Vol.2019-SE-201 No.11
2019/3/7

オープンソースリポジトリから自動抽出されたAPI集合の検索性向上のための頻出パターンマイニングによるAPI集合間の関係付け

近藤 悠志[1]　西本 匡志[1]　西山 佳志[1]　川端 英之[1]　弘中 哲夫[1]

概要：様々な機能の実装例から抽出されたAPI呼び出しの順序付き系列やAPI集合などの，いわゆるAPI使用パターンは，APIの使用法を把握したい開発者にとって有用な情報となりうる．これを踏まえ，我々は，オープンソースリポジトリからAPI集合を自動抽出するシステムを開発しているが，類似したAPI集合が多数得られる場合が多く，ユーザが欲する情報を効果的に提示できているとは必ずしも言えなかった．これに対し本研究では，API集合の検索性を向上させる手法を提案する．本手法では，多量のAPI集合に対して頻出パターンマイニングを適用し，API集合の相互関係や階層構造，個々のAPIの役割を浮かび上がらせる．そして，得られた情報に基づくグラフベースのGUIにより直感的なAPI集合探索を可能にする．ユーザは，提示されるグラフを辿るだけで目的のAPI集合を得ることができるものと期待される．本稿では，提案手法の詳細およびGUIの設計，プロトタイプの実装について述べ，その有用性について議論する．

キーワード：API集合，頻出パターンマイニング，アプリケーション開発，オープンソースリポジトリ

Extracting Detailed Relations among API Member Sets by using Frequent Pattern Mining to Improve Searchability for Usable API Usage Patterns from Open-Source Repositories

Yushi Kondoh[1]　Masashi Nishimoto[1]　Keiji Nishiyama[1]　Hideyuki Kawabata[1]
Tetsuo Hironaka[1]

Abstract: The API usage patterns that are extracted from many examples of implementations for various kinds of functionalities might be usable for application developers who want to know the API usage examples. Although several studies have been proposed tools for obtaining API usage patterns, existing systems are not useful enough for developers because the search results are often difficult to digest. In this paper, we propose a method to improve the efficiency of an existing API member set search system which extracted offers a way for searching open-source repositories for usable information on API member sets. In this research, we extract, by utilizing the Frequent Pattern Mining, detailed relations among API member sets, hierarchical structure and each API member's. The graph-based display of the structure of API member sets is expected to help the user to grasp easily the differences among API member sets and what API members are important compared to others. The API member set search is accomplished smoothly by traversing displayed graphs by the user. In this paper, we describe the details of our method and the design and implementation of the prototype GUI, and we discuss the usability of the proposed system.

Keywords: API member set, Frequent pattern mining, Application development, Open source repositories

1. はじめに

近年のアプリケーション開発は，ライブラリやフレームワークの API (Application Programming Interface) に依存している．例えば Android のアプリケーションは，平均的にソースコードの 30%から 50%が Android API に依存しているという [1]．すなわち，効率的にアプリケーションを開発するためには，API を上手く組み合わせて使用する必要があると言える．したがって開発者には，実装内容に応じた適切な API の使い分けに関する知識や，プログラミングルールに従った API の記述方法に関する知見などが求められる．しかし，API に関する詳細な情報はドキュメント化されていないことも多い．また，API の仕様はバージョンごとによって異なる場合がある．実際，ライブラリやフレームワークは頻繁にアップデートされている [2]．このようなことから，一般に開発者にとって API の使用法や記述例を得ることは難しいと言われている．

開発者が API の使用法や記述例に関する情報を得る手段の 1 つとして，コード検索ツール [3] [4] を活用することが挙げられる．これらのツールを使用することで，Github [5] や Bitbucket [6] などのソースコードリポジトリで公開されている実際のソースコードを参照することができる．しかしこれらのツールの多くは，ソースコードに対して字面検索をするため，開発者が定義した関数の名前やコードとは無関係のコメント部分のみが検索結果として提示される場合も多々あるなど，開発者が欲する API に関する情報を即座に得られないこともある．

これに対し我々は，ソフトウェアリポジトリのソースコードにおける特定の機能の実装例から自動抽出した API の集合を検索することができる，協働 API 集合検索システム (CAPIS) を開発している [7]．CAPIS はユーザの検索に対して，特定の機能を実装する上で不可欠とされる協働 API 集合を提示することができる．しかし，検索によって得られる協働 API 集合はどれも類似しており，集合間の違いを把握しにくいことから，ユーザが欲する情報を効果的に提示できているとは必ずしも言えない．また，検索によって提示される API 集合から機能を整える際に中心的な役割をする API がどれであるかといった，個々の API の役割を把握することも難しいと言える．

そこで我々は，CAPIS を改良し，API 集合の検索性を向上させるための手法を提案する．本手法は，オープンソースリポジトリから得られる多量の API 集合に対して，頻出パターンマイニング [8] を適用することで，API 集合の相互関係や階層構造を抽出し，個々の API の役割を浮かび上がらせる．つまり，API 集合間の関係や API 集合内における個々の API の役割を明確にする．ユーザは，グラフィカルに表示されるグラフを辿るだけで目的の API 集合を得ることができるものと期待される．

本手法は汎用性があり，呼び出し順序関係を持たない API 集合が入力であれば適用可能である．すなわち，提案手法によって API 集合をリスト形式で提示する既存ツールにおける問題点も解決できる可能性がある．

以下，2 章では，関連研究を紹介する．3 章では，CAPIS の発展や改善点について述べる．4 章では，本研究の提案手法について詳述し，5 章で提案手法に基づく API 集合探索システムの設計と実装，システムの使用例について述べる．6 章では，API 集合探索システムのプロトタイプを用いて本手法の有用性を評価し，考察する．最後に 7 章でまとめと今後の課題について述べる．

2. 関連研究

API 使用パターン (API Usage Pattern) の定義は様々であるが，一般的にオープンソースリポジトリマイニングによって得られる，ライブラリやフレームワークにおける API メソッドの組み合わせである．また，個々の API 使用パターンの位置付けは，API の呼び出し順序関係やある機能を実装する上で必要とされる API の組み合わせとされることが多い．MAPO [9] や UP-Miner [10] は，オープンソースリポジトリから API のメソッド呼び出しシーケンス (Method Call Sequences) を抽出し，API 使用パターンとしている．これに対し我々は，協働 API 集合と呼ばれる，アプリケーションにおける特定の機能を実装するために不可欠とされるライブラリのメソッド名や名前付き定数の組み合わせをソースコードから自動抽出する [7] [11]．CAPIS は，API 同士の関連性をクラス名で判断し，メソッドの境界を超えた分析によって関連深い API の集合を抽出する．これに対して，SSS は複数のメソッドをまたがるデータの依存関係を分析することで，API 集合を抽出する．SSS が抽出する API 集合は，CAPIS に比べて，機能を実装するのに適した集合である [11]．

ソフトウェア開発支援の一環として，プログラムのエラー検出や API の推薦，コード補完に API 使用パターンや API 集合を活用する研究がなされている．PR-Miner [12] は，ソースコードから得られる API 呼び出しの順序付き系列に対して頻出パターンマイニングを適用することで，文書化されていない暗黙のプログラミング規則を抽出し，プログラムのエラー検出に活用している．また，DynaMine [13] は，プログラムにおける API の修正履歴から得られる頻出 API 使用パターンをエラー検出に用いている．早瀬ら [14] は，特定のメソッド内で使用される可能性が高い API の集合を抽出し，開発者に推薦するツールを提案している．一方，MAPO [9] は，ソースコードから得られる API 呼び出しの順序付き系列から頻出であるパターンを抽出し，各使用パターンの出現頻度やコードスニペットとともに開

[1] 広島市立大学
Hiroshima City University

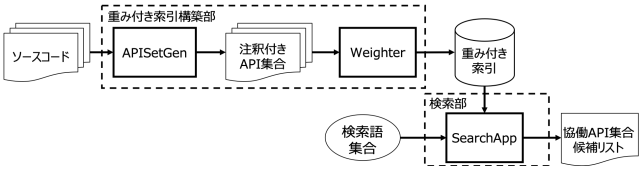

図 1 協働 API 集合検索システムの全体構成

表 1 クラス名に着目して図 2 から抽出した API 集合

APISet
Activity*1→ getSystemService
Context*2→ POWER_SERVICE Context*2→ WINDOW_SERVICE
WindowManager*3→ getDefaultDisplay
PowerManager*4→ newWakeLock PowerManager*4→ SCREEN_BRIGHT_WAKE_LOCK
PowerManager.WakeLock*4→ acquire PowerManager.WakeLock*4→ release
Display*3→ getRotation

発者に提示する API 使用パターン推薦ツールである．また，Android アプリケーションの開発を支援する DroidAssist [15] は，隠れマルコフモデルによる呼び出し順序付き API の状態遷移図に基づいて開発者に API を推薦し，コードを補完する．

3. モチベーション：CAPIS の改善

CAPIS はユーザが与える検索語に適した複数の協働 API 集合を提示することができるシステムである．しかし CAPIS による検索結果表示には改善の余地がある．ここでは，API 集合の検索性をさらに向上させるため，CAPIS の提示方法における改善の余地について述べる．

3.1 CAPIS [7] の概要

CAPIS は，特定の機能を実装するために不可欠とされる API の集合 (以下，協働 API 集合) を検索するためのシステムである．CAPIS は，抽出された協働 API 集合を識別子に含まれる単語集合で構成される文書とみなして TF-IDF により重み付けし，ユーザが与える検索語集合とのコサイン尺度によって得られる数値に基づいた順位付けを行う．

CAPIS は，図 1 に示すように重み付き索引構築部および検索部の二つから構成されている．

重み付き索引構築部 APISetGen と Weighter から成る．APISetGen では，ソースコード中で使用されている API 同士の関連性を所属クラスで判断し，メソッドの境界を超えた分析によって API 集合を自動抽出する．Weighter では，個々の API 集合に対して，API の重複除去と検索のための注釈を抽出する．

検索部 検索部は，ユーザが与える検索語に基づいてコサイン尺度による検索結果のランク付けして提示する．

```
54 :public class AccelerometerPlayActivity extends Activity {
58 :    private PowerManager mPowerManager;
59 :    private WindowManager mWindowManager;
60 :    private Display mDisplay;
61 :    private WakeLock mWakeLock;
64 :    @Override
65 :    public void onCreate(Bundle savedInstanceState) {
72 :        mPowerManager = (PowerManager) getSystemService(POWER_SERVICE);
75 :        mWindowManager = (WindowManager) getSystemService(WINDOW_SERVICE);
76 :        mDisplay = mWindowManager.getDefaultDisplay();
79 :        mWakeLock = mPowerManager.newWakeLock(
                    PowerManager.SCREEN_BRIGHT_WAKE_LOCK, getClass().getName());
86 :    }
88 :    @Override
89 :    protected void onResume() {
96 :        mWakeLock.acquire();
100:    }
102:    @Override
103:    protected void onPause() {
114:        mWakeLock.release();
115:    }
117:    class SimulationView extends FrameLayout implements SensorEventListener {
358:        @Override
359:        public void onSensorChanged(SensorEvent event) {
371:            switch (mDisplay.getRotation()) {
388:            }
389:        }
428:    }
429:}
```

図 2 Google Samples [16] に含まれるソースコード AccelerometerPlayActivity.java

表 2 データ依存関係に着目して図 2 から抽出した API 集合

APISet
Activity*1→ getSystemService Context*2→ POWER_SERVICE Context*2→ WINDOW_SERVICE WindowManager*3→ getDefaultDisplay Display*2→ getRotation
Activity*1→ getSystemService PowerManager*4→ newWakeLock PowerManager*4→ SCREEN_BRIGHT_WAKE_LOCK PowerManager.WakeLock*4→ acquire PowerManager.WakeLock*4→ release

3.2 CAPIS におけるその後の発展

我々は，CAPIS における APISetGen の改良に取り組んでいる [11]．従来の CAPIS における APISetGen では，API の所属クラス名に注目して API 集合を構成する．したがって，所属クラス名に類似性が見られない API 同士が協調している様子を把握できないという欠点がある．一方，改良版 APISetGen はデータ依存関係に基づいて API 集合を抽出することにより，そのような欠点を排除している．

図 2 のソースコードから，それぞれの抽出手法により得られた API 集合を表 1 および表 2 に示す．各 API 名は，クラス名→メソッド名で表される．表 1 では単独で抽出されていた Activity*1→ getSystemService が，表 2 では，抽出された 2 つの API 集合の両方に属していることが分かる．以降，特に明記しない限り，改良版 APISetGen を組み込んだ CAPIS を単に CAPIS と呼ぶ．

3.3 CAPIS におけるさらなる改善の余地

CAPIS では，大量のソースコードから自動抽出された

*1 android.app.
*2 android.content.
*3 android.view.
*4 android.os.

図 3　協働 API 集合検索システム*5による検索語集合 {text, view} の検索結果上位 5 件

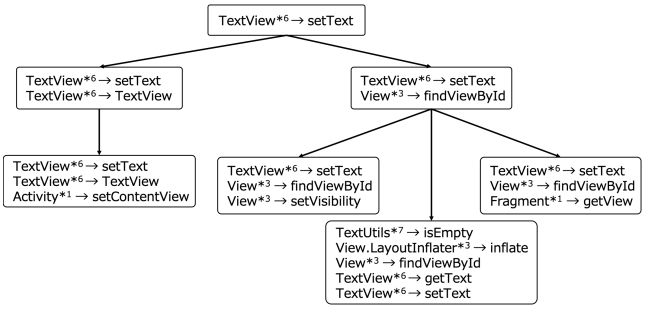

図 4　包含関係 (グラフ表現)

API 集合の提示方法について，いくつかの改善の余地があると考えられる．

3.3.1　API 集合間の違いの見分けにくさ

CAPIS によってリスト表示される API 集合は類似したものが多く見られ，各 API 集合間の違いを把握することは難しい場合が多い．例えば，図 3 に示す検索結果例において，TextView*6→ setText は全ての API 集合で使用されているが，TextView*6 → TextView は 1 番目と 2 番目のみでしか使用されていない．このような各集合間の違いを図 3 の検索結果から一目で見分けることは難しい．

3.3.2　API 集合における構成要素の分析機能の欠如

特定の機能を実装するために必要とされる API に関する情報を得たいユーザにとって，個々の集合中で中心的な役割を果たす API がどれであるかという情報は有益であると考えられる．しかし，CAPIS の画面出力からそのような情報を得ることができない．

*5　http://capis.ca.info.hiroshima-cu.ac.jp:8090
*6　android.widget.
*7　android.text.

表 3　CAPIS の検索結果 (リスト表現)

順位	API 集合
1	TextView*6→ TextView TextView*6 → setText
2	Activity*1 → setContentView TextView*6 → TextView TextView*6 → setText
3	TextUtils*7 → isEmpty View.LayoutInflater*6 → inflate View*3 → findViewById TextView*6 → getText TextView*6 → setText
4	Fragment*1 → getView View*3 → findViewById TextView*6 → setText
5	View*3 → findViewById View*6 → setVisibility TextView*6 → setText

4.　頻出パターンマイニングによる API 集合同士の関係付けとそれに基づく GUI

本研究では，検索結果で得られる API 集合をリスト表現で提示する既存システムに対して，API 集合同士を関係付けし，グラフ表現を用いて提示する手法を提案する．API 集合をグラフ表現で提示することにより，API 集合間の違いや階層構造，特定の機能を実装する際に中心的な役割を果たす API を明確にして提示することができる．

4.1　API 集合の関係付け

表 3 は，CAPIS [7] に対して，検索語集合 {text, view} を与えた時に得られる結果の上位 5 件である．リスト表示された各 API 集合を構成する API 名は類似しており，各集合間の相違点は把握しにくい．また，API 集合を構成する個々の API の役割を表 3 から見出すことは難しい．ここで，表 3 にリスト表示された 5 件の API 集合について，各集合間の包含関係を分析すると，図 4 のような階層構造を見出すことができる．図 4 の構造は，ユーザにとって有用であると考えられる情報が露わになっている．多量の API 集合の中から図 4 のような構造を見出し，ユーザに対して効果的に提示することができれば，ユーザは API 集合を選び易くなるのではないかと考えられる．

4.2　API 集合における包含関係の導出

多量の API 集合から図 4 のような構造を抽出するためには，データ集合中に高頻度で見られるパターンを抽出する手法である頻出パターンマイニング [8] が活用できる．頻出パターンマイニングを用いると，各 API 集合内で同時に使用されている API の組み合わせから成る API 集合を得ることができる．例えば，表 3 のオープンソースリポジトリから抽出された 5 件の API 集合に対して頻出パターンマイニングを適用すると，飽和アイテム集合 (Closed

表 4　表 3 に対する頻出パターンマイニング適用結果

件数	API 集合
5	TextView*6 → setText
3	TextView*6 → setText View*3 → findViewById
2	TextView*6 → TextView TextView*6 → setText
1	Activity*1 → setContentView TextView*6 → TextView TextView*6 → setText
1	TextUtils*7 → isEmpty View.LayoutInflater*3 → inflate View*3 → findViewById TextView*6 → getText TextView*6 → setText
1	Fragment*1 → getView View*3 → findViewById TextView*6 → setText
1	View*3 → findViewById View*3 → setVisibility TextView*6 → setText

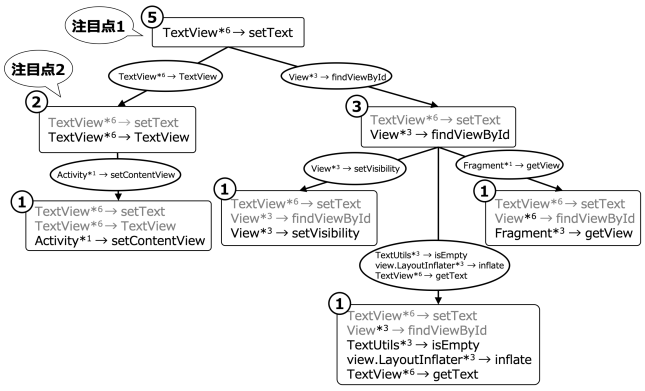

図 5　差分 API 集合を明記した API 集合グラフ

Itemset) として表 4 に示す 7 件の API 集合が生成される．図 4 は表 4 の 7 つの API 集合に対し，「API 集合 A が API 集合 B の部分集合であるとき，API 集合 A から API 集合 B へのパスが必ず存在する」ように描かれている．

表 3 の 5 つの API 集合の包含関係から描かれるグラフは，単一のグラフにはならない．これに対し頻出パターンマイニング適用後に生成される表 4 の API 集合は元の API 集合には存在していなかった集合を含んでおり，生成される全ての集合の関係を一つのグラフで表すことができる．

4.3　提案：API 集合グラフに基づく GUI

ユーザに対して図 4 のグラフを提示することで，ユーザはグラフを辿りながら直感的に API 集合を得ることができると考えられる．図 5 は図 4 に対し，各エッジ上にその両端の API 集合間の差分を明記したものである．我々は図 4 を API 集合グラフ，API 集合グラフにおける各ノードを API 集合ノード，各エッジ上に表示された楕円を差分ノードと呼ぶ．また，API 集合グラフにおける各ノードには当該 API 集合の頻度情報が添えられている．

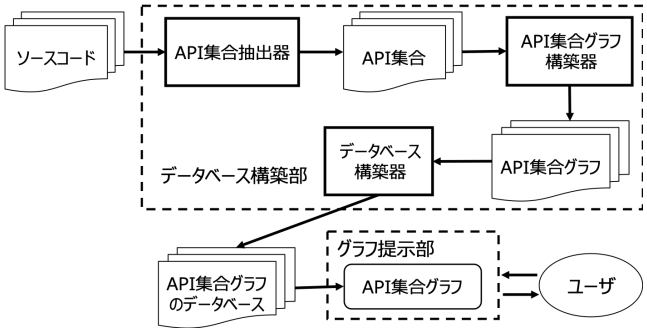

図 6　API 集合探索システムの全体構成

5.　提案手法に基づく API 集合探索システム

本節では，API 集合探索システムの設計とプロトタイプの実装について述べる．

5.1　システムの設計

本システムの内部構成を図 6 に示す．本システムは，オープンソースリポジトリ等のソースコードから API 集合グラフを構築する．本システムは，予め API 集合グラフに関する情報を保存しておくためのデータベース構築部，および，ユーザの操作に応じてインタラクティブに API 集合グラフを表示するグラフ提示部によって構成されている．

5.1.1　データベース構築部

API 集合抽出器　複数のプログラムから抽出されるデータ依存関係に基づいて API 集合を構築する．

API 集合グラフ構築器　抽出された API 集合に対して頻出パターンマイニングを適用し，飽和 API 集合を生成する．生成された飽和アイテム集合の包含関係を分析し，複数の API 集合グラフを構築する．

データベース構築器　構築した個々の API 集合グラフにおけるルートノードからタグクラウドに表示する単語とその重要度 (フォントサイズの指標) を決定し，格納する．また，API 集合グラフにおけるノードの接続関係を格納する．

5.1.2　API 集合グラフ提示部

API 集合グラフ提示部では，ユーザが API 集合グラフを辿ることを補助するために，インタラクティブな操作が可能な GUI を整える．

5.2　プロトタイプの実装

プロトタイプシステムの記述言語には Java を用いた．入力とするソースコードとしては，Google Samples [16] を用いた．API 集合グラフ生成器における頻出パターンマイニングには Apache Spark [17]，データベース構築器では Apache Lucene [18] のライブラリをそれぞれ用いた．

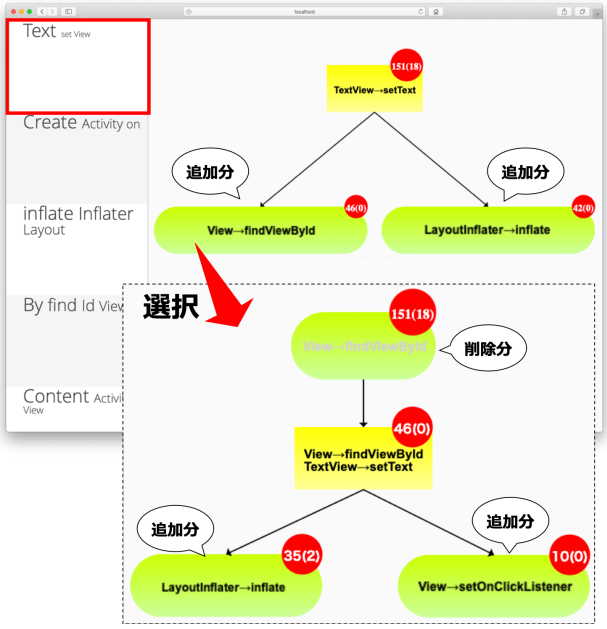

図 7　API 集合探索システムのプロトタイプ

5.3　GUI

本システムは，予め構築しておいたデータベースから，個々の API 集合グラフについてタグクラウドを生成し，図 7 画面左のようにリスト形式で表示される．本システムは検索語集合を与える CAPIS とは異なり，提示されるタグクラウドを選択すると，当該 API 集合グラフを代表するルートノードの API 集合が図 7 画面右に黄色く描画される．

図 7 左画面におけるタグクラウドの各項目は，各 API 集合グラフのルートノードから生成される．またタグクラウドは，API 集合グラフの数だけ表示されるため，一つの機能に対してタグクラウドが複数存在することもある．画面右に黄色く表示されるノードを API 集合ノードと呼ぶ．API 集合ノード周辺に緑色で表示される楕円ノードは差分ノードを表しており，当該 API 集合に対して API をいくつか追加あるいは削除した API 集合が存在していることを示している．ユーザは差分ノードをクリックすることで，右画面上の API 集合ノードを変更し，API 集合グラフを辿っていくことができる．

図 7 は，図 5 の API 集合グラフを探索する様子である．図 7 画面左上のタグクラウドを選択すると，図 5 のルートノードが画面右に黄色で表示される．合わせて表示される差分ノードは，図 5 のルートノードから伸びる 2 本のエッジに添えられたものと対応している．図 7 左画面左下のノードを選択すると，図 7 右上のような画面が表示される．これは，図 5 において注目点 1 から注目点 2 へとグラフを辿ったことを表している．

本システムは，ユーザがグラフを辿る際の参考情報として，CAPIS では提示されなかった API 集合の頻度情報をノードに添える (図 7 ノード右上の赤丸)．赤丸内の数値のうち括弧外の数値は，当該 API 集合が頻出パターンマイニングによって生成された数，括弧内の数値は，当該 API 集合がソースコードから自動抽出された数をそれぞれ示している．また，差分ノードに添えられるこれらの数値は，その差分ノードを経て辿った先に存在する API 集合の頻度情報である．

図 5 下において，それぞれのノードに添えられた頻度情報を見ると，上部の差分ノードの頻度に比べて，API 集合ノードや下部の差分ノードは頻度が大きく減少することが分かる．この頻度情報は，多くの開発者が上部の差分ノードを選択した場合の API 集合を用いて TextView 機能を実装していることを明らかにしている．

6.　評価および考察

本節では，提案手法の有用性について評価するために API 集合探索実験を行う．探索実験では，Android フレームワークを用いて，ある機能を実装する場面を想定し，それぞれの機能を実装するための有益な情報を得るべく，提案システムのプロトタイプを用いて API 集合を探索する．比較対象に CAPIS を用い，API 集合の検索性を評価する．また，プロトタイプを用いた API 集合の探索によって得られる集合が，実際に機能を実装する上で有用な情報であるかについても検証する．

6.1　評価方法の概要

提案する API 集合探索システムの評価にあたって，以下の RQ を明らかにすることにより目的達成の度合いについて確認する．

RQ1　API 集合の探索にあたって個々の API 集合の違いが把握し易い提示がなされているか

RQ2　機能を実装する際に中心的な役割を担う API，もしくは API 集合を把握できるか

RQ3　探索によって選択した API 集合は実装の参考となるか

6.2　API 集合探索実験

本実験で使用する API 集合は，Google Samples [16] において Java で記述された 157 の Android プロジェクトから自動抽出されたものを使用する．

6.2.1　実験方法

本実験では，Android アプリにおいて画面に警告メッセージを表示する AlertDialog 機能を実装する場面を想定し，各機能の実装に必要とされる API 集合を探索する．

プロトタイプを用いた API 集合の探索では，実装する機能に関係する単語が含まれるタグクラウドのうち，最も上位のものを選択したときに表示される API 集合グラフを辿ることで，目的の API 集合を探し出す．CAPIS を用いた検索にあたっては，実装したい機能を整えるために使用さ

表 5　CAPIS に対して検索語集合 {alert, dialog, show} を与えた時の検索結果

順位	API 集合
1	android.app.**Dialog** → **show**
2	android.app.**AlertDialog**.Builder → Builder android.app.**AlertDialog**.Builder → setMessage android.app.**AlertDialog**.Builder → setPositiveButton android.app.**AlertDialog**.Builder → **show**
3	android.app.**AlertDialog**.Builder → Builder android.app.**AlertDialog**.Builder → setMessage android.app.**AlertDialog**.Builder → setPositiveButton android.app.**AlertDialog**.Builder → setTitle android.app.**AlertDialog**.Builder → **show** android.content.**Dialog**Interface.OnClickListener → OnClickListener
4	android.app.**AlertDialog**.Builder → Builder android.app.**AlertDialog**.Builder → create android.app.**AlertDialog**.Builder → setNegativeButton android.app.**AlertDialog**.Builder → setPositiveButton android.app.**AlertDialog**.Builder → setSingleChoiceItems android.app.**AlertDialog**.Builder → setTitle android.app.**Dialog** → **show** android.content.Context → getString
5	android.app.**AlertDialog**.Builder → Builder android.app.**AlertDialog**.Builder → setMessage android.app.**AlertDialog**.Builder → setPositiveButton android.app.**AlertDialog**.Builder → **show** android.app.Fragment → getActivity

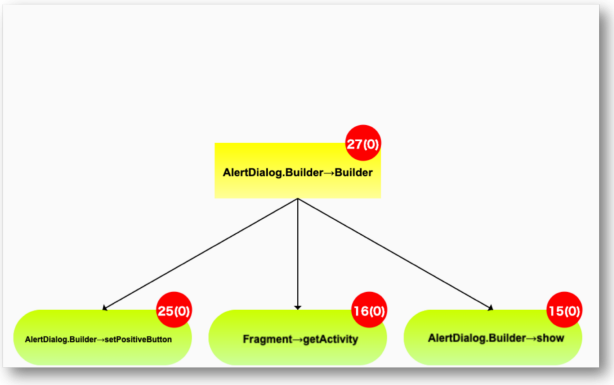

図 8　API 集合探索実験の様子 1

れる API 名を構成する語を与える．具体的には，CAPIS に対して検索語集合 {alert, dialog, show} を与える．

6.2.2　実験：AlertDialog 機能のための API 集合探索

CAPIS の場合　検索によってリスト表示される API 集合上位 5 件は表 5 のとおりである．すなわち，比較的集合サイズが大きいものばかりが列挙されている．また，各集合を構成する API のほとんどが，名前に “AlertDialog” や “Dialog” を持っている．したがって，各集合においてどの API が共通して現れているのかを判別することは難しい．

API 集合探索システムの場合　プロトタイプを用いて AlertDialog 機能について探索する際，最初に表示される画面を図 8 に示す．図 8 において，頻度が大きく，“AlertDialog” をクラス名に含んだ左下の差分ノードを選択し探索を続けると，図 9 の画面が表示される．ここでは，クラス名に “AlertDialog” を含んだ API が 2 つ表示された右下の差分ノードを選択することで図

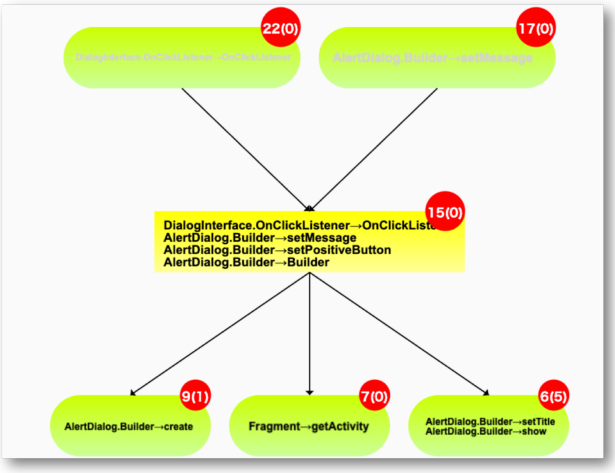

図 9　API 集合探索実験の様子 2

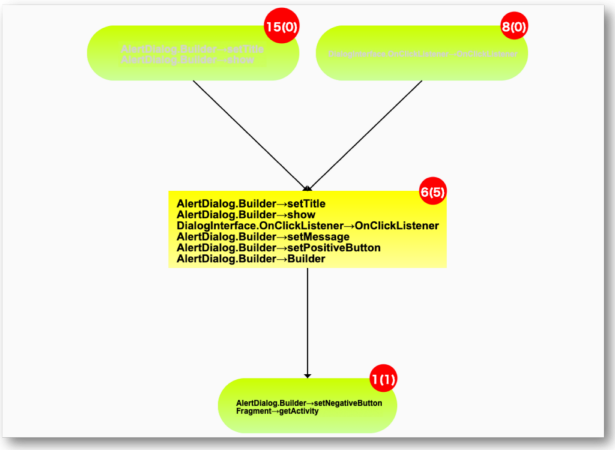

図 10　API 集合探索実験の様子 3

10 が表示される．図 10 において，ソースコードから抽出された API 集合の数から，妥当性が高そうな中央の API 集合 1 つを自然に選択することができる．

6.3　評価実験に対する考察

6.3.1　RQ1：API 集合の探索にあたって個々の API 集合の違いが把握し易い提示がなされているか

CAPIS では，サイズが大きいかつ類似点が多い集合同士の比較であるため，単純なリスト表示では違いが判別しにくい．これに対しプロトタイプでは，子ノードに表示される API のクラス名の情報から少しずつ，各機能に関連する API を追加していく形で探索する．これにより，大きな集合同士の比較をしなくて済むため，集合間の違いを捉えやすくなっており，検索性が向上していると言える．

6.3.2　RQ2：機能を実装する際に中心的な役割を担う API，もしくは API 集合を把握できるか

CAPIS では，各 API 集合を構成する個々の API の重要度を表すような指標は提示されない．これに対し本提案手法は，各集合において中心的に使用される API，または API 集合から順に選択していく形でグラフを辿ることがで

```
new AlertDialog
        .Builder(this)
        .setTitle("Title")
        .setMessage("Message")
        .setPositiveButton("OK", new DialogInterface.OnClickListener() {
      public void onClick(DialogInterface dialog, int whichButton) {}
}).show();
```

図 11　AlertDialog 機能を実装するソースコードの一部

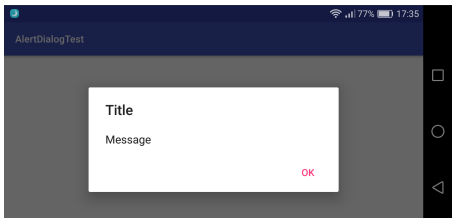

図 12　実装した AlertDialog 機能のスクリーンショット

きる．また，AlertDialog 機能に関する API 集合の探索では，同時に使用されることが多い API の組み合わせについても確認できる．

6.3.3　RQ3：探索によって得られた集合は実装の参考となるか

図 11 は，探索実験によって得られた API 集合のみを用いて，AlertDialog 機能を実装したコードの一部である．実装した AlertDialog は，アプリが起動すると同時に図 12 に赤枠で囲った警告メッセージを出力する．ダイアログ内に表示される “OK” と表示されたボタンをクリックすることで，何らかの処理を実装することができる．

6.3.4　API 集合探索実験のまとめ

RQ1 から RQ3 を踏まえた結果，直感的にグラフを探索することで API 集合を得ることができる本システムは，CAPIS と比較して目的の API 集合を選択し易くなったと言える．また，探索実験によって得られた API 集合は，実際に開発の参考になりうることも検証された．今後は，提案システムを用いた探索によって得られる API 集合の有用性についてさらに議論するために，多くの機能について探索実験を行う必要がある．

7.　まとめと今後の課題

本研究では，API 集合の検索性向上を目的として，多量の API 集合から抽出された集合間の相互関係をグラフベースで効果的に提示する手法について提案し，プロトタイプを用いた評価を行った．プロトタイプを用いた API 集合探索実験によって，各 API 集合間の違いや中心的な API の役割が把握し易くなり，API 集合を選択し易くなることが分かった．

今後の課題としては，実際の開発の場面を想定し，被験者実験等によって提案手法の有用性を明らかにすることが望まれる．また，API 集合探索機能に加えて，コード合成補助機能の導入等による総合的なアプリケーション開発支援に向けた本手法の応用についても検討したい．

参考文献

[1] Joao Eduardo Montandon, Hudson Borges, D. F. and Valente, M. T.: Documenting APIs with Examples: Lessons Learned with the APIMiner Platform, *20th Working Conference on Reverse Engineering (WCRE '13)* (2013).
[2] Lamba, Y., Khattar, M. and Sureka, A.: Pravaaha: Mining Android Applications for Discovering API Call Usage Patterns and Trends, *Proceedings of the 8th India Software Engineering Conference (ISEC'15)*, pp. 10–19 (2015).
[3] searchcode, available from ⟨https://searchcode.com/⟩
[4] Nerdydata, available from ⟨https://nerdydata.com/search⟩
[5] GitHub, available from ⟨https://github.com⟩
[6] Bitbucket, available from ⟨https://bitbucket.org⟩
[7] 西本匡志，川端英之，弘中哲夫：アプリケーション開発支援のための協働 API 集合検索システム，電子情報通信学会論文誌 D， Vol. J101-D, No. 8, pp. 1176–1189 (2018).
[8] Han, J., Pei, J. and Yin, Y.: Mining Frequent Patterns without Candidate Generation, *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data (SIGMOD '00)*, pp. 1–12 (2000).
[9] Zhong, H., Xie, T., Zhang, L., Pei, J. and Mei, H.: MAPO: mining and recommending API usage patterns, *European Conference on Object-Oriented Programming (ECOOP '09)* (2009).
[10] Wang, J., Dang, Y., Zhang, H., Chen, K., Xie, T. and Zhang, D.: Mining succinct and high-coverage API usage patterns from source code, *10th Working Conference on Mining Software Repositories (MSR '13)* (2013).
[11] Nishimoto, M., Nishiyama, K., Kawabata, H. and Hironaka, T.: Easy-Going Development of Event-Driven Applications by Iterating a Search-Select-Superpose Loop, *Journal of Information Processing* (to appear).
[12] Li, Z. and Zhou, Y.: PR-Miner: Automatically Extracting Implicit Programming Rules and Detecting Violations in Large Software Code, *Proceedings of the 10th European software engineering conference held jointly with 13th ACM SIGSOFT international symposium on Foundations of software engineering (ESEC/FSE-13)*, pp. 306–315 (2005).
[13] Livshits, B. and Zimmermann, T.: DynaMine: finding common error patterns by mining software revision histories, *Proceedings of the 10th European Software Engineering Conference with 13th ACM SIGSOFT International Symposium on Foundations of Software Engineering (ESEC/FSE-13)*, pp. 296–305 (2005).
[14] 早瀬康裕，鬼塚勇弥，山本哲男，石尾隆，井上克郎：API 呼び出しとメソッド周辺の識別子の実績に基づいた API 集合推薦手法，情報処理学会論文誌，Vol. 56, No. 2, pp. 692–700 (2015).
[15] Nguyen, T. T., Pham, H. V., Vu, P. M. and Nguyen, T. T.: Recommending API Usages for Mobile Apps with Hidden Markov Model, *Proceedings of 30th IEEE/ACM International Conference on Automated Software Engineering (ASE 2015)*, pp. 795–800 (2015).
[16] Google Samples, available from ⟨https://github.com/googlesamples⟩
[17] Apache Spark, available from ⟨https://spark.apache.org⟩
[18] Apache Lucene, available from ⟨https://lucene.apache.org/core/⟩