特別研究報告

題目

API 呼び出し列の差分を利用した Android アプリケーション比較ツールの試作

指導教員
井上 克郎 教授

報告者
神田 哲也

平成 23 年 2 月 16 日

大阪大学 基礎工学部 情報科学科

平成 22 年度 特別研究報告

API 呼び出し列の差分を利用した Android アプリケーション比較ツールの試作

神田 哲也

内容梗概

アプリケーションの利用者・開発者双方にとって，複数の類似するアプリケーションの中から必要とする機能を持ったものを選択することは重要である．利用者にとっては，アプリケーションを選択する際に，類似する機能を持つアプリケーション間でどのような機能の違いがあるのかを知ることが選択の際に 1 つの手掛かりとなる．開発者にとっても，既存のアプリケーションに機能追加を行う場合に，どのアプリケーションがどの機能を実装しているのか，差異を理解することが重要である．このため，アプリケーション同士の違いを明らかにする必要がある．

類似したアプリケーションの持つ機能を比較する方法としては，実際に利用して比べる事や，ドキュメントを比較する事，ソースコードを比較する事などが考えられる．しかし，これらの方法には，労力がかかりすぎる，比較基準の設定が難しい，といった問題点があり，アプリケーション同士を理解しやすい形で比較することは容易ではない．

そこで本研究では，ソースコード中の API 呼び出し列に着目したアプリケーションの比較手法を提案する．アプリケーションが機能を実装する際には，あらかじめ用意された API の中から必要なものを選択し，組み合わせて利用することが一般的である．このことから，アプリケーションにおいて利用される API の種類や順序と，機能との間には関連性があると考えられる．そこで，アプリケーション中に出現する API 呼び出し列の種類の違いを抽出し，アプリケーション間の機能の違いを明らかにする手法を提案する．さらに，提案手法を実装し，手法の有効性を確認する実験を行った．実験対象のプラットフォームとしては，携帯端末向けのプラットフォームである Android[1] を用いた．そして，Android 用アプリケーションを対象に API 呼び出し列を抽出し，アプリケーションを比較するためのツールを試作した．ツールを用いたケーススタディの結果，提案手法によりアプリケーションの機能の比較が実現できていることを確認した．

主な用語

ソフトウェア比較
ソースコード解析
API 呼び出し
Android

目 次

1　まえがき

アプリケーションの利用者・開発者双方にとって，複数のアプリケーション同士の機能の違いを明らかにすることは重要である．

類似する機能を持ったアプリケーションは，付随する機能の違いや，同じプロジェクトからの派生などにより複数存在することが多い．利用者にとっては，利用するソフトウェアを選択する際，ソフトウェアに利用したい機能が存在するかを確認したいという要求がある．開発者にとっても，既存のアプリケーションに機能追加を行う際，今あるソフトウェアがどのような機能を持つかを知る必要がある．

類似したアプリケーションを機能面で比較する方法としては，(1) 実際に利用する，(2) 開発者側が用意したドキュメント同士を比較する，(3) ソースコード同士を比較するなどの方法が考えられるが，それぞれの方法において問題点がある．(1) に関しては，利用するだけではすべての機能を把握できない可能性が有り，また多数のアプリケーションを比較したい場合に負担が大きい．(2) の場合には，ドキュメントの形式が統一されていない場合に比較が難しい点や，ドキュメントに不備があったりそもそもドキュメントが存在しない場合，意味のある比較ができないという点が問題になる．(3) のソースコードを比較する方法としては，Unix diff が考えられる．しかし，Unix diff のような単純な比較では，使用言語と実現している機能が同じアプリケーション同士でも，設計が違う時にはほぼ全体が差分として抽出されてしまうという問題がある．また，機能とは関係のない差分も多く出力されてしまうことになり，理解が難しくなり現実的ではない．

そこで本研究では，特定のプラットフォームに向けたアプリケーションを開発する際に用いられる API 呼び出しに着目した．API 呼び出しは，OS やハードウェアなどの特定のプラットフォーム，若しくはプログラミング言語などに用意されている関数や命令の集合である．このうちプラットフォームから提供される API は，プラットフォーム固有の機能が高度になってきているため，アプリケーションが細かい操作をしなくてもプラットフォーム固有の機能にアクセスしやすいように設計されている．このように，抽象度が高く高機能な API が提供されているプラットフォーム上で動作するアプリケーションを開発する際には，あらかじめ用意された API の中から必要なものを選択し，組み合わせて利用することが一般的である．このことから，アプリケーションにおいて利用される API の種類や並びと機能との間には関連性があると考えられる．

本研究では，API 呼び出し列を用いて，アプリケーション間の機能の差異を表示するツールを試作した．試作したツールは，2 つのアプリケーションからそれぞれ API 呼び出し列を抽出し，それらの差分と事前知識を比較することにより，機能の差異を特定する．そして，抽象度が高く高機能な API を提供しているプラットフォームとして Android を対象とし，作

成したツールを用いてケーススタディを行った．その結果，提案手法によりアプリケーションの機能の比較が実現できていることを確認した．

以降，2 節では，本研究の背景や関連研究について述べる．3 節では，提案する手法について述べる．4 節では，ツールの実装について説明し，5 節で適用実験について説明する．6 節では，本研究のまとめと今後の課題を述べる．

2　背景

API 呼び出しを利用したアプリケーション比較ツールを試作した背景として，アプリケーションの比較の必要性と，今回注目した API 呼び出し，API 呼び出しと機能の関係，Android について説明する．

2.1　ソフトウェアの比較

今日では大量のアプリケーションが入手可能である．例えば，パソコン向けには vector[8]，Android 端末向けには Android Market[2] などのアプリケーションダウンロードサイトがあり，そこには多くのアプリケーションが登録されている．また，インターネット上には，SourceForge.net[6] や Google Code[3] など，オープンソースソフトウェアの開発者向けにコードホスティング機能を提供しているサービスが多くある．開発者はそこから大量のソースコードを容易に入手することが可能である．

これらのたくさんのアプリケーションの中には，同じ機能を持つアプリケーションが複数あるため，利用者・開発者はその違いを理解したうえで自分が利用するアプリケーションを選択することになる．違いを理解するためには，アプリケーション同士を効率的に比較することが重要である．

利用者は，複数のソフトウェアを比較して，その中に自分が利用したい機能が存在するかを確認する必要がある．また，開発者も既存のアプリケーションを利用することがある．ソースコード中の関数やクラスなどのソフトウェア構成要素はソフトウェア部品と呼ばれ [4]，ソフトウェア開発の生産性や品質を高めるために既存のソフトウェア部品を再利用することが行われている．再利用のためには，ソフトウェア同士を比較して機能の違いを知り，必要な部品を選択する必要がある．

ソフトウェアの比較方法としては，実際に利用する，ドキュメントを読んで比較する，またソースコード同士を比較するなど様々な方法が考えられる．しかし，実際に利用することは確認漏れが生じる可能性があり，時間面でのコストも大きい．また，ドキュメント間の比較も，形式が統一されていなかったり，ドキュメントが存在しない場合もあるため，確実な方法ではない．ソースコードを比較する方法のうち単純なものとしては，Unix diff が挙げられる．しかし，Unix diff のような単純な比較では，使用言語と実現している機能が同じソフトウェア同士でも，設計が違えばほぼ全体が差分として抽出されてしまう恐れがある．また，機能とは関係のない差分も多く出力されてしまうことになり，理解が難しくなり現実的ではない．

このような問題があるため，ソフトウェアの比較や分類，部品の評価を効率的に行うことが重要になっている．

大量のソースコードから必要なものを選択するために，ソフトウェアを自動的に分類する研究 [5][12] や，ソフトウェアやソフトウェア部品を定量的に評価する研究 [13] が行われている．

2.2 プラットフォーム依存の開発と API

現在，OS やハードウェアなどのプラットフォームには様々な種類がある．各プラットフォームは固有のデバイスや UI などの特徴的な機能を持つ．例えば，Microsoft Windows 上で動作するソフトウェアは，多くが共通の GUI コンポーネントを利用している．携帯端末であれば，カメラや通話などの機能が搭載されている．このようなプラットフォームを対象に開発を行う場合は，特徴的な機能を利用するための API が提供されている．

API は，Application Programming Interface の略語であり，あるプラットフォーム向けにアプリケーションを開発する際に利用できる関数や命令の集合である．ウィンドウ表示など多くのソフトウェアで共通して呼び出される機能は，あらかじめ OS などのプラットフォームから API が提供されており，開発者は必要な API を呼び出すことで機能を実現できる．また，ファイル入出力などを API 経由で実現させることで，システムの保護にも役立っている．

2.3 API 呼び出しと機能・比較

アプリケーションは複数の機能を組み合わせることで実装される．高度な API が多く提供されている環境では，アプリケーションの実装は API 呼び出しの組み合わせにより表現されている場合がある．つまり，API 呼び出しはアプリケーションが実現している機能を表現しているといえる．

API の呼び出しは，1 回の呼び出しが 1 つの機能と対応していることもあるが，多くの場合複数の呼び出しの組み合わせで 1 つの機能を実装している．本稿では，複数の API の呼び出しの並びを API 呼び出し列と呼ぶ．

API 呼び出しの解析は，ソフトウェア解析の分野でも活用されている．Xie ら [9] は，オープンソースのリポジトリから API の利用方法を検索するためのフレームワークを作成した．岡本ら [10] は，ソフトウェア個々の特徴（ソフトウェアバースマーク）として API 呼び出しの呼び出し順序，呼び出し頻度を利用することを提案した．牛窓ら [12] は，Java アプリケーションの API 呼び出しから，ソフトウェアを機能ごとに自動分類する手法を提案した．

2.4 Android と Android API

Android[1] は，米 Google 社を中心に組織された Open Handset Alliance によって開発されている，オペレーティングシステムやミドルウェア，いくつかのアプリケーションを含む携帯端末向けのプラットフォームである．Android を搭載する端末の多くは，携帯電話として電話やデータ通信が可能であり，またタッチパネルや加速度センサー，カメラなどのデバイスを搭載している．

Android は世界中で普及しており，米 Google 社が提供している Android アプリケーション配布場所である Android Market[2] では，2010 年 10 月に利用可能なアプリケーションの数が 10 万を超えた [7]．Android Market を経由しないアプリケーションの配布も認められているため，ユーザは数多くのアプリケーションを利用可能である．

また，Android アプリケーションのソースコードも多く公開されている．例えば，Google Code のプロジェクトホスティングサービスには 2011 年 2 月 16 日現在，“Android” のラベルがついたプロジェクトが 5547 件登録されている．これらの中には Android Market でのダウンロード数が 25 万件を超えるようなアプリケーションも存在している．

Android 向けのアプリケーションは Java で記述される．使用可能なライブラリは，Java のオープンソースな実装である Apache Harmony から GUI ツールキットの Swing などを取り除き，カメラ，GPS，タッチスクリーンなどのデバイスを利用するための独自の API などを付加したものとなっている．このため，Android のソフトウェア開発キット（Android SDK）には Java 由来のライブラリ，Apache HttpComponents プロジェクトのライブラリ，Android オリジナルの API などが混在している．また，Google Maps の機能を Android アプリケーション上で利用するための API も，標準で提供されている．

3　提案手法

この節では，今回提案する API 呼び出し列の差分を用いたアプリケーションの比較方法について説明する．アプリケーションのソースコード中の API 呼び出し列は，そのアプリケーションの機能を表していると考え，API 呼び出し列を比較することで 2 つのアプリケーションに共通する機能，片方のアプリケーションにしか出現しない機能を特定する．

3.1　概要

提案手法の全体の流れを図 1 に示す．比較は以下の手順で行う．

ステップ 1 比較したい 2 つのアプリケーションのソースコードを解析し，メソッド呼び出しの中から API 呼び出し列を抽出する．

ステップ 2 ステップ 1 で抽出した API 呼び出し列を比較し，共通して出現する API 呼び出し列の集合（以下，共通部分）と，これに当てはまらなかった API 呼び出し列の集合（以下，差分）を得る．

ステップ 3 ステップ 2 で得た共通部分，差分それぞれを事前知識（後述）と照合し，2 つのアプリケーションに共通する機能，片方のアプリケーションにしか出現しない機能を特定する．

また，単に API 呼び出し列を比較するだけでは，結果が API の並びとなりまだ理解しにくい．そこで，共通部分，差分がどのような機能を表しているのかを理解しやすくするために，あらかじめ機能のある API 列に名前を付けておく．本稿ではこれを事前知識と呼ぶ．事前知識は，API 呼び出し列と機能名の組である．事前知識との照合を行うことで，共通部分，差分にどのような機能を持つ API 列が含まれているかを容易に知ることができる．

API 呼び出しの判定基準と事前知識は，比較を行う前に対象とするプラットフォームごとに作成しておく必要がある．

以下，各ステップごとに詳細な手順を説明する．

3.2　ステップ 1: ソースコードからの API 呼び出し列の抽出

ソースコードからの API 呼び出し列抽出の概要を図 2 に示す．

このステップで抽出される API 呼び出し列は，メソッドから抽出した興味のある API 呼び出しの並びである．興味のある API 呼び出しは，解析対象のプラットフォームに特有の API 呼び出しであると考えられるものである．１つのメソッドから抽出した API 呼び出しの並びは，出現した行の順と一致するものとする．同一行内に複数の API 呼び出しが出現

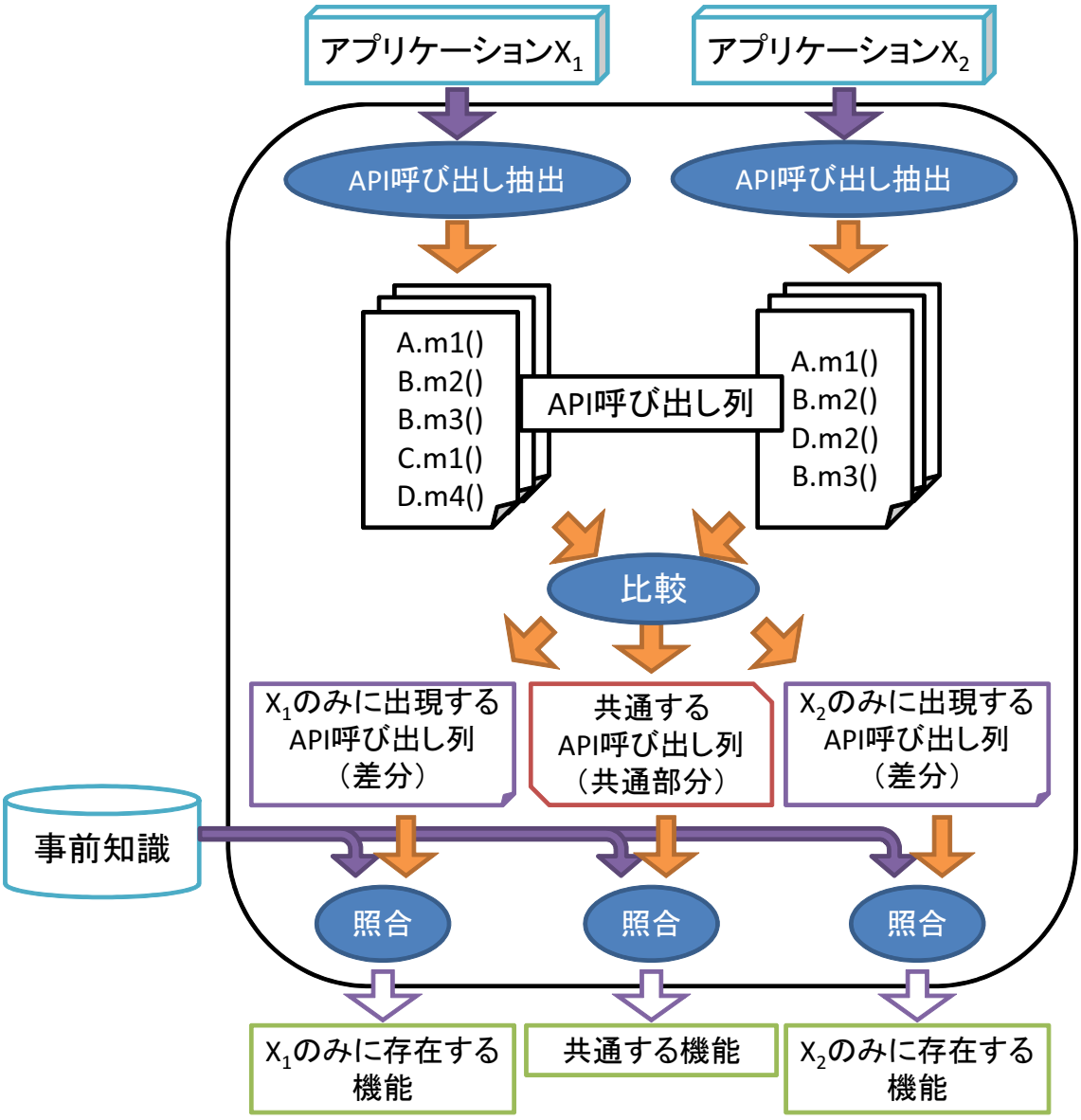
アプリケーションX1
アプリケーションX2
API呼び出し抽出
API呼び出し抽出
A.m1()
B.m2()
B.m3()
C.m1()
D.m4()
API呼び出し列
A.m1()
B.m2()
D.m2()
B.m3()
比較
X1のみに出現する
API呼び出し列
（差分）
共通する
API呼び出し列
（共通部分）
X2のみに出現する
API呼び出し列
（差分）
事前知識
照合
照合
照合
X1のみに存在する
機能
共通する機能
X2のみに存在する
機能

図 1: 提案手法の概要

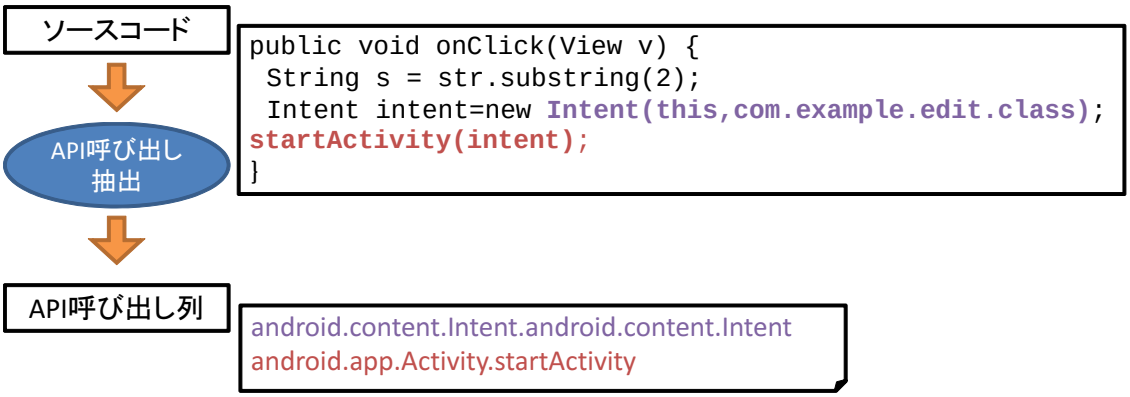

図 2: API 呼び出し列の抽出

した場合は，その中でアルファベット順に並べる．if 文などの制御構造は無視する．クラス内でのメソッドの出現順，またプロジェクト内でのクラスの順番については，メソッドごとに別の API 呼び出し列を構築するので考慮していない．

Android アプリケーションを開発する際に利用できるソフトウェア開発キット（Android SDK）には，Java 由来のライブラリ，Apache HttpComponents プロジェクトのライブラリ，Android オリジナルの API などが混在している．また，Google MAPs の機能を Android アプリケーション上で提供するための API も標準で提供されている．今回は，メソッドの完全限定名が “android.”，“com.google.android” で始まるものが Android アプリケーションに特有の API 呼び出しであると判定した．

API 呼び出し列の抽出は以下の手順で行う．

まず，アプリケーションのソースコードを解析する．これには MASU[11] を用いた．解析結果から，メソッド呼び出しのうち，Android 特有の API 呼び出しであるものを抽出する．抽出した結果は API 呼び出しとソースコード中に出現した行番号の組である．これを API 呼び出し情報と呼ぶ．API 呼び出し情報を，出現した行番号の順に並べる．同じ行に複数の API 呼び出しが現れた場合は，API 呼び出しの完全限定名のアルファベット順に並べる．1 つのメソッドから抽出した API 呼び出し情報を並べたものを，1 つの API 呼び出し列とする．

メソッドから抽出した API 呼び出し列は，そのメソッドが出現するファイル，クラス，メソッドの情報とともにメソッド情報としてアプリケーションごとに保存する．メソッド情報の集合がアプリケーション情報である．

API 呼び出し情報やメソッド情報，アプリケーション情報は，API 呼び出しからソースコード中の出現位置をたどるために利用する．

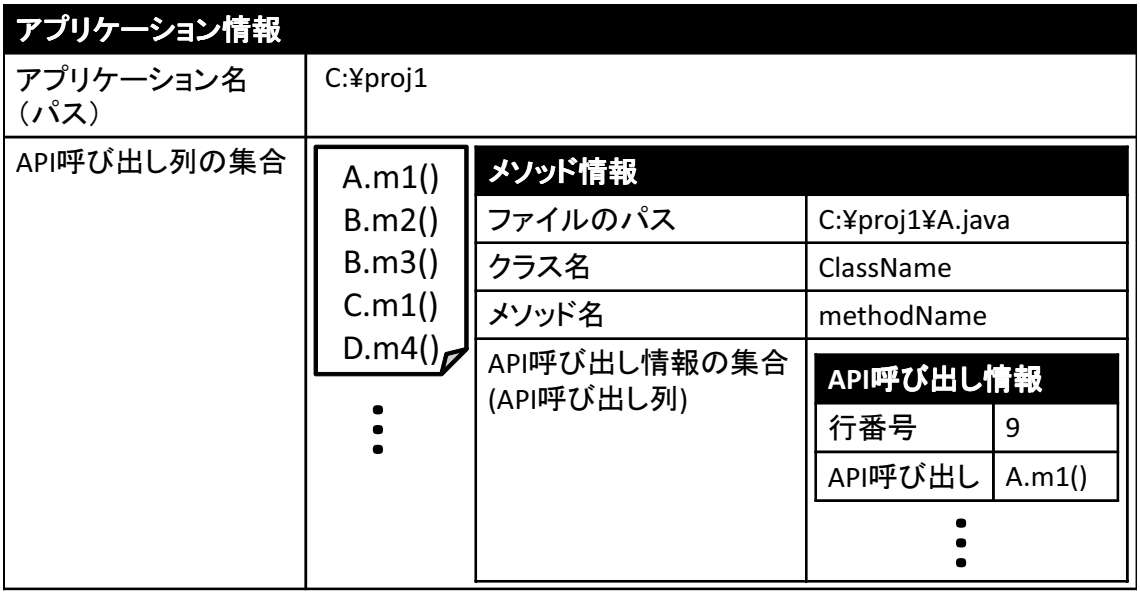

図 3: アプリケーション情報，メソッド情報と API 呼び出し情報の例

3.3 ステップ 2: API 呼び出し列の比較

次に，API 呼び出し列の比較を行い，共通部分と差分の API 呼び出し列を得る．API 呼び出し列の比較の概要を図 4 に示す.

API 呼び出し列の比較方法を具体的に説明する．まず，図 5 のような表を作成し，共通して出現する API 呼び出しの並びを探す．この際，長さが 2 以上の共通する連続部分に絞って共通部分を探すことにより，API 呼び出しの種類が一致するだけでなく使用方法が一致しているかを確認する．共通部分として以下の情報が抽出される．

- API 呼び出し列
- 出現位置情報の集合
  - API 呼び出し列開始位置の API 呼び出し情報

共通部分として抽出した API 呼び出しには，図 6 のように印をつけておく．この印をつけた API 呼び出しを除いたものを差分とする．共通部分を取り除く際，もともと連続していなかった API 呼び出し同士が隣接することがないようにする．具体的には，共通部分を列から取り除く際に，その部分で別々の API 呼び出し列に分割する．差分は，分割してできた API 呼び出し列の集合となる．

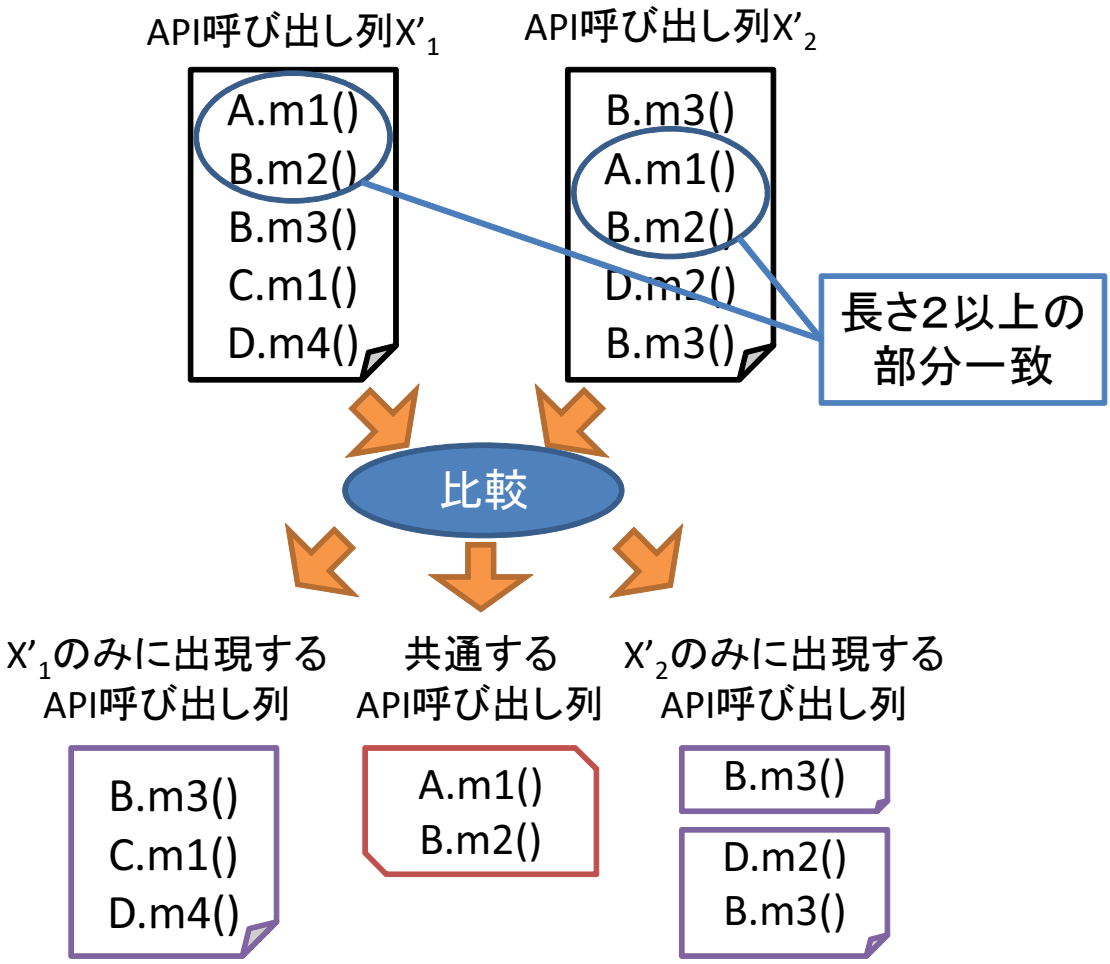

図 4: API 呼び出し列の比較

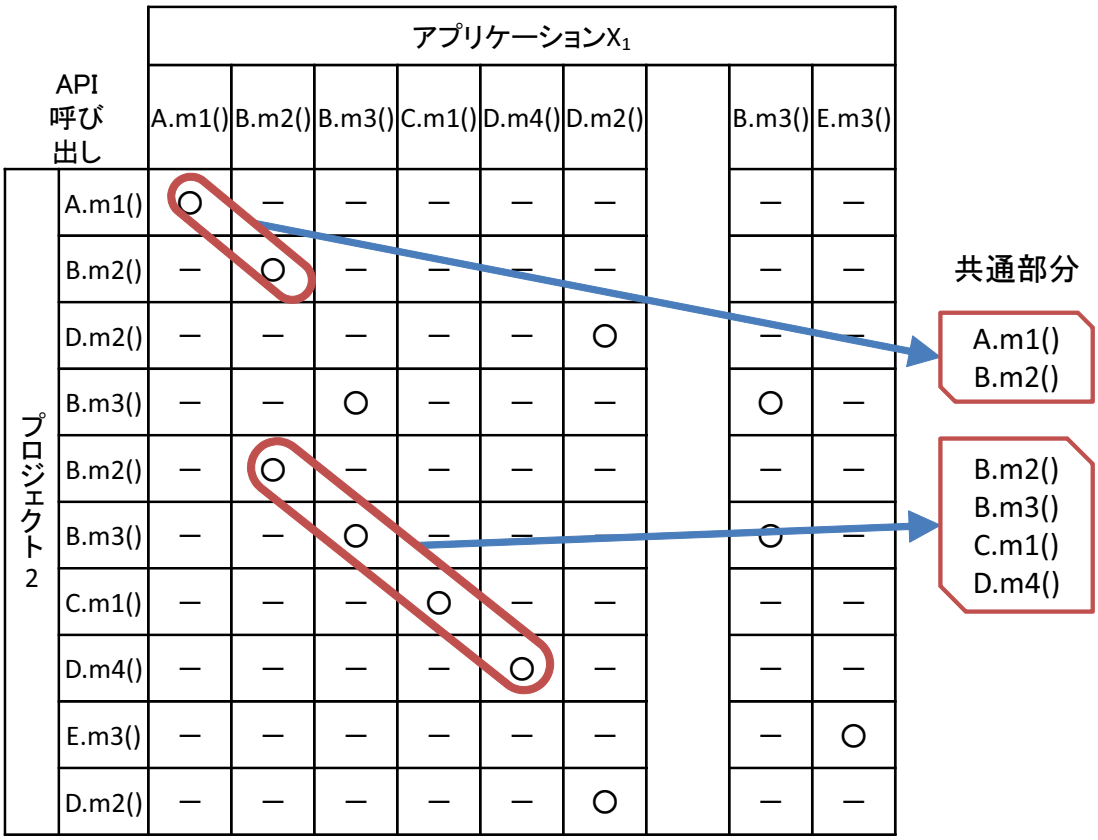
アプリケーションX1
API
呼び
出し
A.m1()
B.m2()
B.m3()
C.m1()
D.m4()
D.m2()
B.m3()
E.m3()
プロジェクト2
A.m1()
B.m2()
D.m2()
B.m3()
B.m2()
B.m3()
C.m1()
D.m4()
E.m3()
D.m2()
共通部分
A.m1()
B.m2()
B.m2()
B.m3()
C.m1()
D.m4()

図 5: 共通部分の抽出

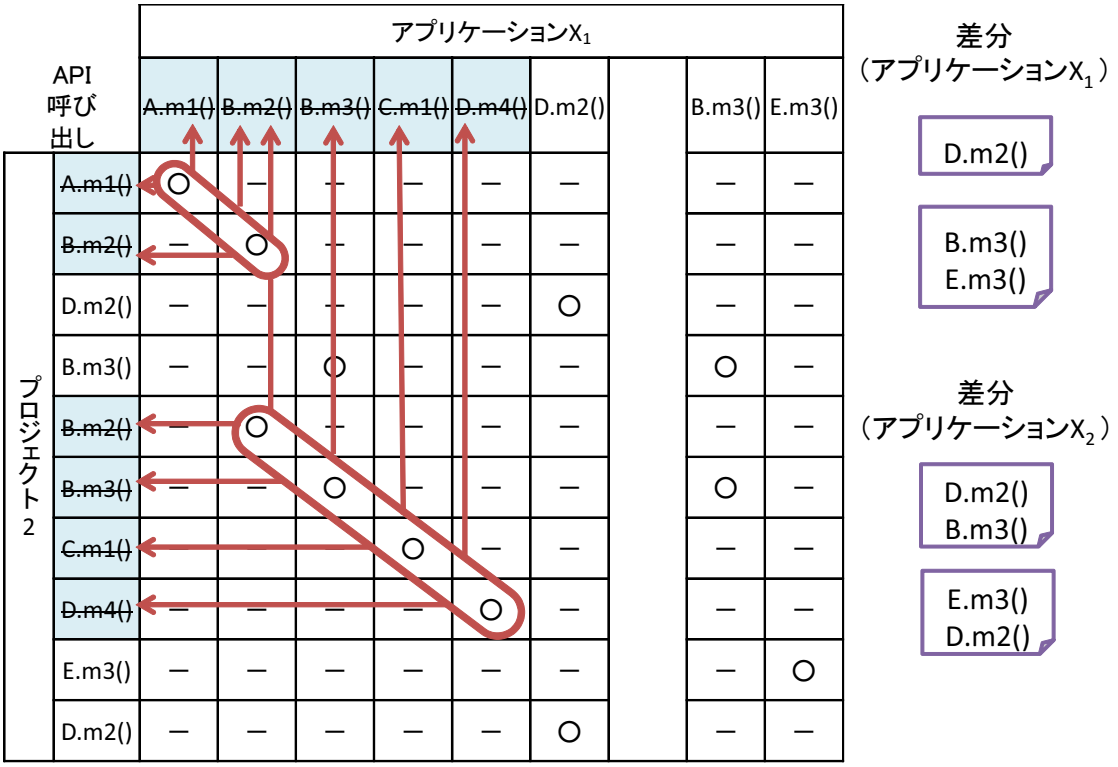
アプリケーションX1
API
呼び
出し
A.m1()
B.m2()
B.m3()
C.m1()
D.m4()
D.m2()
B.m3()
E.m3()
プロジェクト
2
差分
（アプリケーションX1）
D.m2()
B.m3()
E.m3()
差分
（アプリケーションX2）
D.m2()
B.m3()
E.m3()
D.m2()

図 6: 差分の抽出

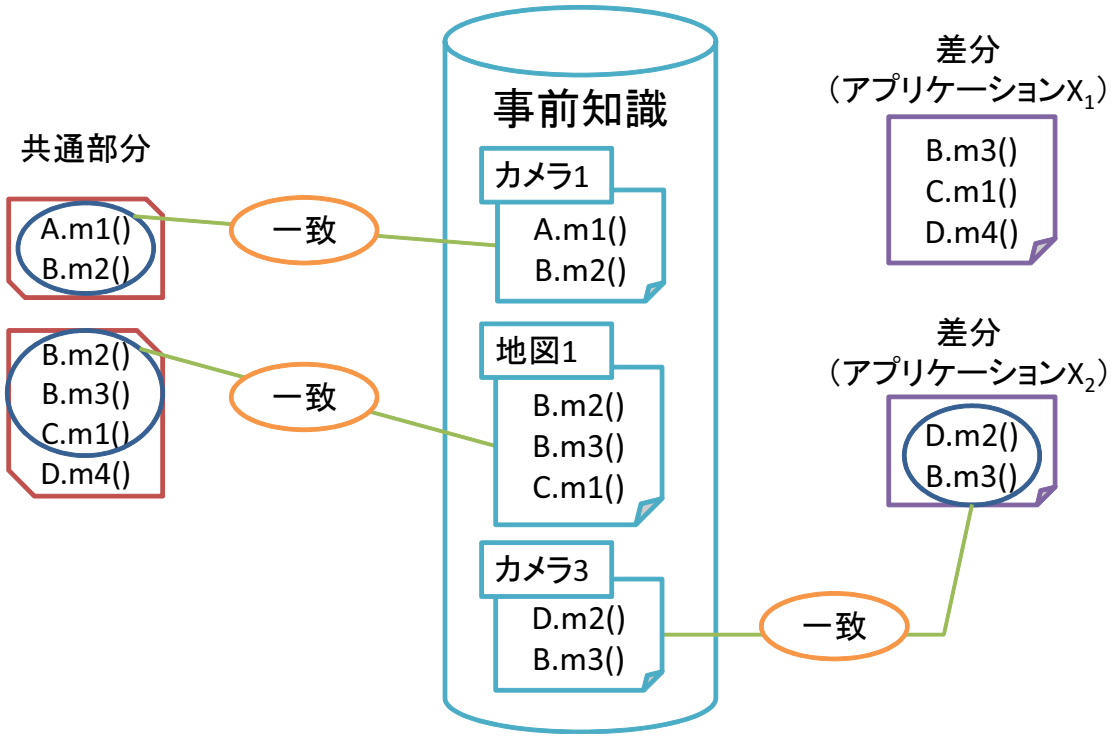

図 7: 事前知識との照合

3.4 ステップ 3: 事前知識との照合

API 呼び出し列の比較により得られた共通部分及び差分と，作成した事前知識とを照合し，得られる機能名と出現位置の対応情報を提示する．

事前知識の API 呼び出し列と同じ API 呼び出し列が，共通部分及び差分の API 呼び出し列のどこかに含まれているかを検索する (図 7)．一致した部分があれば，その位置情報を記録していく．

事前知識との照合が終わると，機能名と出現位置の対応情報 (図 8) が得られる．この情報が，比較結果として提示される．

3.5 事前知識の作成

事前知識作成の概要を図 9 に示す．今回，事前知識は以下の手順で作成した．

ステップ a ある共通した機能を持つと考えられるアプリケーションのソースコードを集め，ソースコードを解析し，API 呼び出し列を抽出する．比較におけるステップ 1 と同様である．

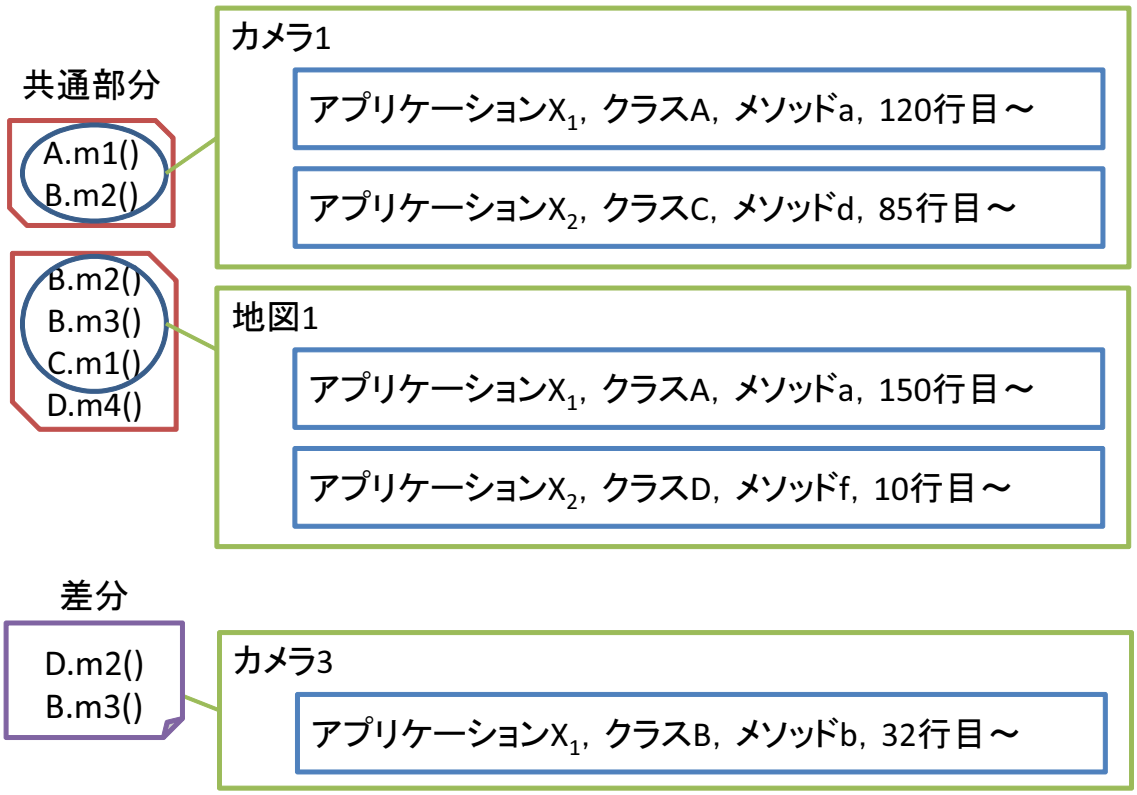

図 8: 照合した後のデータ

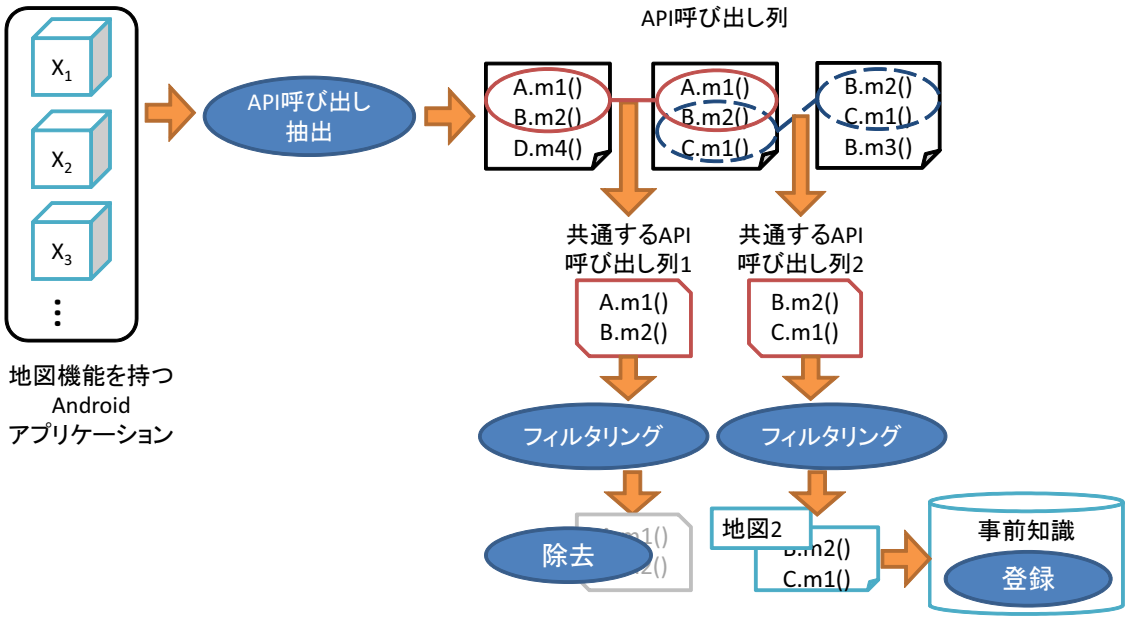

図 9: 事前知識の作成

ステップ b ステップ a で抽出した API 呼び出し列を比較し，共通部分を得る．比較における ステップ 2 と同じ方法を，すべてのアプリケーションの組で試すことにより実現する．

ステップ c ステップ b で得た共通部分をフィルタリングし，残ったものに名前を付けて事前知識として登録する．

以下で手順の詳細を示す．

まず，ある共通した機能を持つと考えられるアプリケーションを集める．解析と比較のステップは 3.2 節，3.3 節で述べた手順と同一であるためここでは省略する．

事前知識の作成には，共通部分を利用する．共通の機能を持つアプリケーションから抽出された API 呼び出し列の共通部分は，その機能を実現する部分であると考えられるからである．そのため，共通部分に名前付けをしていくことで，API 呼び出し列とそれが実現する機能についての事前知識が作成できる．共通部分は，API 呼び出し列を比較したときの長さ 2 以上の部分一致と定義しているため，事前知識中の API 呼び出し列も長さ 2 以上の API 呼び出し列となる．

次に，フィルタリングを行う．フィルタリングの目的は，具体的な機能を表していない API 呼び出し列を事前知識から取り除くことである．今回は，著者自身が以下の基準に従い，API 呼び出し列が機能を表しているかを調べた．

- カメラなど端末のデバイスを利用し，操作や情報の取得を行うか

- ダイアログや通知領域などを利用したユーザへの情報の提示を行うか
- インテント（ほかのアプリケーションの呼び出し）を利用するか

フィルタリングを通過したものには，機能名を付けて事前知識として収録する．

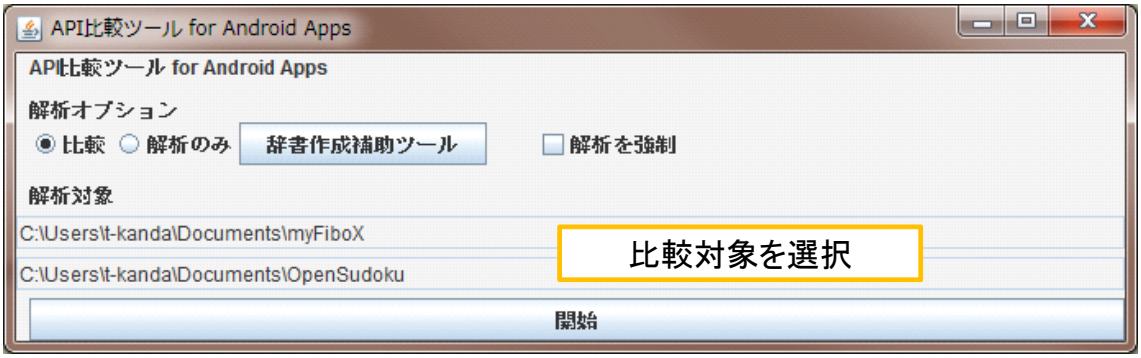

図 10: ツール起動時

4 ツールの実装

本研究で試作したツールの実装を説明する．

利用者は，比較前に事前知識を準備しておく．比較したいアプリケーションを 2 つ選択すると，提案手法に基づいて解析を行い，結果を提示する．

4.1 対象の選択

ツールを起動すると，初期画面 (図 10) が表示される．

左側のラジオボタンで，比較を行うか解析のみを行うかを選択する．

プログラムは前回の解析結果を優先的に利用するので，解析をやり直したい場合は「解析を強制」チェックボックスにチェックを入れる．

比較対象のプロジェクトが表示されている部分をクリックすると，ダイアログが現れ，プロジェクトを選択することができる．

「開始」ボタンを押すと，解析と比較を開始する．

4.2 比較結果の表示

図 11 は比較を行った結果を表示する画面である．

上部は 3 つに分かれており，中央に共通部分を，左右に比較元それぞれにしか出現しないものを表示している．検出された機能はボタンで表示されている． ボタンのうち一つを選んでクリックすると，下部に情報が表示される．表示される情報は以下のとおりである．

- 機能名
- 対応する API 列

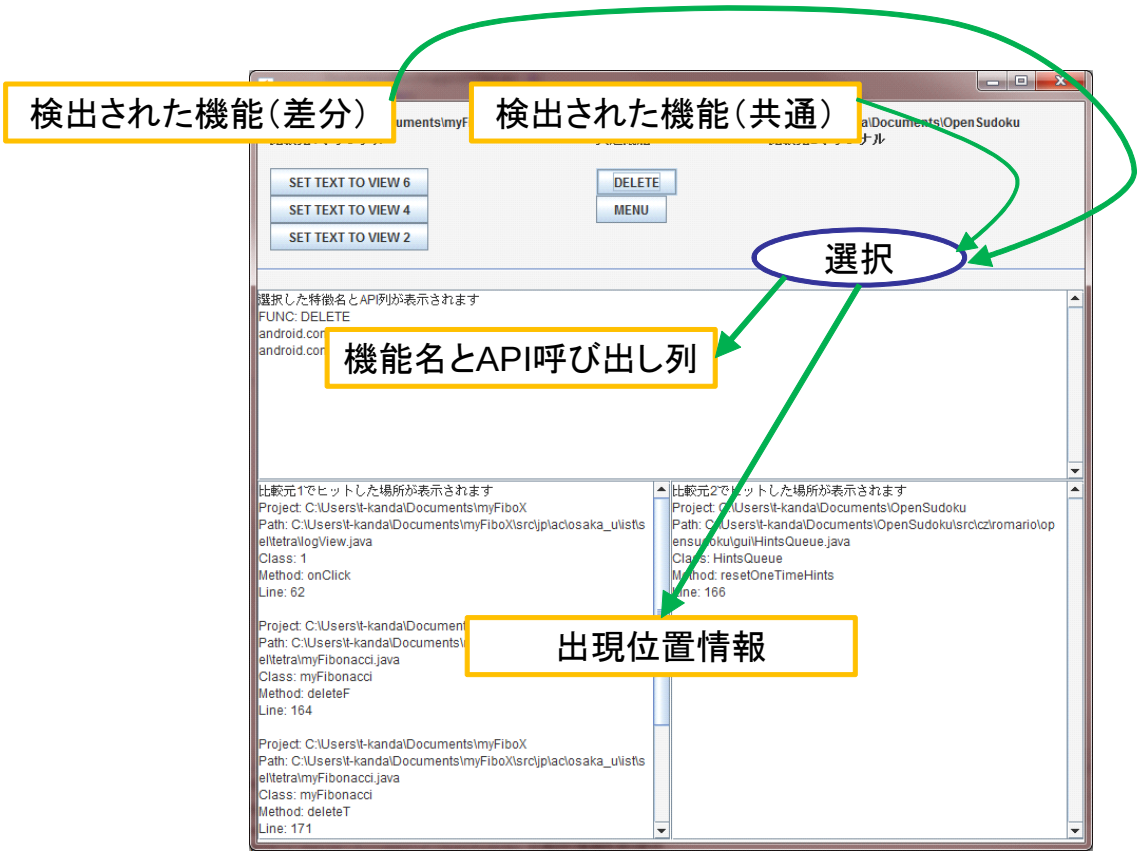

図 11: 比較結果表示画面

- 対応する API 列が出現した位置情報の並び
  - プロジェクトのパス
  - ファイルのパス
  - クラス名
  - メソッド名
  - 行番号

5　ケーススタディ

作成した事前知識が比較対象のアプリケーションに含まれていることと，試作したツールにより，Android アプリケーション同士の比較が可能であることを確認するため，地図機能を持つアプリケーションを対象としてケーススタディを行った．

5.1　実験対象

地図機能を持つアプリケーションとして，Google Code に登録されているプロジェクトのうち “Map” のラベルがついたプロジェクトから 11 個のアプリケーションを取得した．このうち 6 個を事前知識作成に利用し，残りの 5 個を比較対象とした．事前知識作成に利用したアプリケーション名とソースコードの行数 (LOC) を表 1 に，比較に利用したアプリケーション名とソースコードの行数 (LOC) を表 2 に示す．事前知識作成では，6 個のアプリケーションから 156 個の API 呼び出し列を抽出し，フィルタリングにより 23 個に機能名をつけ事前知識として収録した．

5.2　手順

ケーススタディの手順を示す．

ステップ 1 試作したツールを用いて比較対象の 5 つのアプリケーションからそれぞれ API 呼び出し列を抽出し，抽出した API 呼び出し列と事前知識を照合する．比較対象の各アプリケーションについて，API 呼び出し列と事前知識との対応関係が得られる．

ステップ 2 対応関係から，5 つのアプリケーションが区別できていることを確認する

表 1: 事前知識作成に利用したアプリケーションの一覧

アプリケーション名	LOC	抽出 API 数
OpenGPSTracker	8122	1099
mapsforge	37326	1407
OSMandroid	3150	175
TripComputer	14487	825
shareyourdrive	2761	346
savage-router	1041	66
合計	66797	3918

ステップ 3 試作したツールを用いて 2 つのアプリケーションを比較し，機能の差分が提示されることを確認

5.3 結果

各アプリケーションと事前知識を照合し，機能を表しているかどうかを判定した結果を表 3 にまとめる．表 3 から，事前知識のうち 8 割近くが比較対象のアプリケーションに表れていた．いくつのアプリケーションに含まれているかは機能によりばらつきが見られた．

次に，5 つのアプリケーションが持つ機能の違いの例を表 4 に，例に挙げた機能の API 呼び出し列を表 5 に示す．

各アプリケーションに共通する機能や，一部のアプリケーションにしか検出されなかった機能があることがわかる．例えば，緯度経度を取得する機能は，5 つのアプリケーションに共通である．また，地図機能以外の面でも，警告ダイアログはすべてのアプリケーションで利用されることや，サブメニューやトースト（ウィンドウの前面にポップアップされるメッセージ）を利用した UI が提供されていることなどを表から読み取ることができる．いくつかの機能でアプリケーション間の違いを確認できていることから，API 呼び出し列がアプリケーションそれぞれの機能を表現しているといえる．

最後に，試作したツールを用いてアプリケーション cycroid とアプリケーション maps-minus を比較した結果を図 12 に示す．アプリケーション 2 つを比較し，共通する機能とどちらか片方にしか出現しない機能が区分されて出力されている．

5.4 考察

事前知識作成において抽出した API 呼び出し列のうち，実際に機能を表していると判断されたものは少なかった．機能を表していないと判断した API 呼び出し列の多くは，アプ

表 2: 比較に利用したアプリケーションの一覧

アプリケーション名	LOC	抽出 API 数
MapDroid	6387	1160
cycroid	1278	761
yozi	5348	159
maps-minus	1785	218
BigPlanetTw	4139	432
合計	18937	2730

表 3: アプリケーションと事前知識を照合した結果

	機能を含むアプリケーション数						
	0	1	2	3	4	5	合計
機能を表しているものの数	5	8	3	0	6	1	23

表 4: 5 つのアプリケーションが持つ機能の違いの例

機能名	警告ダイアログ表示	緯度経度取得	トースト表示	緯度経度設定	サブメニュー表示
MapDroid	○	○	○	○	—
cycroid	○	—	○	—	—
maps-minus	○	○	—	—	—
yozi	○	○	○	—	—
BigPlanetTw	○	○	○	○	○

表 5: 機能の API 呼び出し列の例

機能名	API 呼び出し列
トースト表示 (画面にホップアップする通知)	android.widget.Toast.makeText android.widget.Toast.show
緯度経度設定	android.location.Location.setLatitude android.location.Location.setLongitude
警告ダイアログ表示	android.app.AlertDialog.Builder.android.app.AlertDialog.Builder android.app.AlertDialog.Builder.setTitle
緯度経度取得	android.location.Location.getLatitude android.location.Location.getLongitude
サブメニュー表示	android.view.Menu.addSubMenu android.view.SubMenu.setIcon

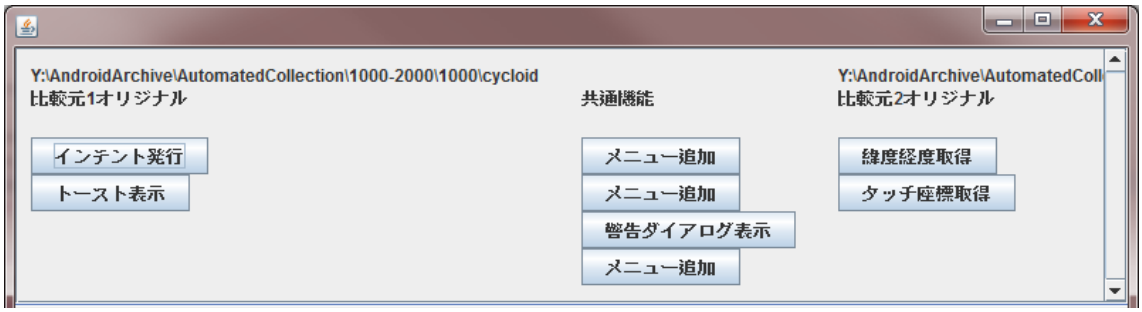

図 12: cycroid と maps-minus の比較

リケーション内部でデータを参照するようなものであった．これらをフィルタリングした後の事前知識では，8 割近くの API 呼び出し列が比較対象のアプリケーションに表れていた．このことから，あるアプリケーションにおいて特定の機能を実現している API 呼び出し列は他のアプリケーションでも出現することや，フィルタリングを適切に行うことで，比較に有用な事前知識が得られることがいえる．実際に機能を表している API 呼び出しが少なかった原因として，今回の手法では API 呼び出しの順序や回数が違うものは別の API 呼び出し列として扱っていることが考えられる．例えば，地図アプリケーションに出現する位置情報の取得機能は，緯度・経度の 2 つの値を取得する 2 種類の API 呼び出しを利用するため，並び順を考慮しても 2 つのパターンしかない．これに対して，地図アプリケーションに限定せず Android アプリケーションで広く使われている API は，出現回数がアプリケーションによって異なり，今回の手法では同じ API 呼び出しでも連続する回数ごとに別の事前知識として収録されてしまう．Android アプリケーションで広く使われている API であっても出現回数などに違いがあるものが多いため，他のアプリケーションと長さまで一致するパターンが少なくなったと考えられる．同じ API 呼び出しの繰り返しは回数にかかわらず 1 つの事前知識として収録することや，API 呼び出しの順番違いも 1 つの事前知識としてまとめるなどの対応が必要である．

アプリケーションごとの比較においては，地図機能以外にも様々な機能が共通していることが確認された．表 4 に示した例でも，いくつかの機能でアプリケーション間の違いを確認できていることから，API 呼び出し列を用いてアプリケーションに存在する機能を特定することが可能であるといえる．試作したツールを用いた実験でも，機能の違いが明確に表れており，アプリケーションの比較に成功したと言える．

今回の手法で検出された機能は，アプリケーションが実際に持っている機能であると考えてよいが，検出されなかった機能は存在しないとは限らない．事前知識に登録されていない API 呼び出し列は検出されないため，事前知識の精度によっては存在するはずの機能が検出

されないからである．先に述べたように，事前知識の精度を上げる工夫をおこなうことで，このような問題は減らすことができると考える．

また，検出された機能は，ドメインごとに固有の機能と，ドメインによらない機能が混在している．今回は地図機能を持つアプリケーションのみから事前知識を作成したが，事前知識を様々なドメインから取得することで，ドメインごとに固有の機能とドメインによらない機能に分類することができると考えている．

6　まとめ

本研究では，アプリケーションのソースコード中の API 呼び出し列を利用し，アプリケーションの比較を行うツールを試作した．試作したツールは，2 つのアプリケーションの API 呼び出し列を比較して共通部分，差分を抽出し，事前知識と照合することで 2 つのアプリケーションに共通する機能，片方のアプリケーションにしか出現しない機能を特定する．

そして，Android 用の地図機能を持つアプリケーションを対象に事前知識の作成とアプリケーションの比較を実際に行い，比較が可能であることを確認した．

今後の課題としては，ツールの精度向上のための事前知識作成方法の見直しが挙げられる．本研究で提案した手法では，API 呼び出し列が完全に一致した部分のみを共通部分として抽出しているが，実際には同じ API 呼び出しが並んでいても順序が異なって利用されている可能性がある．この場合，同じ機能を表している部分でもツールは差分として抽出してしまう．このような差異を吸収して提示できれば，比較結果がより正確なものになる．また，機能を表していると判断できない API 呼び出し列も事前知識に多く含まれたため，事前知識作成の際にこれらをフィルタリングする仕組みが必要である．

また，機能の分類を自動化して，出力を理解しやすいものにすることも重要な課題である．機能名の代わりに，現在どのドメインから作った事前知識かをもとに通し番号を付けているが，アプリケーションのドメインによらない機能と，ドメインごとに固有の機能を自動的に分類して表示できるような仕組みがあれば，ツールの出力がより理解しやすいものになると期待できる．

謝辞

本研究において，常に適切な御指導及び御助言を賜りました大阪大学大学院情報科学研究科コンピュータサイエンス専攻 井上 克郎 教授に心より深く感謝いたします.

本研究において，随時適切な御指導及び御助言を賜りました大阪大学大学院情報科学研究科コンピュータサイエンス専攻 松下 誠 准教授に深く感謝いたします.

本研究において，逐次適切な御指導及び御助言を頂きました大阪大学大学院情報科学研究科コンピュータサイエンス専攻 石尾 隆 助教に深く感謝いたします.

本研究において，適時適切な御指導及び御助言を頂きました東洋大学総合情報学部 早瀬 康裕 助教に深く感謝いたします.

本研究において, 数多くの御指導及び御助言を頂きました大阪大学大学院情報科学研究科コンピュータサイエンス専攻 真鍋 雄貴 氏に深く感謝いたします.

最後に，その他様々な御指導，御助言等を頂いた大阪大学大学院情報科学研究科コンピュータサイエンス専攻井上研究室の皆様に深く感謝いたします.

参考文献

[1] Android. http://www.android.com/.

[2] Android Market. http://market.android.com/.

[3] Google Code. http://code.google.com/.

[4] M.L. Griss. Software reuse architecture, process, and organization for business success. In *Proceedings of the Eighth Israeli Conference on Computer Systems and Software Engineering*, pp. 86 –89, Los Alamitos, CA, USA, 1997.

[5] Shinji Kawaguchi, Pankaj K. Garg, Makoto Matsushita, and Katsuro Inoue. MUDABlue: An automatic categorization system for open source repositories. In *In Proceedings of the 11th Asia-Pacific Software Engineering Conference*, pp. 184–193, Busan, Koreja, 2004.

[6] SourceForge.net. http://sourceforge.net/.

[7] Twitter . "One hundred thousand apps in Android Market.". http://twitter.com/androiddev/status/28701488389 :2011 年 2 月 10 日閲覧.

[8] Vector ソフトライブラリ & PC ショップ - 国内最大級の フリーソフトダウンロードサイト. http://www.vector.co.jp/.

[9] Tao Xie and Jian Pei. MAPO: Mining API usages from open source repositories. In *In Proceedings of the 2006 international workshop on Mining software repositories*, pp. 54–57, Shanghai,China, 2006.

[10] 岡本圭司, 玉田春昭, 中村匡秀, 門田暁人, 松本健一. API 呼出しを用いた動的バースマーク (ソフトウェア基礎, プログラム理論). 電子情報通信学会論文誌. D, 情報・システム, Vol. 89, No. 8, pp. 1751–1763, 2006-08-01.

[11] 三宅達也, 肥後芳樹, 楠本真二, 井上克郎. 多言語対応メトリックス計測プラグイン開発基盤 MASU の開発 (ソフトウェア工学). 電子情報通信学会論文誌. D, 情報・システム, Vol. 92, No. 9, pp. 1518–1531, 2009-09-01.

[12] 牛窓朋義, 門田暁人, 玉田春昭, 松本健一. 使用クラスに基づくソフトウェアの機能面からの分類. 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. 109, No. 170, pp. 31–36, 2009-07-30.

[13] 横森励士, 藤原晃, 山本哲男, 松下誠, 楠本真二, 井上克郎. 利用実績に基づくソフトウェア部品重要度評価システム (ソフトウェア工学). 電子情報通信学会論文誌. D-I, 情報・システム, I-情報処理, Vol. 86, No. 9, pp. 671–681, 2003-09-01.