

ソースコードに含まれる機能を要約して一覧表示できる プログラム理解支援ツール

西山 佳志¹ 西本 匡志¹ 川端 英之¹ 弘中 哲夫¹

概要: 一つのプログラムは一般に複数の機能を組み合わせて構築される。開発者が保守や拡張を行うにあたって、プログラムに含まれる機能の記述内容の詳細を把握する必要がある。この記述内容の把握は長大なソースコードの字面から各機能を構成する API や複数の機能間の関係性等の様々な分析を行わなければならない。一般に多大な労力が求められる。本研究では、これらの作業の負荷軽減を目的として、ソースコードから機能ごとの記述を抽出・提示する手法を提案する。また、本手法に基づくソースコード理解支援ツールを設計し、そのプロトタイプを実装した。本ツールはソースコード解析に文の順序関係を考慮しないグラフ構造を用いて、複数のメソッドに跨って機能を抽出できる。本ツールの評価の結果、特定の機能における記述箇所の把握に要する負担を軽減できる可能性があることを確認できた。

キーワード: ソースコード理解, データフローグラフ, タグクラウド, イベント駆動型プログラミング

Supporting Program Understanding by Itemizing Summaries of Functionalities Extracted from Source Code

KEIJI NISHIYAMA¹ MASASHI NISHIMOTO¹ HIDEYUKI KAWABATA¹ TETSUO HIRONAKA¹

Abstract: A program, in general, comprises of many functionalities. In order to maintain and/or extend a program, a programmer is required to grasp how each functionality is implemented. Generally speaking, detailed analyses of a lengthy program for understanding which part of one functionality is described where, how multiple functionalities are merged, and what kinds of APIs are used, are burden on programmers. In order to alleviate the situation, we propose a method of classifying sentences of which a program consists, and a way to show the information in a comprehensive manner. We designed and developed a prototype tool that itemizes summaries of functionalities extracted from source code. Our tool accepts programs consisting of functionalities that are composed by using library APIs. The tool utilizes a graph representation of program structure dealing with unordered set of sentences to analyze source code to extract functionalities described by sentences scattered over multiple methods. The results of preliminary experiments confirm that the tool is effective for finding out important sentences for maintenance purpose from application programs.

Keywords: program understanding, dataflow graph, tag-cloud, event-driven programming

1. はじめに

一つのプログラムは一般に複数の機能を組み合わせて構築される。開発者が保守や拡張を行うにあたっては、それら個々の機能の処理内容の把握が必要となる場面がある。例えば、保守において、ある機能の振る舞いを変更する場

面や、拡張において、ある機能を他のソースコードから流用して実装する場面などである。

しかし、個々の機能の処理内容を把握するためには、ソースコード中に散在する特定機能の記述箇所の把握が必要である。それにかかる、文と文の間の制御やデータの流れを推移的に辿る作業は、開発者にとって負担となっている。さらに、プログラムに付随するドキュメントやコメント等の説明が不十分であれば、プログラムに含まれる個々の機

¹ 広島市立大学
Hiroshima City University

能の把握が困難である。

我々は個々の機能の処理内容を把握する際に生じるこれらの障害を取り除く支援を行うことで、保守や拡張における開発者の負担軽減を目指す。具体的には、プログラムに含まれる各機能を要約して一覧提示できるツールがあれば、ドキュメントやコメントとは異なる手段でプログラムに含まれる各機能を把握できそうである。さらに、一覧提示された各機能について記述箇所を提示できる機能があれば、ソースコード中に散在する特定機能の記述箇所を把握することが可能となる。

このような支援環境を実現するためには、ソースコードから個々の機能に関連する処理をしている文の集合の記述を抽出する手法が不可欠である。一般に機能は複数の文が密接に協調しあうことで実現されることから、互いに協調しあう文の集合を抽出することにより、ソースコードから個々の機能の実装に深く関連する部分のみを取り出して把握することが可能になるのではないかと考えられる。

昨今のアプリケーションプログラムは API を駆使して構築されるため、API の相互関係に着目した分析が不可欠である。また、アプリ開発においては Android アプリに代表されるイベント駆動型プログラミングが高頻度で用いられており、これを適切に扱える支援環境の実現が望まれる。これらを踏まえると、メソッド境界を超えて API の相互関係を分析する手法が必要である。これに対し、我々は、オブジェクトを介して相互に作用しあう API の関係に着目し、データ依存グラフをベースとした文集合抽出手法により、メソッド境界を超えて密接に関連する文集合を抽出してユーザに提示する手法を提案する。

これまでに、ソースコードから機能毎の記述を抽出する手法がいくつか提案されている [7][9]。しかしながらこれらの方法では、記述の抽出の単位がブロック境界やメソッド境界の制約を強く受けるため、イベント駆動型のアプリケーションにおける機能の実装に関連する文集合を必ずしも適切に抽出することはできない。これに対し我々の手法では、共通のオブジェクトに対して相互作用する API 参照に関連する文をまとめて抽出できるため、メソッド境界などの構文上の制約を受けることなく、協調しあう API にまつわる文集合を抽出することができる。

我々は、ライブラリを利用して実装されている機能(以下、ライブラリ機能と略する)毎に記述を抽出・提示するソースコード理解支援ツールを設計し、プロトタイプを実装した。提案ツールは、各ライブラリ機能について、実装に利用した API を構成する単語を用いて要約を一覧表示し、各機能について記述箇所を提示できる GUI を持つ。

2. ソースコード記述に対する機能の抽出

我々は Android 等のイベント駆動型プログラミングに対応したライブラリによって構築されるアプリケーション

のソースコード理解を支援することを目指している。我々の支援はソースコードから機能ごとの記述を抽出してユーザに機能の概要や対応箇所を提示する方式である。我々のソースコード記述に対する機能抽出は、API にまつわる文の集合を機能とみなし、メソッドごとではなく、ソースコードごとに文集合を取り出すといった特徴を持っている。

2.1 我々が求めるソースコード記述の機能抽出

プログラムの可読性や保守性を意識して、モジュール化を徹底したプログラミングがなされていれば、一般に、プログラム全体の振る舞いがメソッド単位の記述の組み合わせとして記述される。しかし、イベント駆動型プログラミングではアプリケーション内部の状態が変化するたびに、予め決められたメソッドが呼び出される。開発者はそのメソッドに対して機能を実現する記述を行うため、複数のメソッドにまたがることが多いと考えられる。

例えば、Android ライブラリにおけるセンサ値の取得機能は、クラス *SensorManager* のドキュメントにおいて次のように記述することが推奨されている [3]。

- メソッド “onResume” において、「センサ値の取得を開始する API」の呼び出しを記述する。
- メソッド “onPause” において、「センサ値の取得を終了する API」の呼び出しを記述する。

このように複数の API を組み合わせて構築されるライブラリ機能は、API の記述場所に何らかの制約があり複数のメソッドに渡って記述されることが多い。よって、我々は複数のメソッドにまたがる API 呼び出しの連携を把握し、それらの API にまつわる文を集合として取り出す。

2.2 例：Android のソースコードに対する機能抽出

Android アプリケーションのソースコード(図 1)の記述に対し、我々の求める機能で抽出した例について説明する。このソースコード中の個々の文(文 $a \sim g$)は、次の 2 つの機能のいずれかに関わっている。

- 端末電源のオン状態を維持する機能 (WakeLock 機能)
- 端末画面にトースト^{*1}を表示する機能 (Toast 機能)

WakeLock 機能は、文 g による開始処理で果たされる。文 g で参照する変数 $mWakeLock$ は、文 c により定義される。文 c で参照する変数 $mPowerManager$ は、文 a により定義される。以上により、WakeLock 機能の実装に関わった文の集合は文 a, c, g であるため、この文集合を WakeLock 機能として抽出することが望まれる。

また Toast 機能は、文 f による生成処理で果たされる。文 f を実行するかどうかは文 e の条件節の結果に依存する。文 e, f で参照する変数 $text$ は文 d により定義される。文 d で参照する変数 $mEditText$ は文 b により定義される。以上

*1 端末画面上に表示される短いメッセージのこと

```
public class Sample1 extends Activity {
    ...
    @Override
    public void onCreate(Bundle savedInstanceState) {
        ...
        a: mPowerManager = (PowerManager) getSystemService(POWER_SERVICE);
        b: mEditText = (EditText) findViewById(id.edittext);
        c: mWakeLock = mPowerManager.newWakeLock(
            PowerManager.SCREEN_BRIGHT_WAKE_LOCK, getClass().getName());
    }
    @Override
    protected void onResume() {
        ...
        d: String text = mEditText.getText().toString();
        e: if(!text.isEmpty()){
            f: Toast.makeText(this, text, Toast.LENGTH_LONG);
        }
        g: mWakeLock.acquire();
    }
    ...
}
```

図 1 Java による Android アプリケーションのソースコード

により、Toast 機能の実装に関わった文の集合は文 *b*, *d*, *e*, *f* であるため、この文集合を Toast 機能として抽出できることが望まれる。

2.3 既存手法は我々が求める機能を抽出できるか

我々がソースコードから抽出したい機能の単位はブロックよりも小さな粒度である。そのため、吉田ら [7] のブロックの集合を機能として抽出する手法では、我々が求めている機能を抽出することには適していない。一方で、平山ら [9] の文の集合を機能として抽出する手法は、我々が求めている機能を抽出することができそうである。しかし、平山らの手法はプログラム依存グラフ (PDG, Program Dependency Graph) [1] を利用してメソッド単位で機能を抽出するため、我々が目指すメソッドを跨いで記述された機能を抽出することができないと考えられる。

3. 提案するソースコード理解支援ツール

提案ツールはソースコードからライブラリ機能ごとの記述を抽出・提示し、ユーザのライブラリ機能の単位でのソースコード理解を支援する。さらに、ソースコード中における特定のライブラリ機能の記述箇所の把握を手助けする。提案ツールにより、保守や拡張におけるソースコードの理解に要する負担の軽減を目指す。

3.1 提案ツールの機能と使用例

提案ツールは以下の 2 つの機能を持つ。

機能 1 ユーザが与えたソースコードからライブラリ機能ごとの記述を抽出し、その各機能の概要を表したタグクラウドを一覧にして提示する。

機能 2 ユーザが特定のライブラリ機能のタグクラウドを選択すれば、その機能に関するソースコードの記述箇所を強調表示する。

ユーザはタグクラウドの一覧 (機能 1) から、ソースコード中に実装されている各ライブラリ機能を把握できる。さらに、特定のライブラリ機能に対応するソースコードの記述

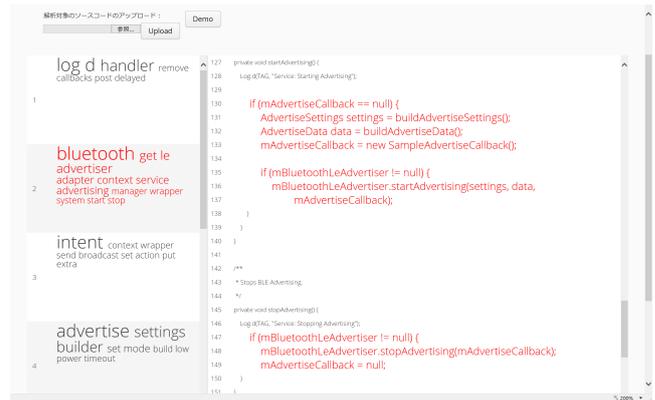


図 2 提案ツールの使用例

箇所 (機能 2) を把握することができる。これにより、ユーザはソースコード中の膨大な記述から理解したいライブラリ機能の記述のみを確認することができる。

提案ツールの使用例について、図 2 を用いて説明する。図 2 の使用例は、Android アプリケーションのソースコードが読み込まれ、タグクラウドの一覧には Android ライブラリの機能が並んでいる様子を示している。それを上から順に流し読みすれば、上から順に、「ハンドラのログ出力」、「Bluetooth」、「インテント」「アドバタイジングの設定」に関する機能だと直感的に確認できる。ここで、Bluetooth に関する機能の記述を理解したければ、2 番目のタグクラウドをマウスでクリックする。図 2 のように、Bluetooth に関する機能の記述箇所が赤字で強調表示される。

3.2 簡易 IVDFG

我々は、3.1 節で述べた機能 1、及び機能 2 におけるソースコードからライブラリ機能の抽出において、異なるメソッドで記述された文と文の間にもデータ依存関係を繋ぐ変数間データフローグラフ (IVDFG, Inter-Variable Data Flow Graph)[8] から制御依存関係を省略した「簡易 IVDFG」を提案し、それを利用する。

簡易 IVDFG は、IVDFG における制御依存関係を省略したグラフである。また、IVDFG は文より詳細な情報を持つが、本研究の領域には関与しないため、扱う情報の最小単位は文にしている。

図 1 のソースコードの簡易 IVDFG を図 3 に示す。簡易 IVDFG の頂点は、変数頂点、条件頂点、文頂点の 3 種類である。変数頂点は、ソースコード中の各変数を楕円形の頂点で表し、内部にその変数に作用する文を加える。本手法ではある変数 *v* に作用する文を、1. 変数 *v* の宣言文、2. 変数 *v* の代入文、3. 変数 *v* のメソッド呼び出し文の 3 種とする。例えば、図 1 の文 *a* は、変数 *mPowerManager* の代入文であるため、変数 *mPowerManager* の頂点に加える。また文 *g* は、変数 *mWakeLock* のメソッド呼び出し文であるため、変数 *mWakeLock* の頂点に加える。次に条件頂点

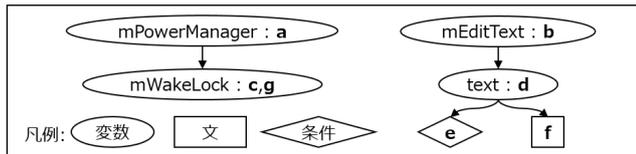


図 3 図 1 のソースコードの簡易 IVDFG

は、if 文などの制御文の条件式をひし形の頂点で表す。例えば文 *e* は、if 文の条件式であるため、条件頂点を新たに生成する。最後に文頂点は、変数頂点、条件頂点に含まれない文を長方形の頂点で表す。例えば、クラスのメソッド呼び出し文、プライベートメソッド呼び出し文、return 文などである。例えば、文 *f* はクラス *Toast* のメソッド呼び出し文であるため、文頂点を新たに生成する。

簡易 IVDFG は、変数 *v* の頂点から、変数 *v* を参照する文を含む全ての頂点への関係を定義する。例えば、変数 *text* は文 *e* と文 *f* において参照されているため、変数 *text* の頂点から条件頂点 *e*、及び文頂点 *f* への依存関係を繋ぐ。

3.3 提案ツールの設計

本節では、3.1 節で述べた機能 1、及び機能 2 に対する設計を説明する。我々の提案ツールは、次の「解析フェーズ」と「インタラクションフェーズ」から構成されている。

● 解析フェーズ

ユーザが与えたソースコードから、タグクラウド付き文集合を生成する。

● インタラクションフェーズ

ソースコードや生成したタグクラウド付き文集合から、ユーザに対して機能一覧や個々の機能におけるソースコードの記述箇所を提示する。

次節以降で、各フェーズの詳細について述べる。

3.3.1 解析フェーズ

解析フェーズでは、ユーザから与えられたソースコードからタグクラウド付き文集合を生成する。図 4 に解析フェーズの流れを示す。本フェーズは「簡易 IVDFG の生成」、「文集合の抽出」、「タグクラウド生成」の 3 つの処理から構成される。

「簡易 IVDFG の生成」では、ユーザが与えたソースコードから 3.2 節で述べた簡易 IVDFG を生成する。例えば、図 1 のソースコードからは図 3 の簡易 IVDFG が生成される。

「文集合の抽出」では、生成された簡易 IVDFG から依存関係の連鎖のある文の集合を、協調しあう文の集合とみなして抽出する。例えば、図 3 の簡易 IVDFG からは、文 *a*, *c*, *g*(左)、文 *b*, *d*, *e*, *f*(右) の 2 つの文集合が抽出される。

「タグクラウド生成」では、文集合で参照されている API の名前に基づいてタグクラウドを生成する。また API はライブラリのクラスのメソッド呼び出しや名前付き定数である。具体的には、文集合に含まれる各文から API を

抽出し、その API (所属クラスを含む) の名前からキャメルケースなどの命名規則を踏まえて単語を抽出する (例えば、“Fragment.getResources” → (“fragment”, “get”, “resources”)。抽出した単語に対し、形態素解析を行って単語の正規化を行う (例えば, (“fragment”, “get”, “resources”) → (“fragment”, “get”, “resource”)。単語の出現頻度に基づいてタグクラウドを生成する。具体的には、出現頻度が高い単語ほど、文字サイズを大きく表示する。

3.3.2 インタラクションフェーズ

図 5 のように、インタラクションフェーズでは 3.3.1 節で述べた解析フェーズで生成されるタグクラウド付き文集合とその生成元のソースコードから、ユーザに対して次の提示を行う。タグクラウド付き文集合の要素であるタグクラウドを一覧で画面に表示する。このとき、タグクラウドの一覧はそれぞれの文集合における文の構成要素が多い順に並べられる。さらにユーザが一覧から特定のタグクラウドを選択すれば、そのライブラリ機能のソースコード中の記述箇所が強調表示される。

4. 評価

提案ツールの有効性を確認するために、プロトタイプを実装した。Android のサンプルアプリケーション (Google 提供 *2) のうち、3 つのソースコードに対してプロトタイプを用いて評価を行う。評価は次の 3 つの観点をを用いる。

観点 1 文集合は、ユーザにとって理解が容易なライブラリ機能の単位となっているか。

観点 2 タグクラウドは、ユーザにとってライブラリ機能の概要を容易に把握できるものとなっているか。

観点 3 保守や拡張における使用感はどうか。

4.1 提案ツールのプロトタイプ

本プロトタイプは、Java 言語によって記述されている。解析フェーズの「簡易 IVDFG の生成」では Java のソースコードを解析するために、Eclipse Java development tools[6] の Java 用の構文解析ライブラリを用いている。また、同フェーズの「タグクラウド生成」の形態素解析には Stanford CoreNLP[5] を用いている。インタラクションフェーズは SpringBoot フレームワーク [4] を用いて Web アプリケーション *3 として実装している。

4.2 実行例 1 : AccelerometerPlay

AccelerometerPlay は加速度センサの使用法を示すサンプルであり、図 6 の動作例のように、端末を傾けると画面上の鉄球が自由に動き回るアプリケーションである。我々がソースコード “AccelerometerPlay.java” のコメントから

*2 <https://github.com/googlesamples>

*3 <http://capis.ca.info.hiroshima-cu.ac.jp:8090/codereading> で公開している

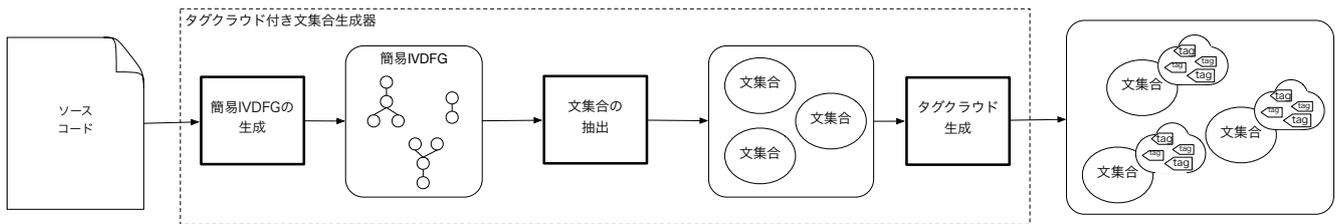


図 4 解析フェーズの処理の流れ

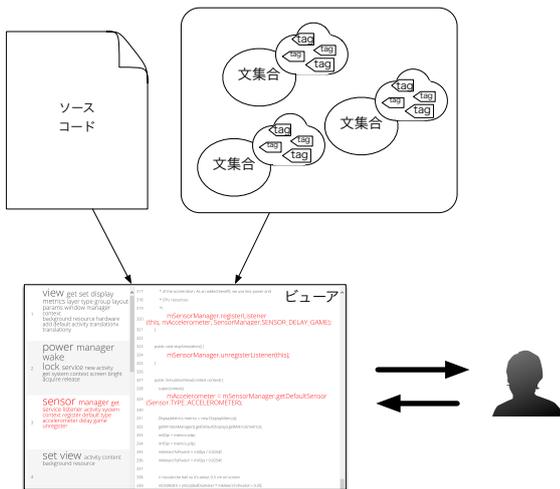


図 5 インタクションフェーズの処理の流れ



図 6 AccelerometerPlay アプリケーションの動作例

読み取った、そのコードに実装されているライブラリ機能の一部を以下に示す。

- 加速度センサ機能
端末の傾きに合わせて画面上の鉄球を動かすために用いられる。
- 端末の画面の向きを取得する機能
画面の向きを調整するために用いられる。
- ウェイクロック機能
時間経過によって画面が自動的にオフになるのを防ぐための用いられる。
- レイアウト機能
鉄球や背景の描画に用いられる。

提案ツールに AccelerometerPlay のソースコード “AccelerometerPlayActivity.java” を読み込ませた画面を図 7 に示す。以下、各観点に基づいて提案ツールを評価する。

4.2.1 観点 1

提案ツールにソースコード “AccelerometerPlay.java” を適用した結果、16 個のライブラリ機能が提示された。この 16 個のうち、上位 1~5 番目のライブラリ機能について、観点 1 に基づいて評価を行った結果を説明する。

1 番目のライブラリ機能の記述箇所は、全ソースコードの約半分の行数を占めていた。この記述箇所には、鉄球のレイアウトの生成・背景セット、鉄球の座標計算、画面情報の取得等が含まれている。また、図 8 に、2~5 番目のラ



図 7 ソースコード “AccelerometerPlayActivity.java” に対するインタクションフェーズにおけるタグクラウドの一覧表示

```

54 : public class AccelerometerPlayActivity extends Activity {
④ 56 : private SimulationView mSimulationView;
③ 57 : private SensorManager mSensorManager;
② 58 : private PowerManager mPowerManager;
⑤ 59 : private WindowManager mWindowManager;
⑤ 60 : private Display mDisplay;
② 61 : private WakeLock mWakeLock;
64 : @Override
65 : public void onCreate(Bundle savedInstanceState) {
③ 69 : mSensorManager = (SensorManager) getSystemService(SENSOR_SERVICE);
② 72 : mPowerManager = (PowerManager) getSystemService(POWER_SERVICE);
⑤ 75 : mWindowManager = (WindowManager) getSystemService(WINDOW_SERVICE);
⑤ 76 : mDisplay = mWindowManager.getDefaultDisplay();
② 79 : mWakeLock = mPowerManager.newWakeLock(
    PowerManager.SCREEN_BRIGHT_WAKE_LOCK, getClass().getName());
④ 83 : mSimulationView = new SimulationView(this);
④ 84 : mSimulationView.setBackgroundResource(R.drawable.wood);
86 : }
88 : @Override
89 : protected void onResume() {
② 96 : mWakeLock.acquire();
④ 99 : mSimulationView.startSimulation();
100 : }
102 : @Override
103 : protected void onPause() {
④ 111 : mSimulationView.stopSimulation();
② 114 : mWakeLock.release();
115 : }
117 : class SimulationView extends FrameLayout implements SensorEventListener {
③ 125 : private Sensor mAccelerometer;
312 : public void startSimulation() {
③ 320 : mSensorManager.registerListener(
    this, mAccelerometer, SensorManager.SENSOR_DELAY_GAME);
321 : }
323 : public void stopSimulation() {
③ 324 : mSensorManager.unregisterListener(this);
325 : }
327 : public SimulationView(Context context) {
③ 329 : mAccelerometer = mSensorManager.getDefaultSensor(
    Sensor.TYPE_ACCELEROMETER);
346 : }
358 : @Override
359 : public void onSensorChanged(SensorEvent event) {
⑤ 371 : switch (mDisplay.getRotation()) {
388 : }
389 : }
428 : }
429 : }

```

図 8 図 7 の 2~5 番目のタグクラウドのライブラリ機能の文集合の記述箇所を示した “AccelerometerPlayActivity.java” のソースコード片

イブラリ機能の文集合の記述箇所を抜粋したソースコード片を示す。なお、図 8 の行番号の左の丸付き数字は、対応するライブラリ機能の文集合の記述箇所を示している。図中の丸付き数字のライブラリ機能の記述箇所には、それぞれ、ウェイクロック機能の開始・終了、加速度センサの測定開始・測定終了、画面のレイアウトの生成・背景セット、端末の画面の向きの取得等の記述が含まれている。

2, 3, 4, 5 番目のライブラリ機能は、4.2 節に列挙したソースコード中のコメントの情報から得た機能とほぼ一致する。一方で、1 番目のライブラリ機能の中にはさらに複数の機能（鉄球のレイアウト、座標計算等）が含まれており、2, 3, 4, 5 番目のライブラリ機能との粒度に差があると考えられる。提案ツールがユーザに提示する文集合は、概ね、ユーザにとって理解が容易なライブラリ機能の単位となっている可能性が高いと考えられる。

4.2.2 観点 2

図 7 の提案ツールの実行画面における 2~5 番目のタグクラウドについて、観点 2 に基づいて評価を行う。

2~5 番目のタグクラウドのライブラリ機能について、その文集合の記述内容から概要を表すのに妥当な単語を考える。2 番目のタグクラウドのライブラリ機能はウェイクロック機能に関する記述が含まれ、“wake”, “lock” が相応しいと考えられる。同様に、3, 4, 5 はそれぞれ加速度センサ、レイアウト、端末の画面の向きの取得に関す

図 9 ソースコード “AccelerometerPlayActivity.java” に対するインタラクションフェーズにおける 3 番目のタグクラウドの選択後

る記述であり、“sensor”, “accelerometer”, (“layout”), (“display”, “rotation”, “get”) が相応しいと考えられる。

これらの各タグクラウドの単語は、図 7 の提案ツールの実行画面におけるタグクラウドに含まれているものが多いが、上位に表示されているものは少ない。提案ツールにおいてユーザに提示するタグクラウドは、概ね妥当であるが改善の余地があるといえる。

4.2.3 観点 3

提案ツールについて、観点 3 に基づいて評価を行う。具体的には、保守、拡張の場面の一例において提案ツールを試用し、使用感を評価する。

- 保守 AccelerometerPlay[2] のソースコードに対し、加速度センサの取得間隔を調整する修正を加えることを考える。提案ツールに適用した結果である図 7 の画面左の一覧を上から順に眺め、3 番目のタグクラウドがセンサの管理や取得に関する機能だと推測できる。3 番を選択すれば、図 9 のように、加速度センサの機能の記述箇所 (57, 69, 125, 320, 324, 329 行目) が強調される。これらの記述箇所のみを確認すれば、変更すべき箇所 (320 行目の名前付き定数 *SensorManager.SENSOR_DELAY_GAME*) を特定できる。これより、短時間で修正を行えると期待できる。

- 拡張 開発者が実装中のソースコードに対し、AccelerometerPlay[2] を参考に、加速度センサの機能を追加することを考える。保守と同様の手順で 3 番のタグクラウドを選択すると、加速度センサの機能の記述箇所 (57, 69, 125, 320, 324, 329 行目) が強調される。これらの記述箇所を踏まえ、実装目的に合わせて修正して実装中のソースコードに流用できる。これより、加速度センサの機能を短時間で追加実装できると考えられる。

以上により、保守や拡張に提案ツールを用いると、少ない作業、短時間でこれらを行うことが出来ると考えられる。

```

connectivity get
network info type
manager service active
activity system context is connected
wifi mobile

context get string

fragment manager
find by id activity get support
    
```

図 10 ソースコード “MainActivity.java” に対するインタラクションフェーズにおけるタグクラウドの一覧表示

```

get camera
capture
request surface size im
builder session activity
manager set rotation rect
callback display window de
handler reader sparse int a
dialog builder
bundle alert
fragment
set string get argume
base activity on click liste
put create positive button mes
finish r ok interface
chic manager get
manager string dialog show
    
```

図 11 ソースコード “Camera2BasicFragment.java” に対するインタラクションフェーズにおけるタグクラウドの一覧表示

4.3 実行例 2 : BasicNetworking

BasicNetworking は、ネットワーク接続を確認する方法を示すサンプルアプリケーションである。このアプリケーションのソースコード “MainActivity.java” のコメント情報から、このソースコードには、ネットワークの接続状況を確認する機能、フラグメント機能 (イントロダクションの表示用とログの表示用) 等が実装されていることが分かる。

図 10 に “MainActivity.java” に対するインタラクションフェーズにおける実行例を示す。観点 1 において各機能が文集合として抽出されることを、観点 2 において各機能を表す単語が上位に表示されることを、観点 3 において各機能の記述を少ない作業で把握できることを確認した。

4.4 実行例 3 : Camera2Basic

Camera2Basic は、カメラの使用法を示すサンプルアプリケーションである。このアプリケーションのソースコード “Camera2BasicFragment.java” のコメント情報から、このソースコードには、カメラ機能、ダイアログフラグメント機能 (カメラ撮影許可の要求用とエラーメッセージ表示用) 等が実装されていることが分かる。また、カメラ機能には、プレビュー画面の準備や開始、写真の撮影、写真の保存といったステップが存在することが読み取れる。

図 11 に、 “Camera2BasicFragment.java” に対するインタラクションフェーズにおける実行例を示す。観点 1 において、各ダイアログフラグメント機能が文集合として抽出

されることを確認した。一方で、カメラ機能の文集合は、各ステップの記述を全て含んでいるためサイズが大きく、理解の容易な機能の単位になっていないと限らない。

また、撮影情報のオブジェクトに自動フラッシュ情報を付与する文集合が、カメラ機能とは別の機能として抽出されている。これは、自動フラッシュの機能が、プライベートメソッドとして切り出されているためである。この文集合はカメラの機能に含まれることが望ましい。

観点 2 において、各タグクラウドについて、カメラ機能を表す (“camera”) は上位に表示される一方で、ダイアログフラグメント機能を表す (“dialog”, “fragment”) のうち、“fragment” が上位に表示されないことを確認した。

観点 3 において、各機能の記述を少ない作業で把握できることを確認した。

5. 考察

5.1 評価の観点 1 に関する考察

4.2.1 節では、図 7 の提案ツールの実行画面における 1 番目のタグクラウドのライブラリ機能の文集合は、ユーザにとって理解が容易なライブラリ機能の単位となっていない可能性があることを述べた。その原因の一つに、文集合の抽出において、ユーザにとって理解が容易な単位の文集合同士が互いに協調しあって 1 つの文集合となり、1 つのライブラリ機能として抽出されている点が挙げられる。

4.2.1 節では、1 番目のタグクラウドのライブラリ機能について、その文集合にレイアウトに関する記述、座標計算に関する記述、画面情報に関する記述が含まれていることについて触れた。これら 3 つを機能とすると、画面情報から取得した解像度を鉄球の座標計算に使用し、鉄球のレイアウトの座標指定にこの計算結果を使用していることから、3 つの機能は互いに協調しあっていると考えられる。それにより、この 3 つの機能の文集合同士が互いに協調しあって 1 つの文集合となり、1 つのライブラリ機能として抽出されたと考えられる。

この問題の解決策として、文集合を階層的に分割し、より小規模な機能をユーザに提示することが挙げられる。

5.2 評価の観点 2 に関する考察

4.2.2 節における観点 2 の評価では、図 7 の提案ツールの実行画面におけるタグクラウドは、ユーザにとってライブラリ機能の概要を容易に把握できるものとなっていない可能性があることについて述べた。その原因の一つに、各ライブラリ機能の振る舞いに関する単語よりも、準備に関する単語の方が上位に表示されている点が挙げられる。

図 12 にこのソースコードの簡易 IVDFG を示す。図 12 中の 4 つの文集合は、それぞれタグクラウド 2~5 番目のライブラリ機能に対応している。どのライブラリ機能の文集合についても、下流の頂点の文集合は機能の振る舞いに

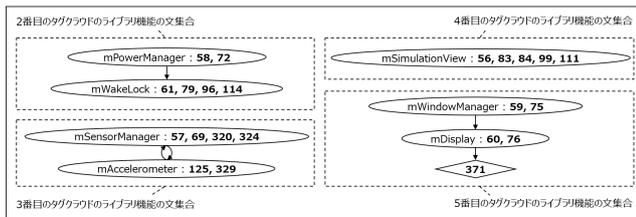


図 12 AccelerometerPlayActivity.java の簡易 IVDFG

関する記述が多く、上流の頂点の文集合は機能に関する記述が多い。例えば、2 番目のライブラリ機能の文集合について、ウェイクロック機能の振る舞いである機能開始・終了の処理は 96, 114 行目であり、振る舞いのための準備に関する処理は 58, 61, 72, 79 行目である。

4.2.2 節では、各ライブラリ機能の概要を表すのに相応しい単語は、2 番目のタグクラウドが (“wake”, “lock”), 3 番目のタグクラウドが (“sensor”, “accelerometer”), 4 番目のタグクラウドが (“layout”), 5 番目のタグクラウドが (“display”, “rotation”, “get”) であることを示した。これらの単語についても、やはりライブラリ機能の文集合の下流の頂点の文集合に記述されていることが多い。

この問題点の解決策として、タグクラウドの単語の順位を計算する際に、簡易 IVDFG における単語の記述箇所による重みづけを行うことが挙げられる。

5.3 評価の観点 3 に関する考察

4.2.3 節では、保守や拡張に提案ツールを用いると、少ない作業、短時間でこれらを行うことが出来る可能性があることについて述べた。さらに少ない作業、短時間でこれらを行うための改善策として、タグクラウドのライブラリ機能のソースコード中の記述箇所を強調表示する際に、対応するスクロールバーの箇所もそれに合わせて色付けすることが挙げられる。これにより、記述箇所を素早く把握することが出来ると考えられる。他にも、提案ツールを Eclipse 等のプラグインとして実装することが挙げられる。これにより、Eclipse 等が提供するソースコード理解支援と組み合わせることで、より高度な支援をすることが出来ることと期待される。

6. 関連研究

吉田らは、密接に協調しあうブロックの集合を機能として抽出する手法を提案した [7]。具体的には、ソースコード中の各ブロックで使用されている変数を抽出し、変数の共有度の高いブロックの集合を機能として抽出する。例えば、2 つのブロックで使用された変数群の強調度合いを表す指標を導入し、その指標により機能をブロック集合を決定する。これにより、任意に設定した協調度以上のブロック集合を機能として抽出することが出来る。機能の記述の粒度は、吉田らの手法はブロックであり、提案ツールは文

である。吉田らの手法は機能を正確に抽出できない可能性があるが、提案ツールはこの問題点を解決する。一方で、ブロックは開発者が意図したソースコード中の構造であり、吉田らの手法はこれを考慮したソースコード理解の支援を行うことが出来る。

平山らは、密接に協調しあう文の集合を機能として抽出する手法を提案した [9]。具体的には、ソースコードから抽出した PDG において、各変数をスライス基点とするプログラムスライスを抽出し、文の共有度の高いプログラムスライスの集合を機能として抽出する。平山らの手法はメソッドを跨いで記述された機能を抽出することが出来ないが、提案ツールはこの問題点を解決する。

7. まとめと今後の課題

本研究では、プログラムの保守や拡張において特定のライブラリ機能の記述箇所の把握が必要となる場面に対し、ソースコード理解を支援するツールの設計・実装を行った。提案ツールはオブジェクトを介して相互に作用しあう API の関係に着目し、データ依存グラフをベースとした文集合抽出手法により、メソッド境界を超えて密接に関連する文集合を抽出してユーザに提示することが出来る。評価により、提案ツールを用いると特定の機能の記述箇所の把握に要する負担を軽減できる可能性があることを確認できた。

今後の課題として、機能を階層的に分割し、より小規模な機能をユーザに提示すること等が挙げられる。

参考文献

- [1] Ferrante, J., Ottenstein, K. J. and Warren, J. D.: The Program Dependence Graph and Its Use in Optimization, *ACM Trans. Program. Lang. Syst.*, Vol. 9, No. 3, pp. 319–349 (online), DOI: 10.1145/24039.24041 (1987).
- [2] Google Inc: AccelerometerPlay, <https://github.com/googleamples/android-accelerometerplay>.
- [3] Google Inc: SensorManager, <https://developer.android.com/reference/android/hardware/SensorManager.html>.
- [4] Pivotal Software, Inc.: SpringBoot, <https://projects.spring.io/spring-boot/>.
- [5] Stanford NLP Group open source software: Stanford CoreNLP, <https://stanfordnlp.github.io/CoreNLP/>.
- [6] The Eclipse Foundation: Eclipse JDT, <https://www.eclipse.org/jdt/>.
- [7] 吉田則裕, 木下正喬, 飯田 元: プログラム理解のための凝集度に基づく機能候補抽出, 日本ソフトウェア学会大会論文集, Vol. 28, pp. 1–6 (オンライン), 入手先 (<https://ci.nii.ac.jp/naid/40020660037/>) (2011).
- [8] 柳 慶吾, 石尾 隆, 井上克郎: ソフトウェア部品利用例抽出のためのデータフロー解析手法の提案と評価, 情報処理学会研究報告. ソフトウェア工学研究会報告, Vol. 2010, No. 29, pp. 1–8 (2010).
- [9] 平山力地, 吉田則裕, 飯田 元: スライスに基づく凝集度を用いて自動分割を行うプログラム理解支援手法, 電子情報通信学会技術研究報告. SS, ソフトウェアサイエンス, Vol. 112, No. 164, pp. 127–132 (オンライン), 入手先 (<https://ci.nii.ac.jp/naid/110009626481/>) (2012).