API呼び出し記述支援手法に向けての ソースコードからマイニングした情報の特徴分析

西本 匡志[1,a)]　川端 英之[1,b)]　弘中 哲夫[1,c)]

概要：複雑な知的活動であるソフトウェア開発を効率化する試みには様々なものがある．しかし，要求に対する仕様の明確化やプログラム部品の詳細設計，実装，テストなど，担当者の開発経験によって開発効率が左右される要因は未だに多く存在する．近年は多様なフレームワークやライブラリの組み合わせによるアプリケーション構築が不可欠であり，開発者にはAPIの使い方への熟練が求められる．これに対し，多くのソースコードからAPI使用に関するノウハウを抽出して，APIの呼び出し系列の記述を補助する仕組みが研究されている．我々はさらなる開発効率向上を目指し，開発者の意図を汲み取った上で支援する手法を検討している．我々は，開発者の手元にあるコードの編集状況や開発者が与えるキーワードから，実現しようとしている機能や追加が必要な処理を推測し，コードの修正案を提示することを目指している．本発表では，API呼び出し記述支援システムで用いる知識基盤のあり方について，Androidの Google Sampleコードを分析した結果を踏まえて議論する．

キーワード：プログラミング支援ツール，データマイニング，API

1. はじめに

現在の多種多様なソフトウェアの開発において，様々なフレームワークやライブラリの組み合わせによるアプリケーション構築は不可欠であり，開発者にはApplication Programming Interface(API)の使い方への熟練が求められる．しかしながら，多様かつ大量のAPIの熟知と，それらを適切に組み合わせる作業は，ソフトウェア開発者にとって大きな負担となっている．

開発者がAPIを使いこなすためには，API仕様のドキュメントやAPIを用いて実装されたサンプルコードを読んでAPIの使い方を理解する．また技術情報共有サービス (e.g. Qiita[3]，Stack Overflow[6]) からAPIの使い方に関する情報を得ることもある．しかし，これらの方法において，開発者は多くのAPIに関する情報を理解したり，多くの情報の中から開発者自身にとって必要な情報を取捨選択したりしなければならず，開発者への負担が大きい．

我々はこのような開発者の負担を軽減させることによってソフトウェアの開発効率向上を目指しており，「API呼び出し記述支援手法 [7]」を提案して，本手法に基づくツールの開発に取り組んでいる．我々の手法は開発者の意図を汲み取ってAPI呼び出しの組み合わせに基づくプログラミングをサポートする仕組みである．我々の手法に基づくAPI呼び出し記述支援ツール (以下，Aツールと

[1] 広島市立大学
[a)] nishimoto.masashi@ca.info.hiroshima-cu.ac.jp
[b)] kawabata@hiroshima-cu.ac.jp
[c)] hironaka@hiroshima-cu.ac.jp

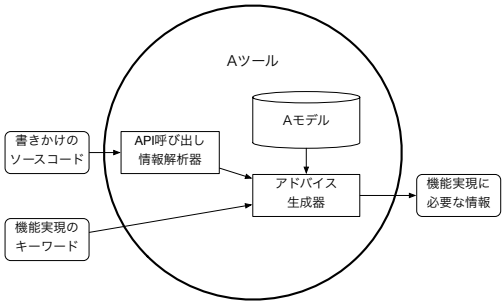

図 1 A ツールの構造

```
1 public class MainActivity extends Activity
2   implements SensorEventListener {
3     private SensorManager mSensorManager;
4     private TextView mTextView;
5
6     @Override
7     public void onCreate(Bundle savedInstanceState) {
8         super.onCreate(savedInstanceState);
9         setContentView(R.layout.main);
10
11        mTextView = (TextView)findViewById(R.id.value);
12    }
13 }
```

図 2 Android アプリケーションのプログラム例

呼ぶ) は，書きかけのソースコードやユーザが与えるキーワードを踏まえて開発状況を把握し，要求される仕様に対してどのような機能が欠けているかを判断できる．また，どこにどのような記述を挿入する必要があるかなどの情報を書きかけのコードに即して指示する機能を備える．

API 呼び出し記述支援手法の実現には，様々な機能の実現に関する API の使い方を集めた知識基盤となる API 呼び出し記述モデル (以下，A モデルと呼ぶ) の構築が必要である．例えば，Github[2] や Bitbucket[1] に代表されるソースコードリポジトリから A モデルを自動生成したり，その際に使用するソースコードに不具合が含まれていないかを自動的に確認したりすることが求められる．

本稿では，ソースコードリポジトリからの A モデルの構成方式について述べ，実際にいくつかの方式で構築した A モデルに含まれる情報の特徴を分析し，A ツールに活用できるかについて議論する．

2. API 呼び出し記述支援手法 [7]

我々は，書きかけのソースコードに対してユーザが実現したい機能を表すキーワードから，ユーザの意図を汲み取り，記述すべき API 呼び出し (の系列) を提示する手法を提案している．提案手法に基づく A ツールの構造や処理の流れについて，次節から説明する．

2.1 A ツールの構造と処理の流れ

A ツールの構造の概略を図 1 に示す．A ツールは次の要素から成る．

- API 呼び出し情報解析器
- アドバイス生成器
- A モデル

開発者は図 2 に示すプログラムに対して「Sensor」機能の実現を望んでいる．このような状況において，A ツールは A モデル内の Sensor に関する情報とプログラムを見比べて，図 2 の 10 行目に `getSystemService` の API 呼び出しが足りないことをアドバイスする．また `onResume` や `onSensorChanged` に関する処理自体が足りないこともアドバイスする．つまり，A ツールは API 呼び出しの順序のパターンに着目している．

2.2 API 呼び出しの順序のパターンに着目したアドバイス生成の流れ

A モデルは，アプリケーション構築に用いられる様々な機能の実現における標準的な API 呼び出し系列 (以下，A 系列と呼ぶ) 集合の集合である．A 系列集合はユーザ定義クラス内に標準的に実装されるべき A 系列を集めたものである．それら個々のメソッド中で呼び出されるべき API の名前情報のリストが A 系列として把握される．なお，ここで述べた機能とは，「カメラで写真を撮る」や「加速度センサーの値を取得する」など単独のアプリケーションとして動作可能な最小構成のことを示す．

2.1 節で述べた例に対して，API 呼び出しの順序のパターンに着目した上での A ツールの動作の流れを説明する．図 3 左は図 2 のソースコードから抽出した A 系列集合である．また，図 3 右は A モデルに含まれるセンサーの機能を実現するために必要になる A 系列集合である．書きかけのコードから抽出した図 3 左の A 系列 1 を眺めて，セ

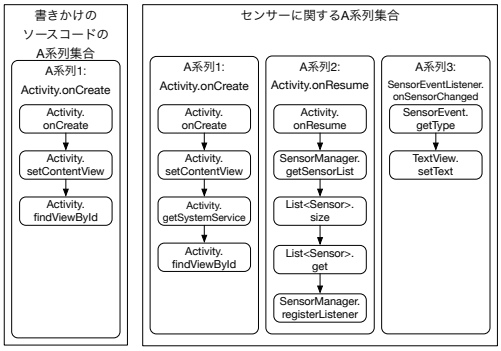

図 3 アドバイスの動作例に関する A 系列の集合

ンサーの機能を実現するための図 3 右の各 A 系列に含まれる API 呼び出しが全て存在するかを確かめる．書きかけのコードの A 系列集合には，`getSystemService` や A 系列 2，3 自体が存在していないため，アドバイスする．

2.3 A ツールの実現に向けた検討項目

我々は API 呼び出し記述支援手法に基づく初期の A ツールのプロトタイプを実装している [7]．さらなる A ツールの効果的な実現に向けて，以下の項目の検討が必要である．

(1) 様々な機能実現に必要な A モデルの管理・収集方式
(2) ユーザの意図を A ツールに把握させるインタラクションの方式
(3) A ツールの実用性に不可欠な高速性や提示情報の精度

(1) に関しては，ライブラリやフレームワークのマニュアル等から手作業で A モデルを作成しているため，様々な機能の実現における情報量が少ない．そのため，A ツールにおける様々な機能に対するアドバイスの有効性評価が充分に行えていない．

(2) に関しては，キーワードのみによってユーザの意図を把握しようとしているが，ソースコード中に現れる識別子名，特にクラスや変数名等からの把握が望まれる．

(3) に関しては，ユーザが A ツールを使うときの処理速度，メモリ使用量の最適化や，不足した API 呼び出しの提示情報の精度向上のための処理方式が必要である．

3. A モデルの構成方式

我々は，A ツールが開発者に対して幅広く，実用性の高いアドバイスを行えるように，ソースコードリポジトリから A モデルを構成するための方式について検討した．この検討は 2.3 節で挙げた検討項目 (1) に対応するものである．

3.1 効果的な A モデルの持つべき性質

効果的なアドバイスを行うための A モデルには，1 つの機能を実現するために必要となる A 系列の数や個々の A 系列に含まれる API 呼び出しの数は多い方が良いと考えている．なぜなら，書きかけのソースコードとの類似性を判断する材料が増えて，精度の高いアドバイスが期待できるからである．

また A ツールはユーザの意図をキーワードから把握した上で，それに合った的確なアドバイスを行うため，ユーザが実現を望む 1 つの機能が 1 つの A 系列の集合で表現できた方が良いと考えている．なぜなら，A モデル内からユーザが望む機能に関する A 系列集合のみを得ることができ，その機能の実現にあたって過不足ないアドバイスが行えるからである．

3.2 A モデルの構成に用いる方法

我々は 3.1 節で述べた効果的な A モデルの構築に向けて，次の 2 つの方法を用いる．

- 1 つの A 系列はより多くの API 呼び出しで構成される方が良いと考えられるため，複数の A 系列間の制御フローを把握してそれらの A 系列に API 呼び出しの順序関係があれば，1 つの A 系列に融合する (3.2.1 節参照)
- 1 つの A 系列集合はユーザが望む単独の機能のみで構成される方が良いと考えられるため，クラス毎の A 系列集合に分離する (3.2.2 節参照)

3.2.1 A 系列の融合

開発者はソースコードの可読性や再利用性を高

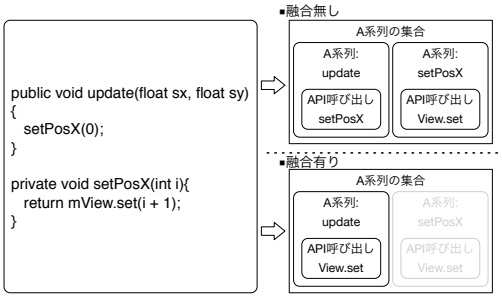

図 4 例：プライベートメソッドに関する融合

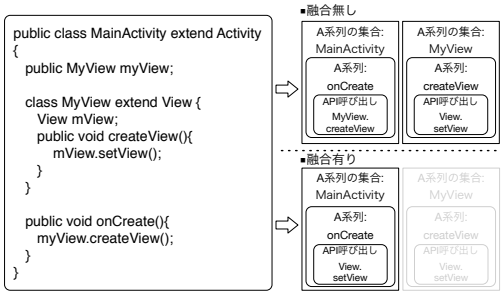

図 5 例：インナークラスに関する融合

めるために，プライベートメソッドやインナークラスなどを用いて処理を抽象化・モジュール化することがある．これに対して，アプリケーション実行時に API 呼び出しが発行される順序を踏まえれば，ソースコード上では A 系列同士が独立しているように見えても，実は関連がある「API 呼び出し順序のパターン」が見えてくる．そこで，A モデルを構成する場合はアプリケーションの実行時の状態を踏まえるため，プライベートメソッド，もしくはインナークラスに含まれる A 系列 (A) が別の A 系列 (B) で呼び出されている場合，A 系列 (B) の該当箇所に A 系列 (A) をインライン展開して，1 つの A 系列に融合する．

図 4 にプライベートメソッドに関する A 系列の融合の例を示す．図 4 中のソースコードにはメソッド update と setPosX が定義されている．このうち，setPosX はプライベートメソッドである．図 4 の右下に示された A 系列集合は図 4 の右上と異なり，setPosX の A 系列が update の A 系列に融合されていることが分かる．

また，図 5 にインナークラスに関する A 系列の融合の例を示す．図 5 中のソースコードにはクラス MainActivity と MyView が定義されている．このうち，MyView はインナークラスである．図 5 の右下に示された A 系列集合は図 5 の右上と異なり，MyView の A 系列集合自体が削除され，MainActivity に A 系列が融合されていることが分かる．

3.2.2 A 系列集合の分離

ソースコードから取り出した 1 つの A 系列集合

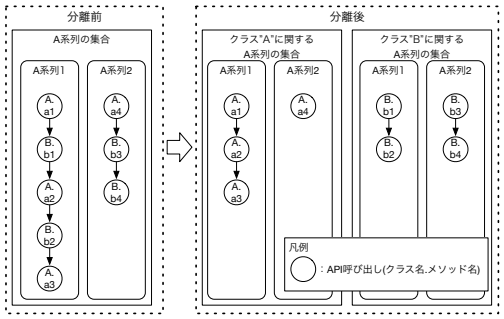

図 6 例：A 系列集合の分離

には，複数の機能が混在している場合が想定される．例えば，位置情報付きの写真を撮影する機能は，カメラで写真を撮る機能と GPS を用いて位置情報を測定する機能の組み合わせで実現されている．開発者は位置情報付きの写真を撮影する機能の実現を望むこともあれば，写真を撮る機能，もしくは位置情報を測定する機能のどちらかのみの実現を望むこともある．そこで，開発者が機能の実現を望む最小構成で A 系列集合を整えるために別々の機能として分離する．

開発者が望む機能の最小構成で分離する方法は様々なものがありうる．例えば，API の属するクラスに基づく分離は，機能毎に A 系列集合を整理する方法として妥当ではないかと考えられる．

図 6 に，ある A 系列集合を API の属するクラスに基づいて分離した例を示す．分離前の A 系列集合は，クラス A と B に所属する API 呼び出しが混在した 2 つの A 系列から構成される．分離後はクラス A，およびクラス B に関する API 呼び出しのみでそれぞれ構成された A 系列集合となる．

表 1　分析対象の情報

項目	数
プロジェクト数	159
ソースコード (Java) の総ファイル数	1794
Android に関する総 API 呼び出し数	18027

4. リポジトリマイニングに基づく各構成方式の効果，および，議論

4.1 分析目的・内容

ソースコードリポジトリから構成した A モデルが 3.1 節の条件を満たした A 系列集合になるかどうかを以下の項目に着目して分析する．また A 系列の融合や分離による効果についても分析する．

分析 1　個々の A 系列に含まれる API 呼び出しの数は平均的にどのくらいあるのか？

分析 2　ある機能を実現するために必要となる A 系列集合に含まれる API 呼び出しの数は平均的にどのくらいあるのか？

4.2 分析対象，および，A モデルの各構成方式

ソースコードリポジトリからの収集・分析対象となるプロジェクト (ソースコード) に関する情報を表 1 に示す．対象プロジェクトは Github に登録された Google 実装の Android のサンプルアプリケーションである．なお，API 呼び出しは Android の標準フレームワークに含まれる API のみを収集する．

分析に用いる A モデルの各構成方式を以下に示す．融合の有無は，3.2.1 節のプライベートメソッドとインナークラスの両方に関する A 系列の融合を適用するか否かである．また分離の有無は，3.2.2 節のクラス毎に A 系列集合の分離を適用するか否かである．

- 方式 1：融合無・分離無
- 方式 2：融合無・分離有
- 方式 3：融合有・分離無
- 方式 4：融合有・分離有

4.3 分析 1 の結果と考察

表 2 は個々の A 系列に含まれる API 呼び出し数のパターン毎に全体に占める A 系列数の割合を，各構成方式とともに示したものである．

表 2　1 つの A 系列に含まれる API 呼び出し数の割合

構成方式	A 系列に含まれる API 呼び出し数			
	1	2	3	その他
方式 1	34%	20%	14%	32%
方式 2	65%	21%	6%	8%
方式 3	18%	19%	10%	53%
方式 4	53%	22%	9%	16%

方式 1，2，4 では，各パターンのうち，単独の API 呼び出しからなる A 系列の占める割合が最も多い．また，融合を行って A モデルを構成すると，短い A 系列の割合が減少する．方式 1 と 3，2 と 4 について，それぞれ単独の API 呼び出しからなる A 系列の占める割合を比較すると，このことが明確に表れている．

方式 3 は，その他 (4 つ以上) の API 呼び出しのパターンの占める割合が 53%と全構成方式の中で最も高く，効果的なアドバイスを行うことが可能な構成方式であると考えられる．

方式 2 と 4 は，それぞれ方式 1 および 3 で得られた A 系列集合に対して分離を適用したものである．方式 3 と方式 4 を比較すると，単独の API 呼び出しからなる A 系列の占める割合が 18%から 53%へと増加し，他の A 系列の割合が相対的に減少している．この結果を見ると，クラスに基づく分離は効果的なアドバイスを行うための A モデル構成には不向きである可能性がうかがえる．しかし，クラスに基づいて分離された A モデルは，実装しようとする機能についての細かい単位でユーザの意図を反映したアドバイスの生成には有効ではないかとも考えられる．このことを明らかにするには別の尺度による評価を行って，A 系列集合の分離がアドバイスへ及ぼす効果を確かめる必要がある．また，A 系列集合に対して「クラス毎」に分離するのではなく，別の指標 (e.g. データの依存関係など) に基づく分離を検討して，有効性を確かめる必要がある．

4.4 分析 2 の結果と考察

表 3 は 1 つの A 系列集合に含まれる API 呼び出し数のパターン毎に全体に占める A 系列集合の

表 3　1 つの A 系列集合に含まれる API 呼び出し数の割合

構成方式	A 系列集合に含まれる API 呼び出し数			
	1	2	3	その他
方式 1	9%	13%	7%	71%
方式 2	44%	22%	9%	24%
方式 3	4%	18%	7%	70%
方式 4	43%	23%	10%	25%

割合を，各構成方式とともに示したものである．

A ツールが A 系列集合と書きかけのソースコードを比較して同じ API 呼び出しがある場合にアドバイスを生成することを考慮すると，単独の API 呼び出しからなる A 系列集合はアドバイスに活用することができない．

表 3 の API 呼び出し数が 1 のパターンが占める割合に注目すると，方式 1 が 9%で，A 系列の融合を行った方式 3 が 4%と減少して，アドバイスに活用できる情報の割合が増えていることが分かる．しかしながら，A 系列集合の分離を行った方式 2 は方式 1 と比べて，9%から 44%へと大幅に増加して，活用できる情報の割合が減っている．

なお，1 つの API 呼び出しには，`android.util.Log` クラスのメソッド `d` や `android.widget.TextView` クラスのメソッド `setText` などの単独で使用される API が多かった．そのため，単独で使用される API に対しては，A ツールによる個別のサポートを行うのが妥当な対応であると考えられる．

5. 関連研究

Raychev ら [5] の SLANG は既存のプロジェクトより API 呼び出しの前後関係を抽出して，N-Gram や Recurrent Neural Network(RNN) を用いて学習する．そして編集中のコードにおいて，次に呼び出される関数を予測するシステムを提案した．また，Pham ら [4] は API 呼び出しの前後関係について Hidden Markov Model を用いてモデル化した HAPI を提案した．これらの研究に対し，本研究は，API の呼び出しの前後関係だけではなく，どのような関数から呼び出されるかも含めてモデル化することにより，どこにどのような処理を挿入すべきかを判断可能にしようとしている．

6. まとめと今後の課題

A モデルの構成方式を検討して，実際のソースコードリポジトリからモデルを構築し，アドバイスに活用できるかを分析した．

Google の Android サンプルコードに対して A 系列の融合を行って A モデルを構築した結果，単独の API 呼び出しから成る A 系列の全体に占める割合が約 18%までおさえた A モデルが構築可能であることが分かった．

以下は今後の課題である．

- クラス毎に A 系列の分離を行っているが，別の観点 (e.g. データの依存関係) によって分離した場合，アドバイスに活用可能な A 系列の割合を確認する．
- A モデルに含まれる各 A 系列集合に対して，API 呼び出しの順序関係が似ている集合は統合したり，順序関係に重みを持たせたりすることにより，アドバイスの精度を高める．
- A ツールの有効性を客観的に評価する指標を定め，それに基づいた評価を行う．

謝辞　本研究の一部は，総務省の平成 28 年度 独創的な人向け特別枠「異能 (Inno)vation」プログラム」の助成を受けている．

参考文献

[1] Atlassian: BitBucket, https://bitbucket.org.
[2] GitHub, Inc.: Github, https://github.com/.
[3] Increments Inc.: Qiita, https://qiita.com/.
[4] Pham, H. V., Vu, P. M., Nguyen, T. T. et al.: Learning API usages from bytecode: a statistical approach, *Proceedings of the 38th International Conference on Software Engineering*, ACM, pp. 416–427 (2016).
[5] Raychev, V., Vechev, M. and Yahav, E.: Code completion with statistical language models, *ACM SIGPLAN Notices*, Vol. 49, No. 6, ACM, pp. 419–428 (2014).
[6] Stack Exchange Inc.: Stack Overflow, http://stackoverflow.com/.
[7] 西本匡志，川端英之，弘中哲夫：キーワードと書きかけのコードから開発者の意図を推測する API 呼び出し記述支援手法の検討，日本ソフトウェア科学会第 33 回大会講演論文集 (2016).