma(A) = \{\, o \in O \mid (o, a) \in \mathcal{I} \text{ for all } a \in A \,\}$$

とおく．$O \subseteq \mathcal{O}$, $A \subseteq \mathcal{A}$ に対して $A = \sigma(O)$ かつ $O = \gamma(A)$ を満たすとき，かつそのときに限り，対 $c = (O, A)$ が形式概念（formal concept）であるという．形式概念では O に属するすべてのオブジェクトに共通する属性の集合が A に一致し，また A に属するすべての属性を持つオブジェクトの集合が O に一致する．O を c の外延（extent），A を c の内包（intent）と呼ぶ．

$c_1 = (O_1, A_1)$，$c_2 = (O_2, A_2)$ がそれぞれ形式概念のとき，$O_1 \subseteq O_2$ と $A_1 \supseteq A_2$ は同値であり，形式概念間の半順序関係 $\leq$ を以下のように定義できる．

$$(O_1, A_1) \leq (O_2, A_2) \iff O_1 \subseteq O_2 \iff A_1 \supseteq A_2$$

c_2 を c_1 の上位概念，c_1 を c_2 の下位概念と呼ぶ．ある形式文脈 $\mathcal{I}$ に関するすべての形式概念の集合 $\mathcal{L}(\mathcal{I})$ は，半順序関係 $\leq$ に関して概念束（concept lattice）と呼ばれる完備束を形成する．

形式概念分析はデータの分析に広く利用されている．また，形式文脈から概念束を計算するための多数のアルゴリズムやツールも多数提案されている [11], [12].

2.2　形式概念分析を用いた機能捜索

Eisenbarth らや Koschke らは形式概念分析と動的解析，静的解析を組み合わせて機能とプログラム要素の対応関係を得る機能捜索手法を提案している [10], [13]．この手法では，シナリオと実行プロファイル中に出現するプログラム要素の関係に形式概念分析を適用して概念束を得たあとに，分析者が把握しているシナリオと機能の対応関係を用いて概念束を分析する．

たとえば，シナリオ $S = \{s_1, s_2, s_3\}$ をオブジェクト，シナリオ中に実行されたプログラム要素 $M = \{m_1, \ldots, m_7\}$ を属性とした形式文脈を用いて，形式概念分析を適用した場合を考える．**表 1** は，形式文脈を表した文脈表である．表の行はシナリオ，列はプログラム要素を表しており，たとえば，シナリオ s_1 によりプログラム要素 m_1，m_4，m_6，m_7 が実行されたことが示されている．この表には，各シナリオによってどの機能 $F = \{f_1, f_2, f_3\}$ を実行しているか，という分析者の意図を表す対応関係も含まれている．たとえば，シナリオ s_1 は機能 f_1 および f_3 を実行している．この形式文脈を入力として形式概念分析を適用することにより，概念束を得る．得られた概念束は，ハッセ図の形で**図 1** に示すように可視化できる[*2]．図の各節は形式概念を表しており，それぞれがシナリオ，プログラム要素と機能の部分集合を含む．たとえば f_1 は，s_1，s_2 をそれぞれ外延に含む c_4，c_5 の上位概念であり，かつ s_3 を外延に含む c_6 の上位概念にはあたらない，c_1 の内包に自動的に分類される．このとき，f_1 とともに c_1 の内包に分類された m_4 は，s_1，s_2 の両方で実行されたが，s_3 では実行されなかったプログラム要素であるため，f_1 の実現に関わるプログラム要素として最有力であると判断する．また，下位概念の c_4 や c_5 も，f_1 を含むシナリオ s_1，s_2 のそれぞれ一方のみで実行され，f_1 を含まないシナリオ s_3 で実行されなかったプログラム要素を内包に持つため，部分的に f_1 と関連があると判断する．このことから，f_1 の実現に関わるプログラム要素として m_4 が最有力であり，m_1 や m_2 も有力であると考えることができる．

表 1　文脈表の例
Table 1　Example of a concept table.

	f_1	f_2	f_3	m_1	m_2	m_3	m_4	m_5	m_6	m_7
s_1	×		×	×			×		×	×
s_2	×	×			×		×	×		×
s_3		×	×			×		×	×	×

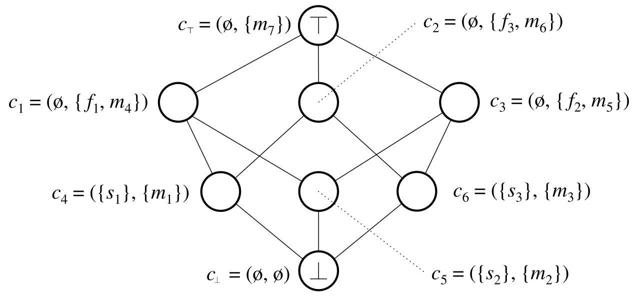

図 1　図 1 の文脈表に対応する概念束
Fig. 1　Concept lattice for formal context in Table 1.

このように形式概念分析を活用することの利点には，複数の機能を同時に分析できる点，部分的な関連性も含めて分析できる点がある．たとえば，図 1 の例では，単一のシナリオ集合から f_1，f_2，f_3 の全機能に対して捜索を行い，その結果の全体像を俯瞰することを可能としている．また，機能と同一の形式概念に分類された，強い関連性を持つプログラム要素だけでなく，上位概念や下位概念などに分類された部分的な関連性についても特定することができる．さらに，形式概念分析の適用は，単純に行えば煩雑

[*2] 我々の手法は希薄（sparse）な概念束表現 [13] における内包を扱う．すなわち，ある形式概念 $c = (O, A)$ に注目したとき，c の内包として，c の上位概念における内包の要素を省いた A の部分集合 A' を用いる．同様に，c の外延として，c の下位概念における外延の要素を省いた O の部分集合 O' を用いる．これは，提案手法では内包に含まれる機能やプログラム要素のうち，上位概念には含まれないものの間に最も強い関連があると考えるからである．説明を簡単とするため，以降ではこの O'，A' を単に c の外延，内包として扱う．

になりがちな動的解析結果の分析の繰返しを，既存の理論体系に従い系統立てて行った結果であり，分析に既存の形式概念分析のツール基盤を利用できるという副次的な利点もある．

この手法では，オブジェクト指向言語のメソッドや手続き型言語の関数などのモジュールを機能の構成単位とし，それらがシナリオの実行時に動作するかどうかを動的解析で観測することによって機能捜索の結果を得る．そのため，多層システムにおけるレイヤ間で記述言語が異なる場合には観測されないレイヤが発生し，機能の捜索結果として当該レイヤに属するプログラム要素を提示できない．レイヤ間でプログラム要素の種類は異なる場合が多いため，既存の機能捜索手法を適用するためにはレイヤごとに繰り返し手法を適用してそれらの結果を統合する必要がある．その場合，繰返しのたびに形式概念の内包としてグループ化されたプログラム要素群を分析する作業を含むため，レイヤ数が増加するほど多くの労力を必要とする．

3. 提案手法

本章では多層システムにおいてレイヤごとに異なる記述言語がソースコードの記述に用いられる場合を考慮した，形式概念分析に基づく機能捜索手法を提案する．提案手法では多層システムを複数のシナリオで実行し，機能の部分集合と全レイヤにわたるプログラム要素の部分集合を形式概念としてグループ化するとともに，シナリオにおけるそれらのグループの出現に関する共通性や可変性を概念束として出力する．

提案手法の前提として，分析者は分析対象となる多層システムに複数の機能が存在することと，それらの機能を実行させるためのシステムの操作手順を知っていることが必要である．分析者は各機能を実装しているソースコードのファイルやファイル内のプログラム要素の場所を把握しておらず，それらを知ることが目的である．このような状況は，システムの開発時と保守時で担当する開発者や組織が異なり，信頼できる設計ドキュメントが存在しない場合に発生する．そのような場合でも，システムのユーザにヒアリングしたり，利用マニュアルを参照したりすることによって機能の存在とその実行手順は把握できるものと考える．

手法の全体像を**図 2** に示す．提案手法の基本的な枠組みは既存手法 [10], [13] と同様である．ただし，分析結果として得られるプログラム要素が，一般的なプログラム言語におけるモジュールだけではなく，他のレイヤのソースコードを記述するための言語の要素も含めて扱っていることが大きな違いである．この図では，プレゼンテーション層，アプリケーション層，データ層の 3 レイヤからなる Web アプリケーションの解析を想定している．プレゼンテーション層では，アクセスする Web ページがプログラム要素となり，たとえば `/index.shtml` が該当する．アプリケーション層では，サーバプログラムのメソッドがプログラム要素となり，たとえば `CatalogService.init()` が該当する．データ層では，関係データベースのテーブルへのアクセスがプログラム要素となり，たとえば `PRODUCT` テーブルの読み込みが該当する．提案手法は大きく（1）シナリオの準備，（2）レイヤごとの実行プロファイルの取得，（3）形式概念分析の適用の 3 ステップからなる．これらを行うことにより，結果として概念束を得る．得られた概念束の分析により，レイヤを横断したプログラム要素群が機能と関連付けられた形で取得できる．以下ではこれらのステップを順に説明する．

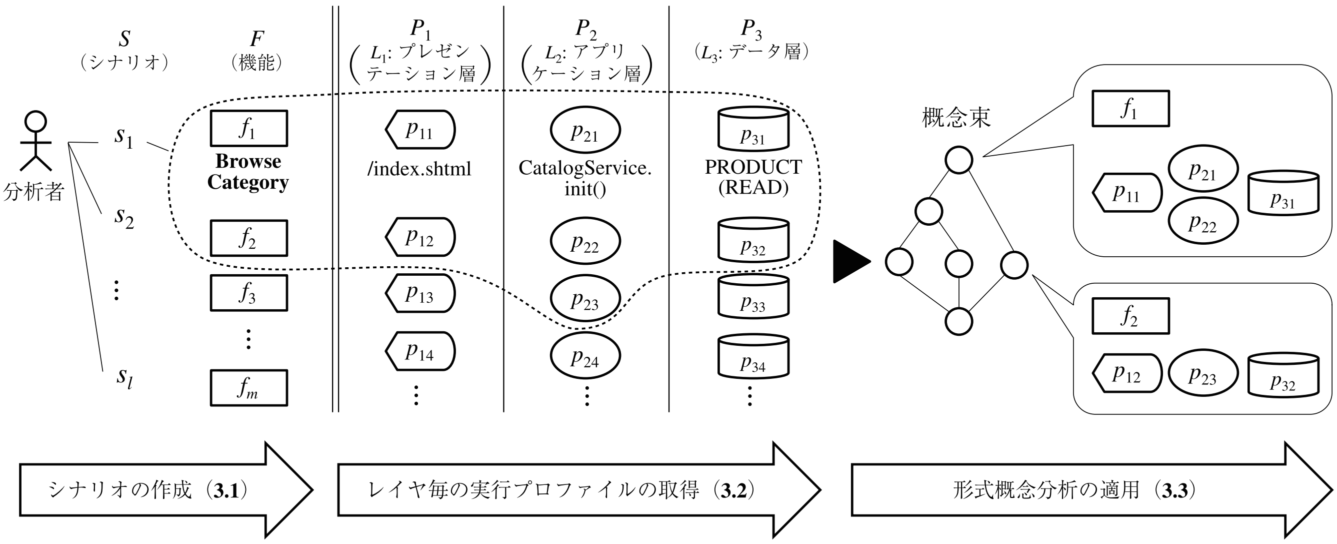

図 2　提案手法の全体像
Fig. 2　Overview of our technique.

3.1 シナリオの準備

分析者は提案手法の入力として，機能を実行するための一連のシナリオ群を準備する．この作業は提案手法がベースとする形式概念を用いた機能捜索手法 [10], [13] とまったく同様であり，想定する機能の集合 $F = \{f_1, f_2, \ldots, f_m\}$ 中の任意の機能 $f \in F$ に対して，f を実行するシナリオ，実行しないシナリオが少なくとも 1 つずつ存在するように複数のシナリオを作成する．作成した l 個のシナリオの集合を $S = \{s_1, s_2, \ldots, s_l\}$ とし，文脈表，すなわちシナリオと機能の対応関係 $R_0 : S \times F$ を分析者のドメイン知識に基づいて定義する．

機能捜索を行ううえでは異なる機能をそれぞれ異なる形式概念に分類することが望ましいが，上記のシナリオの準備方法では，必ずしもそのような性質が得られないことには注意が必要である．希薄な概念束表現において，任意の異なる 2 つの機能 $f_1, f_2 \in F$ がそれぞれ異なる形式概念 c_1，c_2 に分類されるためには，各機能を実行するシナリオの集合が互いに異なること，すわなち $f_1 \neq f_2 \Longrightarrow S(f_1) \neq S(f_2)$ を満たす必要がある．ここで $S(f)$ は機能 f を実行するシナリオの集合を表す．分析者が機能の実行有無を組み合わせて，この条件を満たすシナリオを準備できる場合には，すべての機能が異なる形式概念に分類される．この条件を満たせない場合でも，提案手法を用いて機能捜索を行ってよい．著者らは関連研究として，暫定的な機能とシナリオの集合により機能捜索を行い，得たプログラム要素を調査してシナリオの追加や修正を行った後に，提案手法を再度適用する繰返し型の手法を提案している [14].

3.2 レイヤごとの実行プロファイルの取得

既存手法 [10], [13] はシステムの実装に用いられた単一のプログラム言語に対して動的解析を実行するが，提案手法では動的解析で観測するプログラム要素の種類を多層システムに合わせて拡張する．具体的には，異なる記述言語で実装されたレイヤはシナリオ実行時に個別に動的解析を行い，シナリオとそのレイヤ内のプログラム要素の対応関係として観測する．

n 層システムの k 番目のレイヤ L_k に含まれるプログラム要素の集合を P_k とすると，すべてのプログラム要素の集合は

$$P = \bigcup_{k=1}^{n} P_k$$

と表せる．シナリオ $s_i \in S$ を 1 つずつ実行しながら各レイヤごとに動的解析を行い，シナリオ s_i によって実行されたレイヤ L_k のプログラム要素の集合 $T_{ik} \subseteq P_k$ を，すべての i，k について取得する．これを用いて，S と P_k の対応関係 $R_k : S \times P_k$ を

$$R_k = \bigcup_{i=1}^{l} \{\, (s_i, t) \mid t \in T_{ik} \,\}$$

により得る．

たとえば，図 2 でのシナリオ s_1 の実行により，点線で囲われた範囲のプログラム要素が実行されたとする．この場合，レイヤ L_1 では $T_{11} = \{p_{11}, p_{12}\}$，レイヤ L_2 では $T_{12} = \{p_{21}, p_{22}, p_{23}\}$，レイヤ L_3 では $T_{13} = \{p_{31}, p_{32}\}$ の計 7 つのプログラム要素が実行され，それぞれが s_1 と対応関係を持つ．

3.3 形式概念分析の適用

提案手法は既存手法 [10], [13] と同様に，1 回の形式概念分析を適用することによって複数の機能のそれぞれに関連するプログラム要素を得る．このとき，各機能の捜索結果として，すべてのレイヤからプログラム要素の候補を提示する点が既存手法とは異なる特徴である．この特徴を実現するため，提案手法ではレイヤごとに観測したシナリオとプログラム要素の対応関係 R_k を併合して，単一の形式文脈を形成するステップを追加する．

シナリオの集合 S をオブジェクト，機能またはプログラム要素の集合 $F \cup P$ を属性とし，それらの関係を $R : S \times (F \cup P)$ とすると

$$R = \bigcup_{k=0}^{n} R_k$$

であることから，形式文脈 $(S, F \cup P, R)$ が得られる．この形式文脈に対して形式概念分析を適用し，S の部分集合を外延，$F \cup P$ の部分集合を内包とする形式概念の集合を得る．

抽出された形式概念の集合 C について，各形式概念 $c \in C$ の内包に着目して機能捜索を行う．捜索方法は，ベースとしている既存手法 [13] と同様である．すなわち，まず，機能捜索で特定したい機能 f を内包に含む形式概念 c_f を探す．このとき，f とともに c_f の内包に現れるプログラム要素を最も f と関連性の高いプログラム要素と見なす．また，c_f の上位概念は f を実行するすべてのシナリオに f を実行しないシナリオの一部を加えたシナリオ群と関連するプログラム要素または機能を表す．より上位の概念になるほど f を実行しないシナリオの割合が増え，f との関連性が弱まるため，必要に応じて c_f に近いものの内包に現れるプログラム要素も弱く関連するとして捜索結果に含める．一方，c_f の下位概念は f を実行するすべてのシナリオのうち，一部を除いたシナリオ群と関連するプログラム要素または機能を表す．より下位の概念になるほど f を実行するシナリオ全体に対して外延の占める割合が減り，f との関連性が弱まるが，必要に応じて c_f に近いものの内包に現れるプログラム要素も弱く関連するとして捜索結果に含める．

必要に応じて，分析の対象とするレイヤを選択してもよい．たとえば，レイヤ L_1 と L_3 のみを分析の対象とする場合は，$R = R_0 \cup R_1 \cup R_3$ を用いて形式概念分析を行う．対象とするレイヤを制限すると，形式概念分析に関わる属性も指定したレイヤに属するものに限定されるため，すべてのレイヤを対象とするときに比べ，形式概念数の少ない概念束を得られる可能性がある．たとえば，アプリケーション層に属するモジュールは膨大になりがちであり，形式概念数の増大の主因となるため，全体の解析に先立ちプレゼンテーション層とデータ層にレイヤを制限してより小さな概念束を得ることにより，Web ページとテーブルアクセスから機能の大まかな役割を把握するといった用途がある．このようにして得た知見は，すべてのレイヤを有効にした概念束の絞り込み [15] などに役立つと考える．

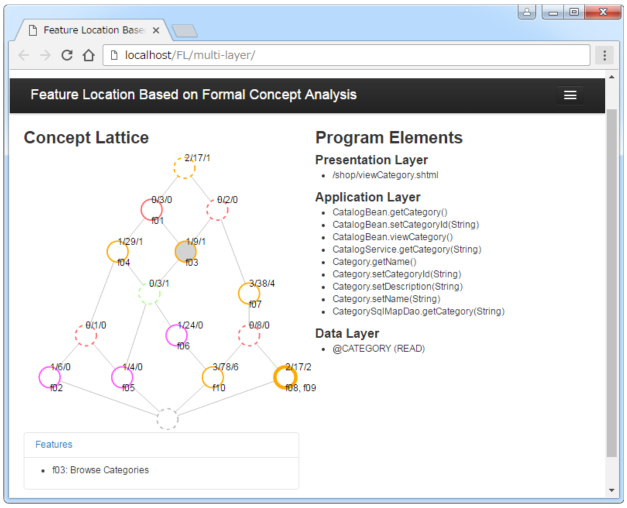

図 3 閲覧ツールの動作例
Fig. 3 Screenshot of the viewer.

4. 実装

Java で記述された Web アプリケーションを想定し，プレゼンテーション層，アプリケーション層，データ層の 3 つのレイヤからなるアプリケーションを解析できるよう，各レイヤで動的解析を行う環境を構築した．

プレゼンテーション層． JSP や HTML などの言語で実現されている．ここでは，Web ページの URL をプログラム要素とした．シナリオ実行中に動作した Web ページを抽出するため，HTTP 要求に含まれる `HTTP_REFERER` ヘッダをログへ出力するサーブレットフィルタを実装し，アプリケーションに追加することにより，アクセスされた URL の一覧を抽出できるようにした．URL に対して正規表現によるパターンマッチを行い，URL の構成要素からホスト名やパラメータを除いたパス部分のみを抽出した．また，JSP ファイルを変換して得られる Java クラスの実行を後述のアプリケーション層と同様の方法で監視することで，得られるメソッド呼び出しの実行トレースから JSP ファイルへのアクセスを抽出できるようにした．

アプリケーション層． Java 言語により実現されている．本論文では，Java のメソッドやクラスをプログラム要素とした．バイトコード変換によってクラスファイルを書き換え，各メソッドの先頭にログ出力を行うメソッド呼び出しを追加する動的解析器 YEBISU [16] を用いて，シナリオ実行中に動作したメソッド呼び出しを抽出できるようにした．また，メソッド粒度での解析が適さないような大きなシステムに対しては，動作したメソッドの所属しているクラスを抽出できるようにした．

データ層． SQL を用いた関係データベースへのアクセスにより実現されている．ここでは，関係データベースのテーブル名とそれらへの挿入，検索，更新，削除といった操作の種類の対をプログラム要素とした．シナリオ実行中に操作されたテーブルを抽出するため，iBatis フレームワーク*3のログ機構や log4JDBC ライブラリ*4を用いて実行されたSQL ステートメントを記録できるようにした．この機構では，ステートメントに対して正規表現でパターンマッチングを行い，SELECT 文や DELETE 文といった SQL ステートメントの種類からテーブルへの操作の種類を，また各文の FROM 句などからテーブル名を取得し，これらの対をプログラム要素として抽出する．

これらの出力に対して，ツール Concept Explorer [12] を用いて形式概念分析を適用した．また，概念束を視覚的に確認しやすいよう，プログラム要素をレイヤごとに閲覧できるツールを開発した．ツールの利用例を**図 3** に示す．左部のハッセ図の節は形式概念を表しており，レイヤごとのプログラム要素の数と，それらに強く関連する機能がラベル付けされている．節を選択すると，対応する形式概念の属性に対応するプログラム要素がレイヤごとに表示される．また，属しているプログラム要素の種類に応じて節が色付けされている．図では，機能 f_3: Browse Categories が属している形式概念を選択しており，節の色は 3 層にまたがるプログラム要素が対応付いていることを示している．実際，この概念にはプレゼンテーション層から 1，アプリケーション層から 9，データ層から 1 のプログラム要素が関連付いていることが分かる．

5. 評価

提案手法の有用性の評価として，複数のシステムに手法を適用した結果を分析した．提案手法は先行研究の手法 [13] の枠組みを用いた拡張であるため，拡張を行っても先行研究の手法の利点が引き続き機能するか，拡張により可能となった新しい分析方法が効果的に機能する状況が存在するか，などが評価の主要なポイントとなる．本論文で

*3 https://ibatis.apache.org/
*4 https://code.google.com/p/log4jdbc/

は以下の2点の評価を行った．

- **評価1：提案手法によって新たに特定されたプログラム要素の精度．**提案手法は，新たに捜索対象となったレイヤから，機能に対応付くプログラム要素を正確に漏れなく特定できることが望まれる．提案手法により機能に対応付けられた，プレゼンテーション層とデータ層に属するプログラム要素が，その機能に真に関連があるかを調査し，その結果を適合率，再現率としてまとめた．多くが正しく対応付けば，提案手法は該当の例題に対して正確にレイヤをまたがるプログラム要素群を特定できたことになり，既存手法 [13] の利点をそのまま享受できる．
- **評価2：レイヤの導入，調整による形式概念数の変化の調査．**提案手法は，モジュールに加え，多数多種のプログラム要素を形式概念分析の属性として扱う．属性数の増加は，形式概念数の増加，すなわち概念束の増大を招く可能性があり，これは後段の分析を難しくするおそれがある．例題への適用により，提案手法が既存手法を拡張したことによる形式概念数の大きな増加がないかを確かめる．また，提案手法による分析対象のレイヤの選択が形式概念数に与える影響についても同様に調べる．

先行研究の手法は，適用結果の概念束の分析者による手動分析をともない，その中では様々な曖昧な状況の判断を必要とするため，実際のシステムに対する適用事例によって評価されている [13]．提案手法に対しても同様の評価を行うことは有意義と考えるが，多層システムの機能捜索に対して上記のような評価を行った例は筆者らの知る限り存在せず，価値のあるものと考える．

5.1　評価1

提案手法の出力の適合精度を確認するため，Web アプリケーションの例題に適用した．例題として，オンラインでペットを購入するための電子商取引アプリケーション JPetStore 5.0*5を用いた．JPetStore を用いた理由は，その機能が書籍により既知だからである．Java EE の解説書 [17] では，このアプリケーションについて，10のユースケースを記述している．我々はこれらを既知の機能の集合 $F = \{f_1, \ldots, f_{10}\}$ として利用した．この中には，商品カタログを閲覧する機能である f_1: Browse Catalog や，サインオン・オフを行うための機能 f_7: Sign On and Off などが含まれている．我々は JPetStore の開発者ではないため，これらの機能のそれぞれを実現している手続きの流れやその手続きで用いられるデータ構造は未知であった．分析者としてそれらを理解し，設計ドキュメントに記述することを例題における機能捜索の目的とし，提案手法を適用した．

表 2　JPetStore のシナリオ
Table 2　Scenarios for JPetStore.

s_1	トップページから Fish カテゴリを参照する．
s_2	s_1 を実行後，Fish カテゴリを参照し，プロダクトの一覧から Koi の商品詳細を確認する．
s_3	s_1 を実行後，Fish カテゴリを参照し，プロダクトの一覧から Koi をショッピングカートに追加する．
s_4	サインオンを実行後に s_3 を実行し，注文を送信してサインアウトする．
s_5	トップページからサインオンを行い，サインアウトする．
s_6	トップページからサインオンを行い，アカウント情報のうち氏名，住所，電話番号，クレジットカード情報を更新してからサインアウトする．
s_7	トップページからキーワードに Koi を指定してカタログを検索し，検索結果の一覧から Koi のプロダクトを参照する．

表 3　シナリオと機能の対応関係
Table 3　Correspondence between scenarios and features.

	f_1: Browse Catalog	f_2: Search Catalog	f_3: Browse Categories	f_4: Browse Product Details	f_5: Browse Item Details	f_6: Update Shopping Cart	f_7: Sign On and Off	f_8: Update Account	f_9: Update Personalization	f_{10}: Submit Order
s_1	×		×							
s_2	×		×	×	×					
s_3	×		×	×		×				
s_4	×		×	×		×	×			×
s_5							×			
s_6							×	×	×	
s_7	×	×		×						

各機能 f について f の実行の有無を交差させ，$s_1, \ldots, s_7$ の7つのシナリオを作成した．シナリオの内容を**表 2** に，機能とシナリオの対応関係を**表 3** に示す．たとえば，シナリオ s_1 は，トップページにアクセスし，カテゴリ Fish を参照するもので，このシナリオは f_1: Browse Catalog, f_3: Browse Categories の2つの機能を実行している．なお，表 2 には明記していないが，すべてのシナリオで最後にトップページに戻るためのリンクをクリックした．

このシナリオは，3.1 節で述べた条件を考慮し，各機能を実行するシナリオの集合ができるだけ互いに異なるよう構成された．ただし，機能 f_8 と f_9 は同一画面で実現されており，一方のみを実行するシナリオを構成できなかったため，これらのみ条件を満たさず，同一の形式概念に分類された．

プレゼンテーション層，アプリケーション層，データ層の実行プロファイルから，それぞれ15種類の Web ページの URL，239種類のメソッド，16種類のテーブルからな

*5　http://mybatis.googlecode.com/files/JPetStore-5.0.zip

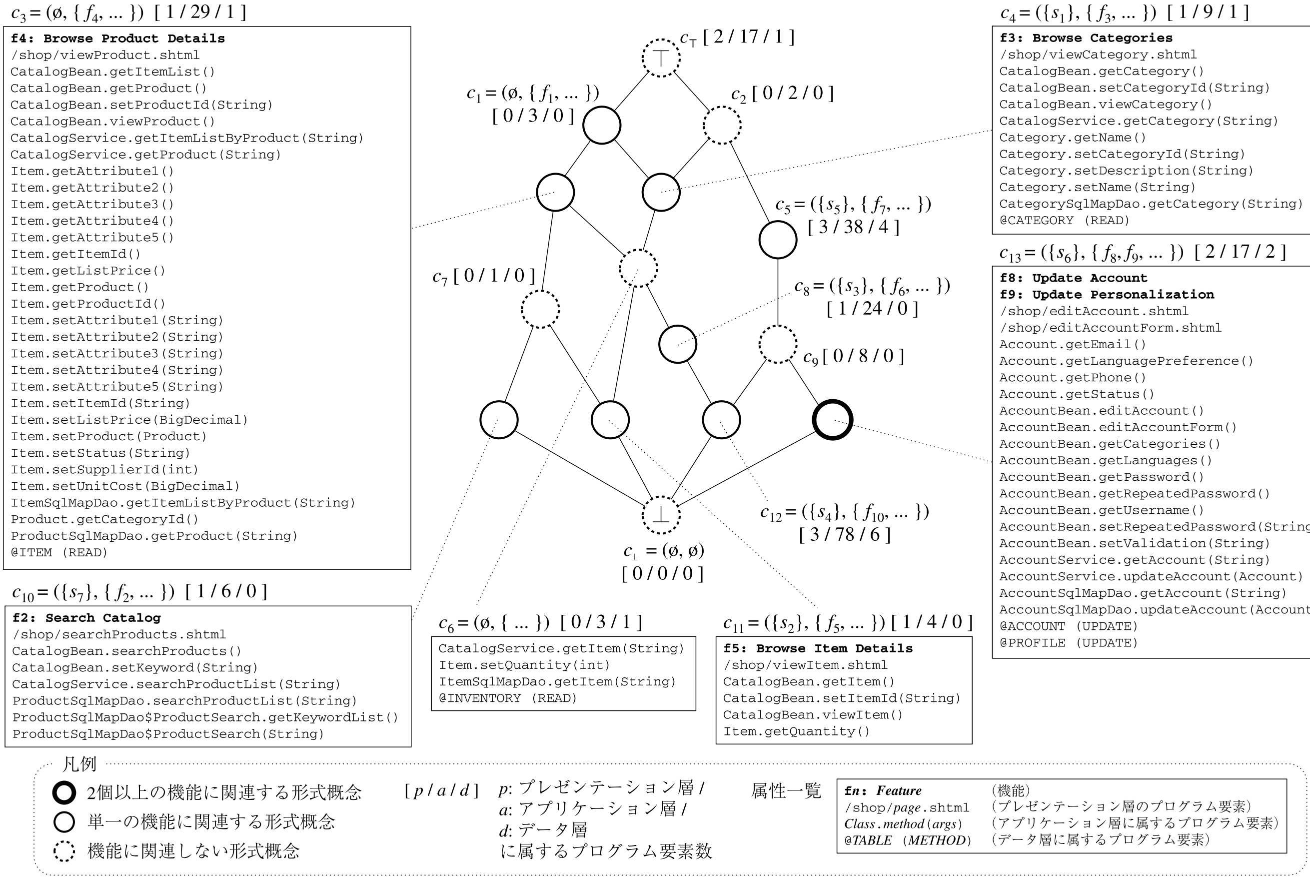

図 4　JPetStore の概念束

Fig. 4 Concept lattice of JPetStore.

る 270 のプログラム要素を観測した．これらのプログラム要素に 10 の機能を追加した 280 項目を属性とし，7 つのシナリオをオブジェクトとする形式文脈を得た．これを入力として形式概念分析を行い，**図 4** に示す，15 の形式概念からなる概念束を得た．この図では，内包に複数の機能を含む形式概念を太線で，機能をまったく含まないものを破線で示している．また，形式概念の内包には多数のプログラム要素が含まれるため，説明に必要な箇所以外はプレゼンテーション層，アプリケーション層，データ層の各レイヤに属するプログラム要素数のみを [p / a / d] のラベルで表示している．ここで，属性のうち “/” で始まるものはプレゼンテーション層に属する Web ページの URL，“@” で始まるものはデータ層に属する関係データベースのテーブルに対する操作を表し，それ以外の属性はすべてアプリケーション層に属する Java のメソッド名を表す．

得られた概念束では 270 のプログラム要素が 15 の形式概念のいずれかに分類され，分析者に既知の機能でラベル付けされる．既存手法に加えて分析対象となったプレゼンテーション層およびデータ層に属する全 31 種類のプログラム要素は，10 の形式概念に分類された．そのうち，機能も同時に分類されたものは 8 つあり，15 のうち 13 の Web ページ，16 のうち 14 のテーブルへの操作がそれらのいずれかに分類され，特定の機能と強く関連付けられることが提案手法によって示された．

本実験では，提案手法の機能捜索の精度を確認するために正解セットを作成した．正解セットは，著者の 1 名がプレゼンテーション層およびデータ層に属する全 51 種類のプログラム要素に対して関連する機能をすべて列挙し，別の著者 2 名によって関連の正しさを確認することで作成した．この調査では，モジュール間の呼び出し関係の分析やソースコードの閲覧を許可し，対象ソフトウェアを正しく理解しての結論を得られるよう努めた．調査の結果，37 種類のプログラム要素に対して 1 つ以上の機能との関連が見つかった．このため，プログラム要素によっては複数の機能が正解となることに注意されたい．また，用意した 7 つのシナリオによって得た実行プロファイルに含まれていた 31 種類の Web ページの URL とテーブルのうち，1 つ以上の機能と関係がある要素は 29 種類であった．用意したシナリオによる正解プログラム要素の実行網羅率は $29/37 \simeq 0.78$ であり，十分なシナリオが用意できていると判断した．

表 4 は，機能ごとの機能捜索された要素数，正答数，正解数と，それらをもとに算出した適合率，再現率を示している．機能捜索の結果は，機能が関連する概念に含まれる

表 4　機能ごとの適合率と再現率
Table 4　Precision and recall results for each feature.

機能	f_2	f_3	f_4	f_5	f_6	f_7	$f_{8,9}$	f_{10}	平均
回答数	1	2	2	1	1	7	4	9	–
正答数	1	2	2	1	1	7	4	9	–
正解数	2	3	3	4	2	8	9	9	–
適合率	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
再現率	0.50	0.67	0.67	0.25	0.50	0.88	0.44	1.00	0.61

プログラム要素のみを対象としている．ただし，f_1 は，正解セットに Web ページの URL とテーブルが含まれなかったため平均の算出からも除いている．また，f_8，f_9 は同一の形式概念に分類されたため，ここでは両方を含む単一の機能 $f_{8,9}$ として扱っている．実行された 29 種類の正解プログラム要素のうち，27 種類が機能捜索結果に含まれており，そのすべての割当てが正しく，平均適合率は 1.00 であった．本手法が対象とする 29 種類の実行されたプログラム要素は正解セット中にのべ 40 回出現し，機能ごとの平均再現率は 0.61 となった．

提案手法では希薄な概念束表現における内包を扱うため，上位概念と重複する要素は下位概念には含まれず，上位概念にのみ登場する．たとえば，f_5，f_6 のように機能が排他的に実行された場合は，双方に関連するプログラム要素は f_5 と f_6 に対応する概念の上位概念に分類され，いずれの機能捜索結果にも含まれない．また，f_4，f_5 のように，排他的に実行されるようにシナリオが構成できないケースでは，共通部分は f_4 に対応する概念にのみ含まれる．機能捜索に失敗した 7 種類のプログラム要素ののべ 13 回の出現を分析した結果，2 種類のプログラム要素（のべ 8 回出現）は，複数の機能において共通に用いられる要素であり，機能が関連付かない概念に分類されていた．また，残りの 5 種類（のべ 5 回出現）は，関連する要素集合間に包含関係が成立したため，2 つの正解のうちの片側にのみ分類されていた．

このように複数機能で共有するプログラム要素に対しては，既存手法 [10], [13] では，概念束として表現された分析結果を対象に，各機能に関係するプログラム要素の共通性と可変性を考慮しながら，着目する概念の上位概念と下位概念を範囲に含めて機能捜索を行う．しかしながら既存手法では，上位，下位概念の探索方法は規定されていない．我々はこれまでに探索方法によって得られる精度が異なることを明らかにしている [18]．本実験では，探索方法による影響を排除するために，上位，下位概念を対象としていないため，原理的に再現率を 100%にすることはできない．

提案手法は，プログラム要素群を機能の実装箇所の候補として提示するという利用方法を想定しており，この利用方法においては，適合率と再現率はどちらも重要な性能指標である．しかしながら，上位，下位概念を探索しない初期結果として，適合率 1.00，再現率 0.61 という精度で候補を提示できることは十分有用であるものと考える．

実際，提案手法によって得られた概念束はプログラム理解のために有用であった．たとえば，形式概念 c_3 では f_3: Browse Category に対して 1 の Web ページ，9 のメソッド，1 のテーブルからなる 11 のプログラム要素が関連付けられた．これらをソースコード上で読解すると，`/shop/viewCategory.shtml` という URL から処理が開始し，`CatalogBean.viewCategory()`，`CatalogService.getCategory(String)`，`CategorySqlMapDao.getCategory(String)` の順に Java メソッドが呼び出されることから，これら 4 つのプログラム要素が f_3 を実現するための手続きであることが理解できる．それ以外の 5 つのプログラム要素は Java のアクセサ（setter/getter）メソッド 4 件と関係データベースのテーブル 1 件であることから，この手続きで用いられるデータ構造に関連していると判断した．

機能捜索で得た Java メソッドがソースコード上で利用する API をもとに，我々が目視でレイヤ間の相互作用を確認した結果，JPetStore アプリケーションはプレゼンテーション層に Apache Struts，データ層に MyBatis というフレームワークを実装技術として用いていることが判明した．それぞれの実装技術は XML ファイルに記述された設定値に基づき，Web ブラウザからの URL の呼び出し，関係データベースに対する問合せをアプリケーション層の Java のメソッドの呼び出しと接続する役割を担う．このため，URL の末尾にパラメータとして付与された文字列が `CatalogBean.setCatalogId(String)` によって Java オブジェクトに格納されて手続きの中から参照されることや，`CategorySqlMapDao.getCategory(String)` メソッドが `CATEGORY` テーブルへの問合せを行い，得たデータを `Category` クラスのオブジェクトに代入することにより，形式概念に含まれるプログラム要素が相互作用を行い，機能を実現していることを理解した．

このような実装技術を用いたレイヤ間やプログラム要素間の相互作用を事前に把握していないくても，機能を実現するための手続きやデータ構造に関係するプログラム要素を提示できたことから，提案手法は機能を実現する手続きやデータ構造を理解するうえで有用な手がかりを提供できたと判断した．

5.2　評価 2

複数の Web アプリケーションに対して，提案手法のレイヤの導入，調整による形式概念数の分析を行った．

評価対象には，前述の JPetStore に加え，iTrust v15 [19] を用いた．その理由は，詳細なユースケース記述を含む要求仕様，ソースコード，テスト計画，自動化されたテストケースなどの iTrust の開発成果物がオンラインで入手可

表 5 形式概念数の分析結果
Table 5 Numbers of formal concepts.

						形式概念数	n_1	n_2	n_3	$n_{1,2}$	$n_{2,3}$	$n_{3,1}$	$n_{1,2,3}$
						L_1	√			√		√	√
						L_2		√		√	√		√
システム	$\|S\|$	$\|F\|$	$\|P_1\|$	$\|P_2\|$	$\|P_3\|$	L_3			√		√	√	√
JPetStore	7	10	15	239	16		12	15	12	15	15	12	15
iTrust$_{34}$	11	7	14	411	25		18	37	19	39	37	23	39
iTrust$^{M}_{\mathrm{tc}}$（メソッド粒度）	18	15	57	1,214	67		42	278	61	289	278	74	289
iTrust$^{C}_{\mathrm{tc}}$（クラス粒度）	18	15	57	208	67		42	129	61	144	132	74	146

能[*6]だからである．我々はこの中から，中程度の適用対象として，最も複雑なユースケースである UC34 に含まれる 11 のシナリオ（iTrust$_{34}$）を，また大規模な適用対象として 16 のユースケース（UC2，6，9，10，11，15，17，24，25，28，29，30，34，38，42，43; iTrust$_{\mathrm{tc}}$）を捜索すべき機能と見なし，それらのテストケースをシナリオとして用いた．これらのユースケースは自動的に実行できるテストケースを備えていることを著者らが確認したものである．また，後述する比較のため，iTrust の大規模な適用対象については，アプリケーション層でのモジュールをクラス粒度（iTrust$^{C}_{\mathrm{tc}}$）とメソッド粒度（iTrust$^{M}_{\mathrm{tc}}$）の両方で計測した．各システムのシナリオ数（$|S|$），機能数（$|F|$），観測された各レイヤでのプログラム要素数（プレゼンテーション層：$|P_1|$，アプリケーション層：$|P_2|$，データ層：$|P_3|$）は**表 5** に示されている．

加えて，表 5 の右部にはレイヤの導入による形式概念数の変化が示されている．右部の各列は，L_1 から L_3 までのレイヤの利用の有無を切り替えて形式概念分析を行った際の，得られた概念束の形式概念数を表している．たとえば，n_2 は，アプリケーション層を意味するレイヤ L_2 のプログラム要素のみを利用した場合の概念数を表しており，これは既存手法における分析結果に対するものにほかならない．また，$n_{1,2,3}$ は，3 層の全レイヤのプログラム要素を利用した場合の概念数を表しており，提案手法の主たる適用結果に対するものを意味する．この 2 列を比較すると，最も形式概念数の増加率が大きい場合は iTrust$^{C}_{\mathrm{tc}}$ によるものだが，それでも 146/129 = 113%（+13%）であり，アプリケーション層のみから得られたものに比べ，提案手法の結果の概念数は大きく増加しない．プレゼンテーション層やデータ層でのプログラム要素のシナリオに対する複雑さが，アプリケーション層での複雑さに比べ小さいことがこれに起因していると考える．

また，レイヤの選択により，小さな概念束を得られることも分かる．表 5 から分かるように，総じてアプリケーション層のプログラム要素を有する概念束の概念数（n_2，$n_{1,2}$，$n_{2,3}$，$n_{1,2,3}$）は，アプリケーション層を除いた概念束の概念数（n_1，n_3，$n_{1,3}$）よりも多い．これは，今回の対象アプリケーションのプログラム要素数において，アプリケーション層が支配的であることを意味している．アプリケーションが扱う Web ページやテーブルアクセスの種類に比べて，Java クラスやメソッドといったモジュールの種類が著しく多いことは予想しやすい結果だが，これはアプリケーション層を除いた 2 層での分析により，システムの外部的な振舞いである Web ページの表示やデータベースへのアクセスの関係に注目しながら，より少ない概念数で概念束を分析できることも示唆している．

*6 http://agile.csc.ncsu.edu/iTrust/wiki/doku.php?id=start

6. 議論

提案手法について，既存の機能捜索手法に対する優位性，多層システムに対する汎用性，他のプログラム理解手法との組合せの可能性，の 3 つの観点で議論する．

6.1 既存の機能捜索手法に対する優位性

形式概念分析を用いた既存の機能捜索手法 [10], [13] では，手続き型やオブジェクト指向言語で定義された第 1 級のプログラム要素を分析対象としており，Web ページやデータベースのテーブルのように純粋なプログラム言語以外の要素を扱わない．そのため，Java EE の例では Java で実装されたアプリケーション層のみが形式概念分析の対象となり，機能に関連するメソッド群を抽出した後，他のレイヤとの境界にあたるサーブレットや JDBC ドライバなどの API と関連するプログラム要素を分析して，機能と関連する Web ページやテーブルを見つける必要がある．

これらの要素を機能捜索の対象に含めたことが提案手法の特筆すべき点である．多層システムではレイヤ間のプログラム要素の協調関係が機能の理解に必要となるためである．これは，たとえば RUP などの方法論で用いられるロバストネス分析 [20] が 1 つの機能を Boundary，Control，Entity の 3 層のプログラム要素の協調関係に分解することからも分かる．提案手法では，協調する可能性の高いプログラム要素の集合を複数のレイヤにまたがって求められるため，得られた集合間の関係を分析することにより，協調関係にあるプログラム要素の理解を行える．

6.2 多層システムに対する汎用性

提案手法はレイヤの数や組合せに依存しないため，多様な構成の多層システムに適用可能である．レイヤ数を増減させたり組合せを変更したりする場合は，新たに追加されたレイヤについてのみ，独立したパラメータであるプログラム要素の定義や具体的な観測方法を考慮すればよい．

また，予備評価で用いた JPetStore はプレゼンテーション層を JSP，アプリケーション層を Java，データ層に関係データベースを扱う典型的な Java EE の参照アーキテクチャに基づいている．このアーキテクチャに基づくアプリケーションは現実的に多く存在しており，それらでは本論文と同様の方法で実行プロファイルを観測することにより機能捜索が可能である．

6.3 他のプログラム理解手法との組合せの可能性

形式概念分析は個々の形式概念の解釈を支援しないため，分析者は手作業で内包に含まれるプログラム要素の識別子名やソースコードの内容を確認し，グルーピングの意味を理解する必要がある．内包に多くのプログラム要素が含む場合にはこの労力が大きくなる．

形式概念に分類されたプログラム要素の一部のみを実行するようなシナリオを追加できれば，該当のプログラム要素の集合を分割できる．内包に複数の機能が含まれている場合，一方の機能を実行し，もう一方の機能を実行しないようなシナリオを構成できれば，形式概念分析を実行し直すことでそれらが異なる形式概念に分類されるようになる．一方で，こういったシナリオが構成不能であれば，提案手法により該当のプログラム要素の集合を分割することはできない．こういった場合，プログラム要素の関係を可視化したり，依存関係に基づき分割したりするような他の手法との組合せを検討する必要がある．たとえば，メソッド単位のプログラム要素をクラス単位に集約して表示したり，複数のプログラム要素の協調関係をデザインパターンとして表示したりするなどの支援が考えられる．また，動的解析結果から頻出する呼び出しのパターンを特定することにより，プログラム要素を部分的に理解していく手法 [16] との組合せも検討できる．さらに，依存グラフを横断することによりプログラムを理解する手法 [21] との併用も有意義と考える．

7. 関連研究

7.1 機能捜索手法

様々な機能捜索手法が提案されている．たとえば，Marcus らは正規表現によるパターンマッチング，依存グラフの探索，情報検索の 3 種類の技術に基づいた解析による機能捜索手法を事例研究によって比較し，それぞれに長所と短所が存在することを指摘した [7]．また，Dit らは機能捜索の分野における網羅的なサーベイを行い，既存の機能捜索技術を静的解析，動的解析，情報検索およびそれらの組合せに分類した [5]．これらの分類は，各手法が機能とソースコード中のモジュールを関連付けるために用いる解析の種類に基づいている．本論文の提案手法は形式概念分析に基づくため，ここでは我々のアプローチと類似する既存手法について紹介する．他の種類の機能捜索手法にも興味を持つ読者は文献 [5] も参考にされたい．

Eisenbarth らは形式概念分析，動的解析，静的解析を組み合わせて機能とソースコード中のモジュールの対応関係を識別するための機能捜索手法を提案した [13]．この手法では，シナリオとその実行プロファイルの中で観測されたモジュールとの対応関係から形式文脈を生成し，概念束を得る．分析者はシナリオと機能の対応関係を用いて概念束を分析する．その後，Koschke らはシナリオ–機能間の対応関係をシナリオ–モジュール間の関係とともに形式文脈に追加し，形式概念分析を適用するようにこの手法を改善した [10]．彼らは 2 つのコンパイラの事例研究の中で，機能捜索の対象とするモジュールの粒度をサブルーチンから基本ブロックの単位まで変化させ，そのときに得られる情報量や必要となるコストの間のトレードオフについて議論した．また，この手法は機能やシナリオが比較的少数のときに有効に機能するが，それらの数が増加すると形式概念の数が指数的に増加し，1 度に分析可能な機能の数や範囲に限度があることを示した．

Poshyvanyk らは潜在的意味解析（latent semantic indexing）と形式概念分析を組み合わせた機能捜索手法を提案した [22]．彼らの手法では，ユーザが指定した検索文字列との類似度に基づいて検索結果のモジュールを順位付けし，上位から一定数のモジュールを選んだうえで，それらの識別子に出現する単語の共通点や相違点を概念束として提示する．彼らは順位付きリストで提示する単純な機能捜索手法よりも，概念束を提示するほうが労力を削減できることを示した．検索結果の数を上位から一定数で制限することにより，彼らの手法は分析者に扱える範囲に概念束の大きさを保つことに成功している．

7.2 ヘテロジーニアスなシステムに対する解析

多層システムに限らず，複数のコンポーネントをまたがったプログラム解析では，利用しているプログラミング言語が単一でなかったり，コンポーネントを横断する依存関係の抽出が難しかったりするため，解析が難しくなる．これに対して，プログラミング言語に依存しないデータ解析基盤に各コンポーネントに対する解析結果を登録し，基盤上で問合せを行うことにより，横断的な解析を実現する手法がある．たとえば，RDF に基づく統一的なプログラムの解析系の試作例が複数ある [23], [24]．提案手法も，形式概念分析をこういったデータ解析基盤と見なせば，同様のアプローチと見なせる．ただし提案手法は，形式概念分

析による共通性・可変性の分析に基づき，コンポーネント（レイヤ）を横断するような解析をいっさい行わずに捜索結果を得ている点が大きな特徴である．これは，新種のレイヤへ提案手法を適応させる際に作成する必要のある解析器の構築が容易であることを示唆している．また，レイヤの組合せ部分に対する解析が不要なため，必要な解析器の種類も肥大しない．

津村らは，ソースコードとデータベース間のトレーサビリティリンク回復のために，静的・動的解析を併用したハイブリッド解析を適用する手法を提案している [25]．この手法では，動的解析を用いて，プログラミング言語にかかわらないプログラム要素の抽出を狙っている点は提案手法と類似している．一方で，この手法はコードとデータベースの間の関係の解析に特化しており，一般的な多層システムに対して適用できるような手法を定義していない．なお，津村らは静的解析を併用することで，あらかじめ用意したテストケースでは実行されない部分の解析も実現している．動的解析に基づく手法は十分なテストケースを準備することが前提となっており，この問題は提案手法にもあてはまる．機能捜索の結果に基づき，対話的にテストケースを更新していく手法 [14] などの併用を検討すべきと考える．

8. おわりに

本論文では動的解析と形式概念分析を組み合わせることにより，ソフトウェアの持つ機能と複数のレイヤに分散したプログラム要素群の対応関係を抽出する手法を提案した．

また，Java EE の 3 層アーキテクチャに基づく例題システムに対して提案手法を適用し，機能とプログラム要素の対応関係を精度良く抽出した事例を示すことにより，手法の有用性を示した．さらに，複数のシステムに対しての分析により，レイヤの導入によるオーバヘッドがなかったこと，またレイヤの選択によるシステムの大まかな分析が可能であることも示した．

今後は提案手法をさらに発展させるとともに，以下のような応用的な課題に取り組みたい．

- **他のレイヤに対する分析．** たとえば，Web アプリケーションにおいては，クライアントサイドで JavaScript を用いた処理が実行されることがある．JavaScript における各モジュールの実行プロファイルを，プレゼンテーション層とは別に取得し，提案手法に組み込むことにより，クライアントサイドに隠れた機能も抽出可能と考える．
- **大規模なプログラムへの適用を通じた事例研究．** Eisenbarth らの取り組み [13] のような，現実的なプログラムの理解に提案手法が役立つかを確認したい．

謝辞 本研究の一部は科学研究費補助金（Nos. 15H02683, 15K15970）の助成を受けた．

参考文献

[1] Kazato, H., Hayashi, S., Okada, S., Miyata, S., Hoshino, T. and Saeki, M.: Feature Location for Multi-Layer System Based on Formal Concept Analysis, *Proc. 16th European Conference on Software Maintenance and Reengineering* (*CSMR'12*), pp.429–434 (2012).

[2] Software Engineering Standard Committee of the IEEE Computer Society: IEEE Std 830: Recommended Practice for Software Requirements Specifications (1998).

[3] Rajlich, V. and Wilde, N.: The role of concepts in program comprehension, *Proc. 10th International Workshop on Program Comprehension* (*IWPC'02*), pp.271–278 (2002).

[4] Rajlich, V.: *Software Engineering: The Current Practices*, CRC Press (2011).

[5] Dit, B., Revelle, M., Gethers, M. and Poshyvanyk, D.: Feature Location in Source Code: A Taxonomy and Survey, *Journal of Software: Evolution and Process*, Vol.25, No.1, pp.53–95 (2013).

[6] Fowler, M.: *Patterns of Enterprise Application Architecture*, Addison-Wesley (2002).

[7] Marcus, A., Rajlich, V., Buchta, J., Petrenko, M. and Sergeyev, A.: Static Techniques for Concept Location in Object-Oriented Code, *Proc. 13th International Workshop on Program Comprehension* (*IWPC'05*), pp.33–42 (2005).

[8] Dilshener, T. and Wermelinger, M.: Relating Developers' Concepts and Artefact Vocabulary in a Financial Software Module, *Proc. 27th IEEE International Conference on Software Maintenance* (*ICSM'11*), pp.412–417 (2011).

[9] Parnas, D.L.: On the Criteria to Be Used in Decomposing Systems into Modules, *Comm. ACM*, Vol.15, No.12, pp.1053–1058 (1972).

[10] Koschke, R. and Quante, J.: On Dynamic Feature Location, *Proc. 20th IEEE/ACM International Conference on Automated Software Engineering* (*ASE'05*), pp.86–95 (2005).

[11] Ganter, B. and Wille, R.: *Formal Concept Analysis: Mathematical Foundations*, Springer (1999).

[12] Yevtushenko, S.A.: System of data analysis "Concept Explorer" (in Russian), *Proc. 7th National Conference on Artificial Intelligence* (*KII'00*), pp.127–134 (2000).

[13] Eisenbarth, T., Koschke, R. and Simon, D.: Locating Features in Source Code, *IEEE Trans. Softw. Eng.*, Vol.29, No.3, pp.210–224 (2003). full version is available from ⟨http://www.bauhaus-stuttgart.de/bauhaus/papers/tse2003.pdf⟩.

[14] Kazato, H., Hayashi, S., Kobayashi, T., Oshima, T., Okada, S., Miyata, S., Hoshino, T. and Saeki, M.: Incremental Feature Location and Identification in Source Code, *Proc. 17th European Conference on Software Maintenance and Reengineering* (*CSMR'13*), pp.371–374 (2013).

[15] Carpineto, C. and Romano, G.: Using Concept Lattices for Text Retrieval and Mining, *Formal Concept Analysis: Foundations and Applications*, Ganter, B., Stumme, G. and Wille, R. (Eds.), Lecture Notes in Computer Science, Vol.3626, pp.161–179 (2005).

[16] Kazato, H., Hayashi, S., Oshima, T., Miyata, S., Hoshino, T. and Saeki, M.: Extracting and Visualizing Implementation Structure of Features, *Proc. 20th Asia-Pacific Software Engineering Conference* (*APSEC'13*), pp.476–484 (2013).

[17] Singh, I., Stearns, B. and Johnson, M.: *Designing En-*

terprise Applications with the J2EE Platform, 2nd Edition, Prentice Hall (2002).

[18] 中野真明貴，林　晋平，小林隆志：動的機能捜索における関連度と探索戦略，電子情報通信学会技術研究報告，Vol.116, No.127, pp.169–174 (2016).

[19] Cleland-Huang, J., Gotel, O. and Zisman, A.: *Software and Systems Traceability*, Springer (2012).

[20] Jacobson, I.: *Object Oriented Software Engineering: A Use Case Driven Approach*, Addison-Wesley (1992).

[21] 加藤哲平，林　晋平，佐伯元司：呼び出し関係グラフ分割手法の動的機能捜索手法との組合せの検討，電子情報通信学会論文誌，Vol.J98-D, No.11, pp.1374–1376 (2015).

[22] Poshyvanyk, D. and Marcus, A.: Combining Formal Concept Analysis with Information Retrieval for Concept Location in Source Code, *Proc. 15th IEEE International Conference on Program Comprehension* (*ICPC'07*), pp.37–48 (2007).

[23] Tappolet, J., Kiefer, C. and Bernstein, A.: Semantic web enabled software analysis, *Web Semantics: Science, Services and Agents on the World Wide Web*, Vol.8, No.2-3, pp.225–240 (2010).

[24] 風戸広史，宮田俊介，星野　隆：複数の言語で記述されたプログラムの解析における RDF リポジトリの応用，情報処理学会研究報告，Vol.2011-SE-174, No.3, pp.1–8 (2011).

[25] 津村耕司，鷲崎弘宜，深澤良彰，土屋良介，大島敬志，三部良太：静的・動的ハイブリッドな解析によるコード・データのトレーサビリティリンクの抽出，ウィンターワークショップ 2014・イン・大洗論文集，pp.5–6 (2014).

風戸 広史 （正会員）

2003 年東京工業大学大学院情報理工学研究科計算工学専攻修士課程修了．同年株式会社 NTT データ入社．2010 年東京工業大学大学院同専攻博士後期課程修了．博士（工学）．ソフトウェア設計方法論，ソフトウェア開発環境等の研究に従事．

林 晋平 （正会員）

2008 年東京工業大学大学院情報理工学研究科計算工学専攻博士後期課程修了．現在，同大学情報理工学院助教．博士（工学）．ソフトウェア変更やソフトウェア開発環境の研究に従事．

大島 剛志

2010 年北海道大学大学院工学研究科応用物理学専攻修士課程修了．同年日本電信電話株式会社入社．ソフトウェア開発技術に関する研究開発に従事．

小林 隆志 （正会員）

2004 年東京工業大学大学院情報理工学研究科計算工学専攻博士課程修了．同大学学術国際情報センター助手，名古屋大学大学院情報科学研究科准教授を経て，東京工業大学情報理工学院准教授．博士（工学）．ソフトウェア再利用技術，プログラム理解，リポジトリマイニング，ソフトウェア設計等の研究に従事．

夏川 勝行

1996 年奈良先端科学技術大学院大学情報科学研究科情報処理学専攻博士前期課程修了．同年日本電信電話株式会社入社．1999 年テレコミュニケーションマネジメント功労賞（電子情報通信学会），2001 年学術奨励賞（電子情報通信学会）．ソフトウェア開発技術に関する研究開発に従事．

星野 隆 （正会員）

1991 年電気通信大学大学院電気通信学研究科電子情報学専攻博士前期課程修了．同年日本電信電話株式会社入社．データベース，データ流通方式，ソフトウェア開発技術に関する研究開発に従事．

佐伯 元司 （正会員）

1983 年東京工業大学大学院工学研究科情報工学専攻博士後期課程修了．同大学助手，助教授を経て，現在，同大学情報理工学院教授．工学博士．要求工学やソフトウェア開発技法等の研究に従事．