Vol.2013-SE-179 No.3
2013/3/11

APIの使用状況に基づくイディオムの抽出とそれを用いたソフトウェア自動分類

丹羽 一平[1]　大久保 弘崇[2]　粕谷 英人[2]　山本 晋一郎[2]

概要：本稿ではソフトウェア内で慣用的に用いられる API の使用パターン，即ちイディオムをソフトウェア群から抽出する手法を提案する．ソフトウェア内での API の使われ方には特徴があるという仮説に基づき，ソフトウェア群に対する API の使用状況を特徴ベクトルとし，その類似度に基づいて API のグループを作成することでイディオムを抽出する．抽出したイディオムに対して，それを利用している実例であるソフトウェアを対応付けることにより，開発者がイディオムとその実例の両方を参照できるカタログを作成する．

1. はじめに

1.1 背景

ソフトウェアはプラットフォームが提供する機能を API を通じて利用する．開発者向けのドキュメントでは一般的に一つ一つの API の利用方法や動作を説明したリファレンスは提供されるが，言葉で簡潔に説明されていても，特殊な場合への対応などで実際のプログラム記述は複雑になる場合もある．また，それらをどのように組み合わせて利用するべきかといった相互関係や連携にまで言及されることは少ない．そのため，ソフトウェア開発者はこれらの知識の獲得を，ドキュメントの断片的情報からの試行錯誤で発見したり，他の開発者が発見した知識をコードリーディングを行い調査するなど，個々にコストをかけている．

開発者がある効果を必要とし，それを実現するイディオムが知りたいときに参照できる資料として逆引き辞典がある．しかし逆引き辞典は人手で編集されるため内容が網羅的でなく，作成コストがかかるためプラットフォームの更新に追従できない問題，また開発者が利用するプラットフォームに関するものがそもそも存在しないことも多い．

プラットフォームの多様化と高機能化により，開発者にとってイディオムに基づいたソフトウェアカタログは需要が高まっており，何らかの方法でそれを提供することが望まれる．それが自動的に生成できるならば，逆引き辞典の編集が人手によることに起因する問題を解決できる．

1.2 目的

本研究の目的は，イディオムに基づいたソフトウェアカタログを作成することである．そのためにソフトウェア群からイディオムを自動的に抽出する手法を提案する．本研究ではソフトウェアの振舞いを実装するために使われている API のグループがイディオムであると考える．また，ソフトウェア内での API の使用状況には特徴があり，イディオムを抽出するための指標にできる．そこで，API の使用状況を表す特徴ベクトルを抽出し，特徴ベクトルの類似度が一定値以上の API をグルーピングすることでイディオムを抽出する．ここで，逆引き辞典ならば項目であるイディオムに名前が付けられるため，開発者は必要とするイディオムを名前を頼りに発見する．一方，上述の手法で自動抽出されたイディオムは提供する効果を表す名前のようなものはなく，開発者が必要なイディオムを発見する手がかりがない．そこで，抽出したイディオムに基づいてそれを利用している実例であるソフトウェアを分類し提示する．開発者はここで提示された具体的なソフトウェアの振舞いを通して，自分が必要とするイディオムを発見する手がかりとする．こうして構成される自動抽出されたイディオムとその実例であるソフトウェアのリストは，開発者が参照できるイディオムのカタログとなる．

1.3 本論文の構成

本論文の構成を述べる．2 節ではイディオムに基づいたソフトウェアカタログの作成について述べ，本研究では API の使用状況に基づいてイディオムを抽出することを示

[1] 愛知県立大学大学院 情報科学研究科
Graduate School of Information Science and Technology, Aichi Prefectural University
[2] 愛知県立大学 情報科学部
School. of Information Science and Technology, Aichi Prefectural University

す．3 節で，本研究で提案する手法を用いた評価実験を行う．4 節では評価実験で作成されたソフトウェアカタログについて評価し，本手法の有用性について述べる．5 節で関連研究について述べ，6 節で，まとめと今後の課題について示す．

2. イディオムに基づいたソフトウェアカタログの作成

利用しているイディオムをソフトウェア群から抽出する方法を述べる．また，イディオムに基づいたカタログの作成方法とその結果を述べる．

2.1 イディオム

言語学における「イディオム (Idiom)」とは「語彙」を意味しその単語組合せで特定の意味を持つ慣用表現である．ソフトウェア開発におけるイディオムとはソースコード中の複数箇所に類似したコード片となって出現する特定の振舞いを実装するためのパターンである．つまり慣用的に使われる使用目的が明確な API の使用パターンをイディオムと呼ぶ．また，古くから多くのソフトウェア開発者により積み重ねられてきたものであり，多くの人が多くの場面で使い再利用することにメリットがある．これらのイディオムは入出力に関するものやセキュリティに関するもの，インタフェースに関するものなど多岐に分類できる．本研究ではイディオムを，振舞いを実装するために使われている API のグループとする．

2.2 API の使用状況

イディオムをなす API のグループを発見する問題を考える．ソフトウェアは多数の API を利用する．その中にはイディオムをなすグループが存在する．そこで，多数のソフトウェアにおける個々の API 使用状況を分析することで，イディオムを発見する．プラットフォームが提供する API の数を n とする．API 使用状況を調査するソフトウェア，すなわちサンプルの数を m とおく．このソフトウェア群を対象に，各々が利用している API を抽出し $n \times m$ の表を作成する．表の項目には，API がソフトウェアで使われている場合 1，そうでない場合 0 を行に記す．したがって各行は 1 と 0 で構成される APIi に対する m 次元の特徴ベクトル $v_i\,(1 \leq i \leq n)$ とみなすことができる．また，各列はソフトウェア i に対する n 次元の特徴ベクトルとみなすことができる．この表が API の使用状況を表している．API 使用状況の仮想例を表 1 に示す．例えば $API1$ に注目するとソフトウェア 1, 3, 5 で使われているため 1，ソフトウェア 2, 4 では使われていないため 0 が行に記されている．

この表の各行は特徴ベクトル v_i となっている．例えば $API1$ の行を参照し特徴ベクトル $v_1 = (1, 0, 1, 0, 1)$ を作成する．これらの特徴ベクトルが各 API 使用状況の特徴を表し，この特徴に基づいてイディオムを抽出する．

表 1 API の使用状況

	ソフトウェア 1	ソフトウェア 2	ソフトウェア 3	ソフトウェア 4	ソフトウェア 5
$API1$	1	0	1	0	1
$API2$	1	1	1	1	0
$API3$	1	1	0	1	0
$API4$	1	1	0	1	0
$API5$	0	1	0	0	1

2.3 イディオムの抽出

本研究では複数ソフトウェアにおける API の使用状況が類似する API の組合せがイディオムであると考える．したがって，特徴ベクトル間の類似度に着目し API をグルーピングした集合がイディオムとなる．API 使用状況の特徴ベクトルから，イディオム候補を抽出する手順を述べる．

(1) 対象 API の制限

特徴ベクトル全体の集合 V を作成する．まず，ソフトウェアサンプル数 m に対して出現頻度が低すぎる API を除外する．ここでは出現回数の下限を 2 とした．次に，逆に出現頻度が高すぎる，普遍的に使用される API もまた除外する．この上限はサンプル数 m に対する係数として指定する．

$V = \{v_i \mid 2 \leq number(v_i) \leq T_{util} \times m\}\ (1 \leq i \leq n)$
$number()$：特徴ベクトル中の 1 の数を返す
汎用度閾値 T_{util}：値 1 の占める割合

(2) 初期グループの作成

特徴ベクトルの類似度に基づいて，API のグループ G を次のように作成する．

$G = \{g_i\}$
$g_i = \{j \mid v_j \in V, sim(v_i, v_j) \geq T_{sim}\}$
類似度閾値 T_{sim}

(3) グループの統合

以下の操作を G が変化しなくなるまで繰返す．

$\forall g_i, g_j \in G$ について
$g_j \subseteq g_i$ のとき $G \leftarrow G \backslash g_j$

G は抽出した全てのイディオムの集合であり，$g \in G$ は抽出したイディオムを構成する API の集合である．

2.4 ソフトウェアカタログの作成

抽出したイディオムに基づいてソフトウェアカタログを作成する．2.2 節で作成した表の各列を参照し各ソフト

表 2　ソフトウェアカタログの例

No.	使用 API	実例ソフトウェア
1	API1, API2	ソフトウェア 1, ソフトウェア 3
2	API2, API3, API4	ソフトウェア 1, ソフトウェア 2, ソフトウェア 4

ウェアの特徴ベクトル $s_l\,(1 \leq \ell \leq m)$ を作成する．イディオム g_k の実例であるソフトウェアの集合 S_k をソフトウェアの特徴ベクトルとの比較により求める．

$S_k = \{\ell \,|\, \forall i \in g_k,\, s_\ell(i) = 1\,\}$
$v(r)$：ベクトル v の r 番目の要素

これにより (g_k, S_k) はイディオムとイディオムを構成する全ての API を使っているソフトウェアの集合の組となる．

表 1 に対して以上の手法を適用すると，表 2 に示すソフトウェアカタログが得られる．g_i が持つ全ての API を使っているソフトウェアをグルーピングする．イディオム g_i と対応するソフトウェアグループ S_i の関係を以下に示す．全てのイディオム G についてソフトウェアをグルーピングしたものをソフトウェアカタログとして提示する．ただし，$T_{util} = 1$，$sim(X,Y) = \dfrac{|X \cap Y|}{|X \cup Y|}$，$T_{sim} = 0.4$ とした．$sim(X,Y)$ については 3.5 で述べる．例での類似度は API3 と 4 の間で最大値 1 をとり，1 と 3，1 と 4，2 と 5 の間で最小値 0.2 をとる．

3. 評価実験

提案手法により抽出されるイディオムの有用性を調べるため評価実験を行った．収集した Firefox のアドオンに対し本手法を適用し，使われているイディオムを抽出する．次にそのイディオムに対応するソフトウェアをグルーピングしソフトウェアカタログを構成する．得られたソフトウェアカタログに対し，イディオムを構成する API の実例ソフトウェア内での使用状況を調査し，意味のある API の組合せが提示できているかを検証する．

3.1 対象ソフトウェア

Firefox のアドオンを対象のソフトウェア群として，Firefox がアドオン向けに提供している API に関するカタログの作成を行い提案手法を評価する．Firefox はオープンソース・クロスプラットフォームのウェブブラウザであり，機能を拡張できるウェブブラウザとして有名なソフトウェアである．Firefox では拡張機能をアドオンと呼び，ユーザは様々なアドオンを Firefox に追加することにより機能を追加することが可能である．アドオンはオープンソースである Firefox の拡張機能であり，JavaScript で記述される．公式サイトからソースを入手可能であり，今日までに 7000 個以上のアドオンが公開されている．アドオンは，動作を記述する JavaScript とユーザインタフェースを記述する XUL から構成されている．

アドオン開発の簡単なチュートリアルを載せている Web ページは存在する．しかし振舞いの実装に必要なイディオムや拡張ポイントの一覧など拡張機能開発に必要な情報を詳しく載せている Web ページは我々が調査した限り発見できなかった．例えそのような Web ページが存在したとしても拡張機能開発に必要な全ての情報を載せると Web ページの調査に時間がかかる．そのため開発者はイディオムや拡張ポイントを探すためににソースプログラムを調査することを余儀なくされ，多くの時間的コストが必要となる．したがって，本手法を用いて作成したイディオムに基づいたカタログを提示することによりこの時間的コストを削減することが可能である．

また，Firefox は 6 週間ごとにアップデートされており，それに伴う API 変更が頻繁なため手動で編集される逆引き辞典などの編集が追従できないなどの問題がある．したがって，アップデートごとに自動的にイディオムのカタログを提示することでその問題を解決することができる．

表 3　実験データ

対象アドオン数	330
対象 JavaScript API 数	820
類似度指標	ジャッカード係数
類似度閾値	0.5
汎用度閾値	0.1

3.2 実験環境

表 3 の環境下で評価実験を行った．Mozilla のアドオン紹介サイトから 330 個のアドオンを収集した．また，3 つのサイトから JavaScript の API を 820 個収集しリストにして用意した．本実験では特徴ベクトルの大きさに影響を受けにくい類似度指標であるジャッカード係数を利用する．選択理由については 3.5 節にて議論する．設定閾値の組み合わせとその結果を複数検証し，適切な設定値を主観的に判断した．閾値の決定方法については 3.6 節にて議論する．

3.3 イディオムのグルーピング

2.3 節で提案した手順に基づいてイディオムの抽出を行った．この結果 18 個のイディオムが抽出された．グループの一覧を表 4 に示す．

3.4 ソフトウェアカタログの作成

2.4 節の手法に基づいてソフトウェアカタログを作成した．作成結果の抜粋を表 5 に示す．ソフトウェアカタログ全体の評価は 4 節で行う．

表 4 抽出イディオム一覧

nsICookie, nsICookieManager
nsIGlobalHistory, nsIBrowserHistory
getUTCDate, getUTCFullYear, getUTCHours, getUTCMinutes, getUTCMonth, getUTCSeconds
setHours, setDate, setMilliseconds, setMinutes, setSeconds
innerHeight, innerWidth, outerHeight, outerWidth
nsIFactory, nsIComponentRegistrar
nsINavHistoryService, nsIFaviconService, nsINavBookmarksService
nsILoginInfo, nsILoginManager
nsIStandardURL, nsIServerSocket, nsIBinaryOutputStream, nsIProtocolHandler
nsIProtocolProxyService, nsISocketTransportService, nsIHttpChannelInternal
onBlur, onFocus
onDragOver, onDragStart, onDrop
onMouseOut, onMouseOver
acos, atan, tan
asin, sin
scrollByPages, onSelectionChange, nsIDocShell
onScroll, nsIDOMElement
nsISimpleEnumerator, default

表 5 ソフトウェアカタログ抜粋

No.	使用 API	実例ソフトウェア
1	nsICookieManager, nsICookie	AddonFox, BetterPrivacy, Biblion.ru Toolbar, Evernote Web Clipper, Facebook Toolbar, feedly, Flash Video Downloader Youtube Downloader, FoxLingo, FoxTrick, Ghostery, MyAthens Toolbar, Splashtop Connect, Torrent Finder Toolbar, Web Developer, WiseStamp, WOT, X-notifier
2	nsIBrowserHistory, nsIGlobalHistory	PDF Download, Web Developer
3	nsILoginManager, nsILoginInfo	CoolPrevies, Clearly, FireFTP, FVD Speed Dial with Full Online Sync, Gmail Manager, Password Exporter, Pocket, ReminderFox, Shopple eGift Rewards Bar, Xmarks, Yoono, Evernote Web Clipper

3.5 ベクトル間類似度

2 つのデータが似ている度合いをあらわす指標であり，類似度の高さや距離の近さといった数値を表す．よく使われるベクトル間類似度の関数は以下のものがある．

- ユークリッド距離
  $sim(X,Y) = sqrt((X-Y) \times t(X-Y))$
  ($t()$：転置関数))
  ベクトル間の幾何学距離の指標
- ピアソンの相関係数
  $$sim(X,Y) = \frac{(X-\bar{X}) \cdot (Y-\bar{Y})}{|X-\bar{X}| \times |Y-\bar{Y}|}$$
  2 つの確率変数の相関関数
- ジャッカード係数
  $$sim(X,Y) = \frac{|X \cap Y|}{|X \cup Y|}$$
  2 つの集合の共起頻度の指標
- コサイン類似度
  $$sim(X,Y) = \frac{X \cdot Y}{|X| \times |Y|}$$
  ベクトル間のなす角度の指標

ユークリッド距離とピアソンの相関係数は対象データの要素数の影響を受けるためロバストではない．本手法で用いる類似度指標は対象データの要素数が対象ソフトウェア数に依存するため，要素数による影響を受ける指標を用いることはできない．したがってこれらの指標は本手法に適切ではない．一方ジャッカード係数とコサイン類似度は対象ベクトルの大きさの影響を受けにくい類似度指標である．本手法においてジャッカード係数を用いた場合とコサイン類似度を用いた場合の抽出イディオム数を確認する．図 1, 2 はそれぞれジャッカード係数とコサイン類似度を用いた場合の抽出イディオム数の関係を確認したグラフである．それぞれの類似度閾値，汎用度閾値の抽出イディオム数の関係を表している．グラフより類似度閾値が 0.15 以上の範囲で両者の類似度が同じ傾向を示していることが分かる．また，本手法で用いる特徴ベクトルは 0,1 で構成されるものであり，共起頻度に関係する類似度指標を用いるのが妥当である．したがって本手法ではジャッカード係数を用いる．

3.6 設定閾値の考察

類似度閾値，汎用度閾値と抽出イディオム数の関係を確認する．図 1, 2 から汎用度閾値を大きくすると抽出イディオム数は減り，類似度閾値を大きくすると抽出イディオム

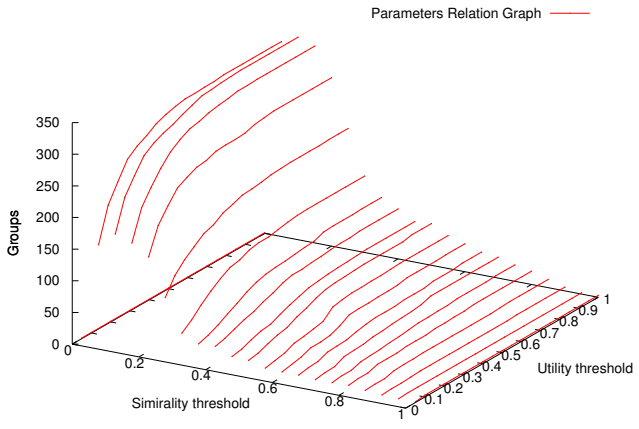

図 1 ジャッカード係数を用いた場合

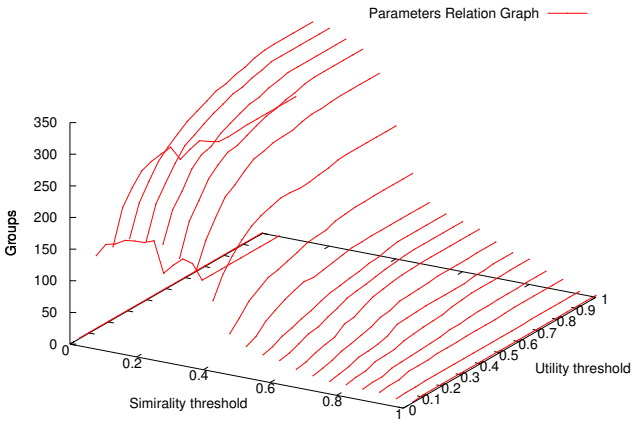

図 2 コサイン類似度を用いた場合

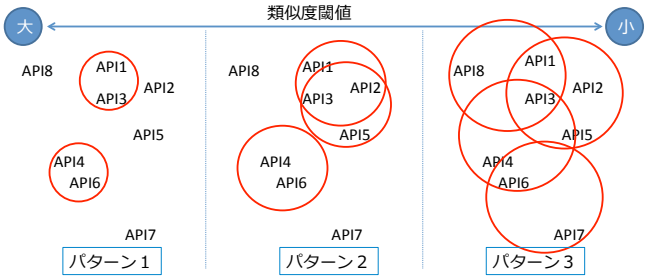

図 3 類似度閾値と出力グループの関係イメージ

表 6 出力グループ

パターン 1	パターン 2	パターン 3
{1, 3}	{1, 2, 3}	{1, 2, 3, 5}
{4, 6}	{2, 3, 5}	{1, 3, 8}
	{4, 6}	{3, 4, 6}
		{6, 7}

数は増えることが分かる．

3.6.1 類似度閾値

類似度閾値と抽出イディオムの関係イメージを図 3 に示す．各 API は類似度が高いものを近くに配置している．また，類似度閾値を小さくすると類似度の低い API を同じグループに結びつけることになるので円で表した 1 つのグループが大きくなる．グループの円にちょうど入る API の集合をイディオムとする．表 6 に各パターンの抽出イディオムを示す．パターン 3 ではパターン 2 の {1, 2, 3} と {2, 3, 5} が統合され {1, 2, 3, 5} となりこの部分ではグループ数が減る．しかし新たに {1, 3, 8} と {6, 7} が抽出されるためグループ数が増える．また，{4, 6} のグループが {3, 4, 6} となり要素が増える．次に，パターン 1 ではパターン 2 の {2, 3, 5} が消滅しグループ数が減る．また，{1, 2, 3} の要素が減り {1, 3} となる．

このように類似度閾値を変更することにより出力結果に以下の影響を与える

- 抽出イディオム数が変化
  新規抽出，消滅，統合
- 抽出イディオム要素が変化

3.6.2 汎用度閾値

汎用度閾値による制限により汎用的な API を排除できる．汎用的な API は以下の特徴をも．つ

- 多くのソフトウェアから使われている
- 実装フェーズで調査する必要性が低い

グルーピングの際に汎用的な API を除外しないことは，結果に以下の影響を与える．

- 汎用的な API で構成される要素数の多いグループが抽出される
- 要素の似たグループが数多く抽出される

これらはカタログの見通しを損ねるため，提案手法には汎用的な API を除外する操作を加えた．汎用度閾値を大きく設定することで汎用的な API を減らすことができ，生成カタログの見通しを良くする．しかし，汎用度閾値を大きく設定しすぎることで汎用的な API 以外の API も排除してしまう可能性が高くなる．

3.6.3 2 つの閾値の決定

類似度閾値の調整によりグループの新規抽出，消滅，統合により抽出イディオム数が変化する．また各グループの要素が変化するが，これらの変化を予想できない．汎用度閾値の調整により汎用的な API を排除できるが，汎用的な API を定量的に定義できない．本評価実験ではそれぞれの組合せを変化させて実験を繰返すことにより対象ソフトウェアのカタログ生成に適していると考えられる値を導き出した．したがって，一般的な対象における効果的な結果が得られる 2 つの閾値の一意的な決定は困難であり，これらは利用者が調整するのが妥当だと考える．

表 9　誤検出のイディオム

nsIDocShell, scrollByPages, onSelectionChange
nsIDOMElement, onScroll
default, nsISimpleEnumerator
nsIFactory, nsIComponentRegister
nsINavHistoryService, nsINavBookmarksService, nsIFaviconService
nsIServerSocket, nsIBinaryOutputStream, nsIProtocolHandler, nsIStandardURL
nsIProtocolProxyService, nsISocketTransportService, nsIHttpChannelInternal

4. 評価

3.4 節で作成したソフトウェアカタログが有用であるか判断する．抽出されたイディオムの API がソフトウェア内でそれぞれどのように使われているかを調査した．それにより抽出された各イディオムを有用なイディオム，有用だが自明なイディオム，誤検出のイディオムに分類し考察を行う．

4.1 有用なイディオム

表 7 に示す 3 グループを有用なイディオムと判断する．

{nsICookie, nsICookieManager} の使用箇所をソースコードから抜粋し図 4 に示す．9 行目で `nsICookieManager` を使いクッキーのリストを取得し，13 行目で `nsICookie` クラスのインスタンスであるかチェックをしている．このようにこのイディオムはブラウザに保存されているクッキーに対してある特定の処理を行う．

4.2 有用だが自明なイディオム

表 8 に示す 8 グループを有用だが自明なイディオムであると判断した．

有用だが自明なイディオム {getUTCMinutes, getUTCMonth, getUTCSeconds, getUTCDate, getUTCHours, getUTCFullYear} のグループ要素 API の使用箇所をソースコードから抜粋しリスト 5 に示す．これらの API は定義されている関数の引数から Date オブジェクトの情報を取得するために使われており，日付や時間などを扱うための実装方法だといえる．しかしこれらは一緒に使うことが自明であると判断する．

4.3 誤検出のイディオム

表 9 に示す 7 グループを誤検出のイディオムと判断した．これらの API はソースコード上で関係性を持たず使われている．これらの API はソースコード上の離れた場所で使われていることが確認できているため，それを考慮して排除することができる可能性がある．

4.4 評価のまとめ

有用なイディオムを 3 グループ，有用だが自明なイディオムを 8 グループ，誤検出のイディオムを 7 グループ抽出した．このうち有用なイディオム，有用だが自明なイディオムは構成する API がソフトウェアの機能を実装するために使われていることを確認した．したがってこれらのイディオムに基づいたカタログは開発者がイディオムを調べる際の実例として有用なものである．また，誤検出のイディオムを構成する API はソースコード上の離れた場所で使われているので，API 抽出時に利用場所の情報も抽出し利用することにより排除できる可能性がある．

本手法では汎用度閾値により汎用的な API を対象 API から排除し，類似度閾値により API をグルーピングしイディオムを抽出する．これらの閾値は抽出イディオム数，構成する API，グループの大きさに影響を与えるがそれらの予測は困難である．したがって一般的な対象における効果的な結果が得られる 2 つの閾値の一意的な決定は困難であり，これらは利用者が調整するのが妥当だと考える．

5. 関連研究

5.1 Software Similarity and Classification

本研究と同様，ソフトウェアをある基準に従い分類する手法として Software Similarity and Classification[1] がある．この研究はマルウェア，ソフトウェア盗作，コードクローン，学生の課題盗用の発見を目的としている．これらの発見のための 2 ソフトウェア間の類似度計算方法が紹介されており，それを利用して対象ソフトウェアと類似するソフトウェアをデータベースから分類する．この研究ではソフトウェア作成者のプログラム変形や難読化といった偽造を見破る必要があるため，プログラム解析，バイナリの静的解析，動的解析，特徴抽出など厳密な類似度計算を必要とする．本研究は開発者の既存ソフトウェア実例の調査を支援するものであり，プログラムの変形や難読化を考慮する必要がない．また，1 つのソフトウェアと類似するソフトウェアを分類するのではなく，対象ソフトウェア群をイディオムに基づいて分類する点が異なる．

5.2 FCDG に基づいたコーディングパターン

本研究と同様，ソフトウェア開発者が調査する既存ソフトウェアの実例を抽出する研究として FCDG に基づいたコーディングパターン [2] がある．この研究はコーディングパターンの抽出を目的としており，頻出するライブラリ関数の組合せパターンをコーディングパターンとして提示する．また，条件分岐からなら制御構造を考慮した手法を提案しており，本研究よりも厳密な抽出を試みている．しかし関数呼び出し依存グラフの作成コストが高い．一方で本研究は API 使用状況の類似度を用いて大量のソフトウェアからイディオムを抽出する手法でありコストが低いが，

表 7 有用なイディオムに基づくソフトウェアカタログ

No.	使用 API	実例ソフトウェア
1	nsICookieManager, nsICookie	, AddonFox, BetterPrivacy, Biblion.ru Toolbar, Evernote Web Clipper, Facebook Toolbar, feedly, Flash Video Downloader Youtube Downloader, FoxLingo, FoxTrick, Ghostery, MyAthens Toolbar, Splashtop Connect, Torrent Finder Toolbar, Web Developer, WiseStamp, WOT, X-notifier
2	nsIBrowserHistory, nsIGlobalHistory	PDF Download, Web Developer
3	nsILoginManager, nsILoginInfo	CoolPreviews, Clearly, FireFTP, FVD Speed Dial with Full Online Sync, Gmail Manager, Password Exporter, Pocket, ReminderFox, Shopple eGift Rewards Bar, Xmarks, Yoono, Evernote Web Clipper

```
1 var getIsFromMozilla = function() {
2     var isFromMozilla = true;
3     var prefType = AddonFoxAddItInstaller.PREFS.getPrefType('isfrommozilla');
4     if (prefType != AddonFoxAddItInstaller.PREFS.PREF_INVALID)
5         isFromMozilla = AddonFoxAddItInstaller.PREFS.getBoolPref('isfrommozilla');
6     else {
7         //Are we coming from mozilla?
8         var cookieManager = Components.classes["@mozilla.org/cookiemanager;1"]
9             .getService(Components.interfaces.nsICookieManager);
10        var cookieIterator = cookieManager.enumerator;
11        while (cookieIterator.hasMoreElements()){
12            var cookie = cookieIterator.getNext();
13            if (cookie instanceof Components.interfaces.nsICookie){
14                if (cookie.host && cookie.host.indexOf("addonfox.com") >= 0 && cookie.name == "AddonFoxS")
15                    isFromMozilla = false;
16            }
17        }
18        AddonFoxAddItInstaller.PREFS.setBoolPref('isfrommozilla',isFromMozilla);
19    }
20    return isFromMozilla;
21 }
```

図 4 イディオム {nsICookie，nsICookieManager} の実例

近似的な抽出であり厳密な抽出ではない．したがって本手法で分類したソフトウェアに対し FCDG に基づいた抽出を行うことで相乗効果を得ることができる．

6. おわりに

6.1 まとめ

本研究ではソフトウェア内で慣用的に使われる使用目的が明確な API の使用パターン，即ちイディオムを抽出する．ソフトウェア内での API の使われ方には特徴があり，API の使用状況を表す特徴ベクトルを作成する．特徴ベクトルの類似度が一定値以上の API をグルーピングすることでイディオムを抽出する．また，抽出したイディオムに基づいてそれを利用している実例であるソフトウェアを分類しソフトウェアカタログとして提示する．

本手法で作成したソフトウェアカタログが有用であるか判断するために Firefox のアドオンを対象に実験を行った．330 のソフトウェア，820 の API を対象に本手法を適用した結果，3 つの有用なイディオム，8 つの有用だが自明なイディオム，7 つの誤検出のイディオムを抽出した．

6.2 今後の課題

以下に今後の課題を挙げる．

- 適切な閾値の設定
  本手法で設定する類似度閾値，汎用度閾値は出力イディオムの数，グループの大きさ，構成する API に対し影響を与える．本研究ではこれらの変化を予測することが困難であり，適切な値を一意的に決定することは困難であった．したがってこの 2 つの閾値を適切に設定する方法を確立する必要がある．
- 誤検出のイディオムの排除
  本手法で抽出した 18 イディオムのうち 7 つが誤検出のイディオムであった．これらのイディオムを構成する API はデータ依存関係や制御依存関係にないものであり，ソースコード上の離れた場所に出現する事が多いことが確認できた．したがって，API 抽出時にこれらの知識を利用し本手法を拡張することで，これらのイディオムを排除し精度を上げる必要がある．
- 他言語環境での実験
  本手法によるイディオムの抽出は，他のプログラミング言語環境でも可能であるかを検証する必要がある．多くの実験データを集計し，本手法に汎用性があるか

表 8 有用だが自明なイディオムに基づいたソフトウェアカタログ

No.	使用 API	実例ソフトウェア
1	getUTCMinutes, getUTCMonth, getUTCSeconds, getUTCDate, getUTCHours, getUTCFullYear	Screengrab, Splashtop Connect, Springpad Extension, Instrument Test, Tilt, WiseStamp, Xmarks, XO, Yoono
2	setDate, setHours, setMilliseconds, setMinutes, setSeconds	ChatZilla, feedly, mail.com MailCheck, Parlor, ReminderFox, Springpad Extension, WiseStamp
3	innerHight, innerWith, outerWidth, outerHeight	Clearly, Evernote Web Clipper, Parlor, Splashtop Connect, WiseStamp, XO, Yoono
4	onBlur, onFocus	ChatZilla, eBay Sidebar for Firefox, Greasemonkey, Tilt, Yoono
5	onDrop, onDragOver, onDragStar	Firebug, FoxClocks, ScrapBook
6	onMouseOut, onMouseOver	AutoPager, Facebook Toolbar, PDF Download, Quick Maps
7	acos, atan, tan	Collusion, Forecastfox, Pray Times!, Splashtop Connect
8	asin, sin	Collusion, Evernote Web Clipper, Integrated Gmail, Parlor, Pray Times!, Splashtop Connect, WiseStamp, Yoono

```
1 var formats = {
2         db: '%Y-%m-%d␣%H:%M:%S',
3         compact: '%Y%m%dT%H%M%S',
4         'short': '%d␣%b␣%H:%M',
5         'long': '%B␣%d,␣%Y␣%H:%M',
6         rfc822: function(date){
7                 return rfcDayAbbr[date.get('day')] + date.format(',␣%d␣') + rfcMonthAbbr[date.get('month')] + date.format(
                      '␣%Y␣%H:%M:%S␣%Z');
8         },
9         rfc2822: function(date){
10                return rfcDayAbbr[date.get('day')] + date.format(',␣%d␣') + rfcMonthAbbr[date.get('month')] + date.format(
                      '␣%Y␣%H:%M:%S␣%z');
11        },
12        iso8601: function(date){
13                return (
14                        date.getUTCFullYear() + '-' +
15                        pad(date.getUTCMonth() + 1, 2) + '-' +
16                        pad(date.getUTCDate(), 2) + 'T' +
17                        pad(date.getUTCHours(), 2) + ':' +
18                        pad(date.getUTCMinutes(), 2) + ':' +
19                        pad(date.getUTCSeconds(), 2) + '.' +
20                        pad(date.getUTCMilliseconds(), 3) + 'Z'
21                );
22        }
23 };
```

図 5 イディオム {getUTCMinutes, getUTCMonth, getUTCSeconds, getUTCDate, getUTCHours, getUTCFullYear} の実例

を確認する.

謝辞 本研究は科研費 22300011, 24300006 の助成を受けたものである.

参考文献

[1] Zhang, J., Chen, C., Xiang, Y., Zhou, W. and Xiang, Y.: Internet Traffic Classification by Aggregating Correlated Naive Bayes Predictions, *IEEE Transactions on Information Forensics and Security*, Vol. 8, No. 1, pp. 5–15 (2013).

[2] 渥美紀寿，山本晋一郎，結縁祥治，阿草清滋：FCDG に基づいたコーディングパターン，コンピュータソフトウェア， Vol. 21, No. 4, pp. 261–270（オンライン），入手先 ⟨http://ci.nii.ac.jp/naid/110003743171/⟩ (2004).