アプリケーション利用状況把握のための JavaScript フレームワーク層でのログ採取手法の検討

中里克久[†1]　　栗原英俊[†1]

概要：アプリケーション（以下，アプリ）の利用ログは，アプリ利用者の行動把握のために不可欠な情報である．従来型の Web アプリでは，主要な処理がサーバ側で行われるため，サーバ側でほとんどのログを採取することが可能であったが，HTML5 などのクライアント側の JavaScript 部分での処理比重が増した新しいアーキテクチャの Web アプリでは，クライアント側でもログを採取する必要が生じる．通常，クライアント側でのログ採取のためにはアプリにログ出力の命令を埋め込むことが必要となる．しかし，アプリがフレームワーク上に構築されている場合，アプリ固有部分を改造せず，フレームワーク層のみでログを採取できる可能性がある．本稿では，AngularJS を対象フレームワークとし，アプリの利用ログをフレームワーク層で効率的に採取するための手法を検討した．結果，フレームワークのソースコード分析が必要なログ採取コードの手動埋め込み手法に近い結果を，詳細なソースコード分析を行わずに自動的にログ採取コードを埋め込み，それによって得られたログデータの統計情報を基にして不要なログを絞り込む手法によっても得られることが確認できた．

キーワード：JavaScript，HTML5，フレームワーク，ログ，可視化

1. はじめに

HTML5 を活用した高機能な Web アプリや，Apache Cordova [1]を用いたスマートフォン/タブレット向けのハイブリッドアプリなど, JavaScript を開発言語としたアプリ開発は，従来型の Web アプリに留まらず，より多様な領域に広がってきている．

従来型の Web アプリでは，クライアント側の JavaScript で行う処理は簡単な画面修飾などに限定され，画面遷移や業務処理などの主要な処理はサーバ側で行われていた．一方，HTML5 などの JavaScript の処理比重を高めた新しいアーキテクチャのアプリでは，画面遷移や業務処理の一部もクライアント側の JavaScript で処理する．これにより，クライアントーサーバ間の通信待ちの契機が減り，利用者の操作に対するアプリのレスポンス向上などの効果が期待できる．本稿では，このようなアプリを JavaScript アプリと呼ぶこととする．

アプリの開発者や提供者にとって，アプリ利用者の行動を把握することは，アプリやサービスの改善検討のために重要である[2]. 従来型の Web アプリでは，主要な処理はすべてサーバ側で行われているため，サーバ側でアプリの利用ログを採取し，それを分析することによって利用者の行動把握に繋げることができた. 一方, JavaScript アプリでは，多くの処理がクライアント側で行われるため，サーバ側でのログ採取だけでは不十分であり，クライアント側の JavaScript 処理部でログを採取する必要が生じる．

サーバ側でログを採取する場合，処理の概要程度であれば Web サーバのアクセスログで把握することができる．この場合，サーバアプリの改造は不要である．一方，クライアント側のログ採取では，共通性が高くカスタマイズが容易な Web サーバに相当する実行基盤が存在しない. Web ブラウザが最もそれに近い存在で，ブラウザへのプラグインを用いてログ採取を行うアプローチも考えられるが，ブラウザはアプリ利用者の都合で変更可能であり，また，モバイル向けのブラウザではプラグインが使用できないなど，適用不可なケースが多く，すべての利用環境でログを採取しようとする場合，基本的にはアプリに直接ログ採取コードを埋め込む必要がある．

一方，JavaScript アプリでは，開発を容易にするためにフレームワークが利用されることがある．フレームワークを利用すると，アプリ内の主要な処理の流れがフレームワークによって制御され，アプリ開発者はその処理の流れを前提としてアプリ固有の処理部分の開発のみに専念できるようになる. JavaScript アプリ向けのフレームワークはオープンソース・ソフトウェア（以下，OSS）として多数開発されており，AngularJS [3]，Backbone.js [4]，React [5]などが代表的な JavaScript フレームワークとして知られている．このフレームワークは，クライアント側の実行環境である Web ブラウザや Web ブラウザ相当の機能を持つ WebView コンポーネントなどとアプリ固有処理部の中間で動作するので，フレームワーク層でログを採取することで，アプリ固有処理部を改造することなくアプリの利用状況を把握可能なログを採取できる可能性がある．

本稿の構成は以下の通りである．2 章でフレームワーク層でのログ採取の概念と実現方法について概要を示す．3 章では有力なフレームワークである AngularJS を対象にした場合のログ採取方法の詳細について説明する．4 章では AngularJS 利用アプリに 3 章の手法を適用してログを採取した結果を示し，それによる手法の評価を述べる．5 章では関連研究との比較を行い，6 章にて全体をまとめる．

†1 株式会社富士通研究所
Fujitsu Laboratories Ltd.

2. フレームワーク層でのログ採取方針

一般に，JavaScript アプリの設計は多様であり，どの箇所でログを採取すればアプリの利用状況把握のために効果的かもアプリ毎に異なる．しかし，フレームワーク上に構築された JavaScript アプリでは，アプリの構造がフレームワークによって規定されるため，アプリ毎の差異が縮小する．また，フレームワークとアプリ固有の処理部とは随時連携し，フレームワークがアプリを制御する形態となるため，フレームワークの内部の適切な部分でログ採取を行うことにより，アプリ固有部分を改造しなくてもアプリの主要な処理の流れを分析するために必要な情報が得られると考えられる．

フレームワーク部分でログを採取する場合，そのためのコードを手動で埋め込むアプローチと自動で埋め込むアプローチが考えられる．

2.1 ログ採取コードの手動埋め込み

フレームワーク特有のアプリ構造を念頭に，アプリの処理の流れを制御する箇所やアプリの状態変更を反映する箇所を選定してログ採取コードを埋め込むことにより，効果的なログ採取が可能になると考えられる．一方で，ログ採取コードの埋め込み箇所の選定のためにはフレームワークの内部構造を分析して各部の役割や重要度を判定する必要があり，作業コストが高い点が欠点である．

2.2 ログ採取コードの自動埋め込み

JavaScript はスクリプト言語であり，プログラムを比較的柔軟に改造することができる．このことから，JavaScript で記述されたフレームワークのごく限られた箇所を改造し，フレームワーク実行時にすべての処理部にログ出力のためのコードを動的に挿入することも可能であると考えられる．このような方式なら，フレームワークの内部構造を詳細に分析しなくてもログ採取コードの埋め込みが可能になる．

2.3 ログ採取コードの高度な自動埋め込み

ログ採取コードの単純な自動埋め込みでは，すべての処理部が同等に扱われ，利用者の行動把握に寄与しないノイズ的なログの割合が多くなってしまうと予想される．よって，何らかの方法でログ採取箇所を絞り込み，手動でのログ採取コード埋め込みに近い結果を自動的に得られるようにすることで，分析に有効な利用ログを低コストに採取することが可能になると考えられる．ログ採取箇所を絞り込む際，フレームワークの内部構造の人手での分析結果を利用したのでは手動埋め込みと同様の高い作業コストが必要となるため，内部構造分析結果以外の容易に入手可能な情報を基にして絞り込むことが必要となる．

3. AngularJS でのログ採取方法

OSS の JavaScript フレームワークは多数存在するが，AngularJS はその中でも人気の高いフレームワークとして知られている[6]．フレームワーク上でのログ採取という考え方は対象フレームワークを限定しないが，本稿では AngularJS を用いた場合について，ログ採取方法の説明およびアプリでのログ採取結果の検証を行う．なお，本稿で対象とする AngularJS のバージョンは 1.5.3 および 1.4.3 である．前者は執筆時点の最新の安定版である．後者は検証に利用したアプリのひとつである TodoMVC の ToDo アプリが依存しているバージョンであり，当該アプリが 1.5.3 では動作しなかったために利用している．2016 年現在，バージョン 1.0 系とは大きく仕様の異なるバージョン 2.0 系が開発中であるが，本稿では 2.0 系は対象としていない．

以下，フレームワーク層でのログ採取アプローチを AngularJS に適用する場合の方法を説明する．

3.1 ログ採取コードの手動埋め込み

AngularJS の本体である angular.js，および，後述のログ採取対象アプリで使用されている追加モジュールである angular-route.js と angular-resource.js に対し，人手でソースを分析して効果的なログ採取が可能であると考えられる処理ポイントを選び，ログ採取コードを埋め込んだ．埋め込んだ箇所は計 8 箇所となった．各モジュールのログ埋め込み箇所の内訳を表 1 に示す．

表 1　ログ採取コード手動埋め込み箇所概要

対象モジュール	埋め込み箇所数	埋め込み箇所概要
angular.js	5	アプリの処理の開始時
		処理依頼キューの実行時
		scope の作成時
		scope の function 実行時
		scope のプロパティ変更時
angular-route.js	2	URL の変更時（変更後の URL を表示）
		URL の変更時（適用するテンプレートの URL を表示）
angular-resource.js	1	リソースの要求時

このログ採取コードによる処理は上記のモジュール内で完結しており，アプリ固有部分への影響は無く，無改造で使用することができる．ログの出力には File System API を用い，表 2 に示す 3 項目をファイルに出力している．このうち ID は，そのアプリ実行時に引き継がれる一意な識別子で，アプリの実行時，すなわち JavaScript の処理開始時に，時刻情報を基にして作成している．タイムスタンプ

は，ログの順序の識別が主目的であるので，通常の Date オブジェクトによる時刻取得よりも高精度な High Resolution Time API [10]を用いてアプリ実行開始時からの経過時間を取得している．ログ内容に関しては，ログ採取コードを埋め込んだ箇所に対応する文字列を基本とし，一部ではパラメタも結合してログ内容の文字列としている．例えば，URL 変更時に採取するログにはパラメタとして変更後の URL を含めており，URL が異なるログはログ内容の異なる別種のログとして扱っている．よって，ログ採取コードを埋め込んだ箇所は合計 8 箇所だが，ログ内容の種類は 8 種類よりも多くなる．

表 2　ログ出力項目

項目名	説明
ID	アプリ実行開始時に採番した，実行開始から終了まで引き継がれる一意識別子
タイムスタンプ	アプリ実行開始時を 0 とするマイクロ秒単位の経過時間
ログ内容	ログ採取箇所およびその時点のパラメタに応じた文字列

3.2 ログ採取コードの自動埋め込み

ログ採取コードの手動埋め込みでは，あらかじめ AngularJS のソースを分析し，ログ採取に適切だと考えられる箇所を選定する作業が必要であった．この作業は技術的な難易度が高く作業量も多い．自動埋め込みでは，AngularJS のソース分析の作業を最小限に留めた上で適切なログを採取するためのコードを埋め込むことが目標となる．これを実現するため，AngularJS 本体のトップレベルの function（関数）群にログ採取コードを動的に埋め込むことを試みた．具体的には，angular.js 読み込み時に自動実行される初期処理内で function 文によって宣言された function 群が対象となり，JavaScript パーサーである Esprima [13]を用いてソースコードから function 群を抽出してログ採取コードを埋め込んだ．なお，今回はアプリ実行時にパースやログ採取コード埋め込みを行うようにしているが，事前にパースやログ採取コード埋め込みを行ってソースを改造しておき，実行時のオーバーヘッドを減らすアプローチもあり得る．

3.3 ログ採取コードの高度な自動埋め込み

ログ採取コードを AngularJS の function 群に一律に埋め込んだ場合，アプリ利用者の行動把握に寄与しないログも採取してしまう可能性が高い．特に，そのようなログの比率が高い場合は行動把握が困難になる上，無駄なログ採取によって CPU，メモリ，ストレージなどの資源を浪費する点も問題となる．よって，そのようなログを採取しないようにログ採取コードの自動埋め込みポイントを絞り込むことが望ましい．

自動埋め込みポイントの絞り込みの手法としては，ソースコードやログ出力内容の意味を解釈して，重要なポイントのみを残すようにする手法が考えられる．しかし，ここでは手動埋め込みで必要となる人手でのソースコード分析を不要にすることを目的としているので，ソースコードの情報は用いず，絞り込む前の自動埋め込みによるログ採取結果に含まれる，ログ種毎の出力回数とプロセス図のトポロジの情報を利用して絞り込む方針とした．

まず，ログ種毎の出力回数については，アプリ利用者の操作の回数に対するログ出力回数の比率を用いる．本稿でのログ採取の目的はアプリ利用者の行動把握であるので，各ログ種の出力回数は，アプリ内の操作回数にある程度近い範囲内に収まることが期待される．一方，操作回数に対して非常に多く出力されるログ種は，様々な処理内で共通に実行される AngularJS の内部処理に起因するログである可能性が高く，排除した方が良いと考えられる．ここで操作回数の把握とログ出力回数のしきい値の設定が問題となる．操作回数に関しては，クリック回数のような細粒度の情報ではなく，試行対象のアプリの試行を開始してから一連の試行を終了するまでを 1 回と数えるアプリの試行回数を用いる．すなわち，表 2 における ID が同一の範囲内が 1 回分の試行となる．また，ログ出力回数のしきい値については，試行対象の 2 つのアプリでログ採取コードの自動埋め込み手法によってログを採取し，両者のログ出力回数の傾向を考慮してしきい値を設定し，それを上回る回数が出力されているログ種を排除する．

次に，プロセス図のトポロジの観点からの絞り込みを検討する．ここでプロセス図とは，ログ採取結果を有向グラフとして可視化したものである．プロセス図の作成には Yano らの手法[11]を用いた．この手法では，ログファイルやデータベースなどのタイムスタンプと識別子を含むデータから一連のプロセスデータを抽出し，プロセス図の可視化と分析を可能にしている．プロセス図の例を図 1 に示す．図中，INITIAL_STATE と FINAL_STATE はアプリ試行の開始点と終了点を示しており，処理の実体はない．各ノードは表 2 のログ内容の文字列に対応しており，ID 毎にタイムスタンプによって整列されたログ内容ノードが有向グラフとして可視化される．各ノード間の遷移の矢印には数値のラベルが示され，その遷移が発生した回数を表す．プロセス図の作成には Graphviz [12]が利用され，図中のノードはおおよその遷移方向が上から下向きになるように自動配置される．ノード間の距離には特別な意味は無い．図 1 の例では，処理 A，処理 Z の順で実行したアプリ試行が 5 回行われ，処理 A，処理 M，処理 Z の順で実行したアプリ試行が 3 回行われたことを表している．

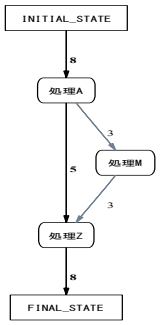

図 1　プロセス図の例

このような方法で可視化したプロセス図において，あるログ種が特定の処理の流れの中でしか出力されない場合，そのログ種の前のログ種と後のログ種は固定される．この時，プロセス図は直線状になり，そのログ種に対する入力方向の矢印は 1 本，出力方向の矢印も 1 本となる．一方，様々な処理の流れの中でそのログ種が出力される場合，前後のログ種も多様になり，入力方向や出力方向の矢印も多くなる．アプリの利用状況把握に寄与しないユーティリティ的な内部処理のログの場合，後者のようになり，入出力の数が多くなると予想される．よって，プロセス図上で入出力の数がしきい値を超えたログ種について，内部処理に起因するログとみなして排除する．

4. AngularJS 利用アプリのログ採取

前章で示した AngularJS のフレームワーク層での各ログ採取手法を実装した改造版の AngularJS モジュールを作成し, AngularJS 利用アプリに適用して各手法でのログ採取結果の評価と比較を行った．

4.1 ログ採取対象アプリ

AngularJS のフレームワーク層でのアプリログ採取結果の成否を評価するため，2 種類のアプリを用いてログ採取を試行した．ひとつは AngularJS 公式のチュートリアル用として用いられている PhoneCat アプリである[7][8]．もうひとつは，様々な JavaScript フレームワークの特徴を比較するために，各フレームワークを用いて同じ仕様の ToDo 項目管理アプリを開発し，そのソースコードを公開している TodoMVC [9]の AngularJS 版 ToDo アプリである．

4.2 PhoneCat アプリの概要

PhoneCat アプリはスマートフォンやタブレットの製品カタログを模したアプリであり，製品の一覧画面と，各製品の詳細画面とからなる．ただし，一覧画面と詳細画面との間の画面遷移において HTML は切り替わらず，同一の HTML 内で表示内容が切り替わる Single Page Application（以下，SPA）の構造になっている．PhoneCat アプリは情報参照が主体のアプリであり，文字入力の契機は一覧画面での表示絞り込みに用いるキーワード入力欄のみである．よって，このアプリの利用者の行動把握には，一覧画面で利用者がどの製品を選択したかが最重要の情報となる．

4.3 ToDo アプリの概要

ToDo アプリも PhoneCat アプリと同様に SPA であり，HTML の遷移は発生しない．相違点としては，PhoneCat アプリが情報の参照が主のアプリであるのに対し，ToDo アプリは利用者が入力した ToDo 情報を不揮発化して管理する，情報の入出力が主のアプリである．ToDo アプリの利用者の行動把握のためには，ToDo 項目の新規作成・完了・削除などの処理の流れが最重要の情報となる．

4.4 ログ採取コード手動埋め込み手法の適用結果

手動でログ採取コードを埋め込んだ AngularJS モジュールを作成して PhoneCat アプリと ToDo アプリに適用し，その状態で各アプリを試行してログを採取した．ログの採取結果を表 3 に示す．各アプリの試行では，アプリのある状態を起点とし，ボタン押下などの操作や画面遷移など，複数の手順を経て別に定めた状態に至るまでを 1 回の試行とし，1 回の試行が終わったらブラウザのリロードを実行して次の試行に移る．表 3 の 5 回の試行の各回の操作手順はすべて異なっているが，各回の操作手順は手順書にて規定しており，以降の別モジュールを用いたログ採取実験でもまったく同じ操作内容でログを採取している．

表 3　PhoneCat アプリと ToDo アプリにおける手動埋め込みによるログ採取結果概要

項目名	結果	
	PhoneCat アプリ	ToDo アプリ
アプリ試行回数	5	5
採取ログ行数	543	349
ログ種別数	48	76

図 2 は表 3 のログ採取結果のうち，PhoneCat アプリの分についてプロセス図として図示したものである．このプロセス図の構造として，右側上部の上下直線状になっている部分は，アプリが起動してモジュールを読み込む部分に対応している. 同一のアプリを 5 回試用した結果であるため，この部分には 5 回とも差異はなく，枝分かれの無いプロセスとなる．プロセス図下部の横に長い部分は，各製品の製品情報の読込みや，製品の画像の読込みに対応している．該当部分の拡大図を図 3 に示す．詳細に観察すると，製品一覧画面に対応する phone-list.html を起点として，各製品画面に遷移し, 再び一覧画面に戻るという PhoneCat アプリの主要なユースケースがプロセス図上でも確認できた．

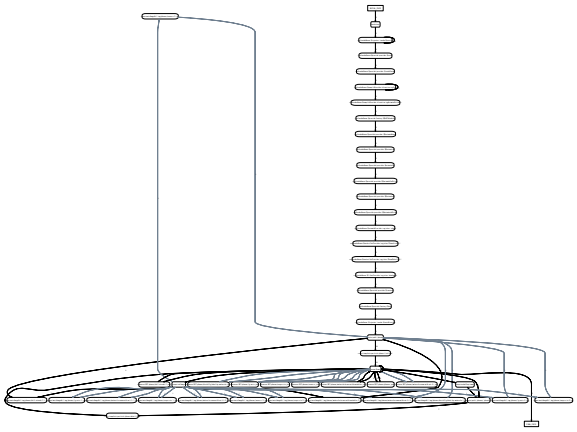

図 2　PhoneCat アプリでの手動埋め込みによるログ採取結果のプロセス図（全体）

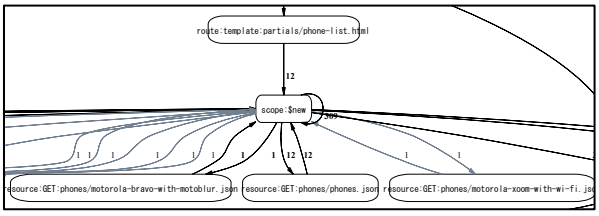

図 3　PhoneCat アプリでの手動埋め込みによるログ採取結果のプロセス図（画面遷移部）

次に，表 3 における PhoneCat アプリと ToDo アプリの結果を比較する．これらを比べると，アプリ試行回数は 5 回で共通ながら，ToDo アプリの採取ログ行数は PhoneCat アプリの約 6 割と少なく，逆にログ種別数は約 1.6 倍と多くなっている．採取ログ行数の差については，画面描画に伴って生成されるオブジェクトの数の差によって生じていると考えられる．そのオブジェクト生成に対応するログは“scope:$new”で，PhoneCat アプリでの出現回数は 347 回であるのに対し，ToDo アプリでは 54 回である．その差は 293 回であり，総ログ行数の差以上の差が“scope:$new”によって生じている．また，ToDo アプリの方がログ種別数が多くなっている点については，一部のログでパラメタの内容も含めてログ種別の文字列化しており，ToDo アプリでは ToDo 項目の入力操作時に，入力内容に応じて別種のログが出力されていることが原因と考えられる．

これらから，ログ採取コードの埋め込み手法を変更した場合の影響はアプリ毎に異なると考えられ，アプリ毎に結果を比較する必要があると言える．

4.5 ログ採取コード自動埋め込み手法の適用結果

PhoneCat アプリと ToDo アプリに，ログ採取コード自動埋め込みの改造を加えた AngularJS モジュールを適用してログを採取した結果を表 4 に示す．手動埋め込みによるログ採取結果の表 3 と比較すると，アプリ試行回数は 5 回で共通ながら，PhoneCat アプリでは採取ログ行数は 250 倍以上に増えており，ログ種別数も 10 倍以上となっている．採取されたログを調べると，AngularJS の内部処理用に実装されたユーティリティ的な function の実行ログが多く採取されていた．例えば，対象が文字列か否かを判定する function である isString については 16513 回実行され，その分のログが出力されていた．図 4 は表 4 のうち PhoneCat アプリの結果をプロセス図化したものであるが，isString のような内部処理のログによって非常に複雑になっており，利用者の行動把握は困難である．ToDo アプリへの適用結果でも同様の傾向であった．これは，トップレベルの function について一律にログを出力するようにしたことに起因するもので，利用者の行動把握に寄与しないログ種別を採取せずに有意なログだけを採取するように絞り込むことが必要と考えられる．

表 4　PhoneCat アプリと ToDo アプリにおける自動埋め込みによるログ採取結果概要

項目名	結果	
	PhoneCat アプリ	ToDo アプリ
アプリ試行回数	5	5
採取ログ行数	140948	140016
ログ種別数	521	285

図 4　PhoneCat アプリでの自動埋め込みによるログ採取結果のプロセス図（全体）

4.6 絞り込みを伴うログ採取コード自動埋め込み手法の適用結果

ログ採取コードの自動埋め込みポイントの絞り込みにはログ種毎の出力回数とプロセス図のトポロジの 2 種類の情報を用いる．これらの情報による絞り込みは同時に行うのではなく，まずログ種毎の出力回数によって絞り込み，その後でトポロジの情報を用いて更に絞り込むという 2 段階の絞り込みを行った．図 4 に示した通り，絞り込む前の自動埋め込みのログ採取結果のプロセス図は非常に複雑であり，ログ種毎の出力回数で一旦絞り込んだ後の方がトポロジの評価を行いやすいと判断したためである．

ログ種毎の出力回数を用いた絞り込みでは，前節の自動埋め込みによるログ採取結果を用いて絞り込み対象のしきい値を決定する．両アプリでのログ採取結果から検討した結果，アプリの試行回数である 5 回に対し，100 倍の 500 回以上の出力がいずれかのアプリで確認されたログ種別，および，20 倍の 100 回以上の出力が両アプリ共通で確認されたログ種別を絞り込み対象とした．このしきい値は 2 つ

のアプリのログ採取結果を観察して人為的に決定したものであるが，より多くのアプリでのログ採取結果を利用できる場合は，しきい値を自動的に決定するアプローチもあり得る．

絞り込みの改造を加えたログ採取コード自動埋め込みモジュールを PhoneCat アプリと ToDo アプリへ適用した結果の概要を表 5 に示す．絞り込み前の表 4 と比較すると，PhoneCat アプリでは採取ログ行数は約 1/19 となり，ログ種別数も約 13%減少した．同様に，ToDo アプリでは，採取ログ行数が約 1/19，ログ種別数が約 19%減少した．

表 5　PhoneCat アプリと ToDo アプリにおける
ログ出力回数を用いた絞り込み後の
自動埋め込みによるログ採取結果概要

項目名	結果	
	PhoneCat アプリ	ToDo アプリ
アプリ試行回数	5	5
採取ログ行数	7508	7319
ログ種別数	451	230

次に，プロセス図のトポロジを用いた絞り込みを行う．表 5 の出力回数によって絞り込んだログ採取結果からプロセス図を作成し，プロセス図上での各ログ種の入出力の数を数えてグラフ化したものが図 5 および図 6 である．横軸は各ログ種に対応しており，入出力数の降順で整列している．図 5 の PhoneCat アプリでは，入出力数の平均値は約 3.76，標準偏差は約 4.11，中央値は 2，最頻値は 2 である．一方，図 6 の ToDo アプリでは，入出力数の平均値は約 5.21，標準偏差は約 6.31，中央値は 3，最頻値は 2 である．

ログ出力回数による絞り込みでは，アプリの試行回数に対する相対的な比率をしきい値としていたため，PhoneCat アプリと ToDo アプリのデータを同列に用いることが可能であった．ログ種毎の入出力数に関しては，入出力数の多寡がアプリの内容や試行時の操作内容に影響を受けると考えられるので，図 5 と図 6 の結果を混合して用いるのは妥当でない可能性がある．しかし，ログを採取するポイントや，データ取得の際のアプリの試行回数は両者で共通しており，ログ種毎の入出力数の平均値なども大きな乖離はないことから，両者共通の基準を用いることとし，いずれかのアプリで入出力数が 9 以上（高い方の中央値の 3 倍）であるログ種，および両アプリで共通して入出力数が 4 以上（低い方の中央値の 2 倍）であるログ種を除外することとした．出力回数による絞り込みに加えて入出力数による絞り込みも行うログ採取コード自動埋め込みモジュールを PhoneCat アプリと ToDo アプリに適用した結果を表 6 に示す．また，PhoneCat アプリの結果についてプロセス図化したものを図 7 に示す．

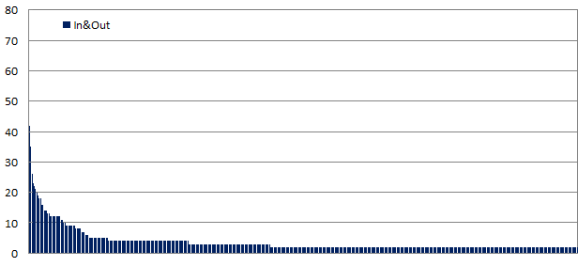

図 5　PhoneCat アプリのプロセス図における
ログ種毎の入出力数

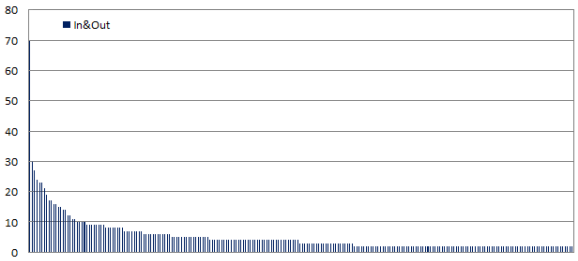

図 6　ToDo アプリのプロセス図における
ログ種毎の入出力数

表 6　PhoneCat アプリと ToDo アプリにおける
再絞り込み後の自動埋め込みによる
ログ採取結果概要

項目名	結果	
	PhoneCat アプリ	ToDo アプリ
アプリ試行回数	5	5
採取ログ行数	4336	3021
ログ種別数	395	174

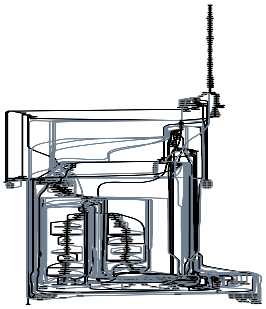

図 7　再絞り込み後の PhoneCat アプリでの自動埋め込みによるログ採取結果のプロセス図（全体：縦 50%圧縮）

4.7 手動埋め込み結果と自動埋め込み結果の比較

手動埋め込みによるログ採取結果と，自動埋め込みの結果がどの程度近いものになっているかを確認するため，プロセス図の内容と複雑度の観点で結果を比較した．

まず，プロセス図の内容については，前述の各アプリにおいて主要な処理の流れが可視化できているかどうかを目視で確認した．PhoneCat アプリでは，製品一覧画面から選択した製品詳細画面への遷移の様子が主要な処理の流れとなる．これは，手動埋め込みでは図 3 において可視化でき

ていたことを確認済みである．図 8 は，図 7 のプロセス図のうち, 図 3 に対応すると考えられる部分の拡大図である．図内で距離があるために表示していないが，図 8 に至る前の部分には“directiveNormalize:phone-listing”という製品一覧を示唆するノードが存在し，製品一覧を経て各製品の情報を含む JSON ファイルの読み込みに遷移するという，図 3 とほぼ同等の内容の可視化結果が確認できた．

ToDo アプリでは，ToDo 項目の新規作成・完了・削除などが主要な処理の流れとなる．ログ出力回数と入出力数によって絞り込みを加えた自動埋め込みによるログ採取結果のプロセス図では，ToDo 項目の作成に伴うオブジェクト生成を直接的に示すノードが確認できなかったが，ToDo 項目の追加・編集時に対応すると考えられるノードは現れていた．一部のログが採取できなかった原因としては，ログ採取コードの自動埋め込みが AngularJS のトップレベルの function のみを対象にしていたためと考えられる．これに対処するには，下層の function まで再帰的にログ採取コードを埋め込むように変更する必要がある．

以上より，ログ採取コードの自動埋め込みによるプロセス図は，手動埋め込みによるプロセス図と比較して，PhoneCat アプリの主要処理ではほぼ同等の可視化結果となり，ToDo アプリの主要処理では，やや不足な点はあるものの両者を対応付け可能な可視化結果となっており，プロセス図の内容の観点から，自動埋め込みによるログ採取について一定の有効性を確認できた．

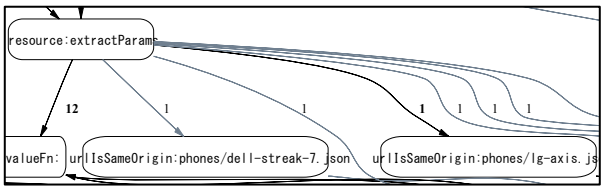

図 8　再絞り込み後の PhoneCat アプリでの自動埋め込みによるログ採取結果のプロセス図（画面遷移部）

次に，各手法で得られたプロセス図の複雑度を比較する．プロセス図は制御フローではないが有向グラフであることから，循環的複雑度（cyclomatic complexity）[14]を用いて評価する．循環的複雑度は以下のように定義される．

$$v = e - n + 2p$$

v: 循環的複雑度
e: エッジ数
n: ノード数
p: 接続されたコンポーネント数

プロセス図には特にコンポーネント構造は無いので，p=1 として良く，ノード数であるログ種別数と，プロセス図中のエッジ数から複雑度を計算できる．

プロセス図の複雑度の解釈について説明する．理想的なプロセス図はアプリ利用者の行動を過不足無く反映するので，利用者の行動に応じてプロセス図の正しい複雑度が決まる．ここで手動埋め込みによるログ採取が利用者の行動を正確に反映していると仮定すれば，そのプロセス図の複雑度が正しい複雑度とみなせ，自動埋め込みによって採取したログから作成したプロセス図がどの程度手動埋め込みに近い複雑度になるかによって自動埋め込み手法の良し悪しを評価できる．なお，循環的複雑度はプロセス図が一直線状，すなわち全ノードを一筆書きで辿るように片方向のエッジで接続した状態になった場合に，ノード数に関わらず最小値の 1 になる．最大値を取るのは全ノードの組み合わせが双方向のエッジで接続された場合だが，最大値はノード数に応じて変化する．よって，ノード数が変化する条件下では絶対値の大小の評価が難しいという問題はあるものの，正しいプロセス図と仮定した手動埋め込みのプロセス図と，自動埋め込みの各プロセス図との接近度合を評価する指標としては利用可能であると考えられる．

各ログ埋め込み手法から得られたプロセス図の循環的複雑度を表 7 に示す．両アプリとも傾向は共通しており，手動埋め込みが最も複雑度が低く，自動埋め込みは複雑度が非常に高いが，回数による絞り込み，および回数とトポロジを用いた絞り込みによって複雑度が段階的に減少している．回数とトポロジを用いた絞り込みの結果でも手動埋め込みよりは複雑度が高いが，絞り込み前と比べると大幅に複雑度が低減され，手動埋め込みの水準に近付いていると評価できる．

表 7　各プロセス図の循環的複雑度

ログ埋め込み手法	循環的複雑度	
	PhoneCat アプリ	ToDo アプリ
手動埋め込み	28	55
自動埋め込み	1939	1495
自動埋め込み＆絞り込み（回数のみ利用）	399	371
自動埋め込み＆絞り込み（回数とトポロジを利用）	206	232

5. 関連研究

JavaScript アプリの開発は, スマートフォンなどのモバイルデバイス用アプリの開発の一種として扱われることが多い[15]．本稿では対象をモバイルアプリに限定していないが, モバイルアプリでは通信速度の制約から SPA が採用されるケースが多いと考えられ，主要なユースケースのひとつと言える．モバイル分野では，Android などのプラットフォームを前提に，異常時のログを採取するロギング・フレームワークの研究がある[16]. [15]で比較されている通り，

モバイルアプリはネイティブ型と Web 型に大別されるが，[16]はネイティブ型のモバイルアプリを対象としており，本稿では Web 型のアプリを対象としている点で相違がある．また，[16]で扱っているのは異常時のログであり，本稿で扱っているアプリ利用ログとは異なっている．

ログ採取や可視化という点では，ビジネスプロセス管理（以下，BPM）[17]に関連した研究とも類似性がある．BPM での関心事は業務プロセスであり，定義済みの業務プロセスに関連したイベントを記録して可視化や分析を行う研究が多い[18][19]. 本稿で対象としているログも広義には業務プロセスに起因するものとみなせるが，業務プロセスの一部を成すアプリの，特にクライアント側での利用者の振舞いに着目したもので，関心の対象や粒度が異なる．

クライアント側での自動的なログ採取で，かつスクリプト言語による実装という点では，Microsoft Excel 上の利用者の行動把握を Visual Basic for Applications（VBA）でのログ採取によって試みた研究もある[20]．しかし，[20]で採取しているのはマウスクリックやキー押下，それに伴うセル操作といった比較的低レベルな操作についてのログであり，本稿で扱っているのはアプリが扱う業務に関する利用者の判断や振舞いについてのログであるので，レイヤが異なる．

6. まとめと今後の課題

アプリ利用者の行動把握のために必要なアプリの利用ログを，アプリを改造せずに採取するために，アプリの実行基盤となるフレームワーク層で採取する手法について検討した．

はじめに，対象としたフレームワークである AngularJS の構造を分析し，手動でログ採取コードを埋め込む手法を試み，埋め込み箇所が 8 箇所程度でも，アプリの主要な処理の流れが把握可能なログを採取できることを 2 つのアプリで確認した．続いて，フレームワークの構造を詳細に分析しなくても同様のログ採取が可能かを検証するため，各ソースコードの 1 箇所のみに改造を加え，アプリ実行時に動的にログ採取コードを埋め込む自動埋め込みの手法を試みた．単純な自動埋め込みでは，アプリ利用者の行動把握に寄与しない内部処理のログが大量に出力され，実用性の乏しい結果となった．しかし，そのログデータを基に，アプリ試行回数に対してログ出力回数が非常に多いものや，ログデータから可視化したプロセス図のトポロジから多種のログと相互作用を持っているものを内部処理に対応するログと判定して採取対象から除外することにより，2 つのアプリの主要機能について，手動埋め込みによるログ採取結果とほぼ同等の利用状況把握が可能で，絞り込まない場合よりも大幅に視認性の高い利用ログおよびプロセス図が得られた．絞り込みの効果に関しては，体感的な視認性の向上だけでなく，複雑度の指標の大幅な改善という点でも確認できた．手動埋め込みで採取されたログの一部は自動埋め込みでは採取されなかったが，この原因としては，自動埋め込みでは AngularJS のトップレベルの function 群にのみログ採取コードを埋め込んでいた点が大きいと考えられる．これに対処するには，自動埋め込みの方法を変えることも検討する必要があり，例えば，実行時の動的埋め込みではなく，ログ採取コードを埋め込んだフレームワークのソースコードを事前に生成するなどのアプローチが考えられる．この検討は今後の課題となる．その他の課題としては，AngularJS 上の他のアプリや，AngularJS 以外の他のフレームワークに対しても今回の手法が有効かどうかを検証することが挙げられる．

参考文献

[1] Apache Cordova: https://cordova.apache.org/
[2] Spiliopoulou, M.. Web Usage Mining for Web Site Evaluation: Making a site better fit its users. Communications of the ACM, August 2000, Vol.43, No.8, 127–134.
[3] AngularJS: https://angularjs.org/
[4] Backbone.js: http://backbonejs.org/
[5] React: https://facebook.github.io/react/
[6] Jain, N., Mangal, P. and Mehta, D.. AngularJS: A Modern MVC Framework in JavaScript. Journal of Global Research in Computer Science, 2014, Vol.5, No.12, 17-23.
[7] AngularJS Tutorial: https://docs.angularjs.org/tutorial
[8] AngularJS Phone Catalog Tutorial Application: https://github.com/angular/angular-phonecat
[9] TodoMVC: http://todomvc.com/
[10] High Resolution Time: https://www.w3.org/TR/hr-time/
[11] Yano, K., Nomura, Y., and Kanai, T.. A Practical Approach to Automated Business Process Discovery. 2013 17th IEEE International Enterprise Distributed Object Computing Conference Workshops (EDOCW), 2013, 53-62.
[12] Graphviz: http://www.graphviz.org/
[13] Esprima: http://esprima.org/index.html
[14] McCabe, T.J.. A Complexity Measure. IEEE Transactions on Software Engineering, December 1976, Vol.SE-2, No.4, 308-320
[15] Charland, A. and Leroux, B.. Mobile Application Development: Web vs. Native. Communications of the ACM, May 2011, Vol.54, No.5, 49-53.
[16] Cinque, M., Cotroneo, D. and Testa, A.. A Logging Framework for the On-Line Failure Analysis of Android Smart Phones. 1st European Workshop on AppRoaches to MObiquiTous Resilience (ARMOR'12), 2012, 2:1-2:6.
[17] Ko, R.K.L.. A Computer Scientist's Introductory Guide to Business Process Management (BPM). ACM Crossroads, 2009, Vol.15, No.4, 11-18.
[18] Wetzstein, B., Leitner, P., Rosenberg, F., Brandic, I., Dustdar, S. and Leymann, F.. Monitoring and Analyzing Influential Factors of Business Process Performance. 2009 IEEE International Enterprise Distributed Object Computing Conference (EDOC'09), 2009, 141-150.
[19] Reichert, M., Kolb, J., Bobrik, R. and Bauer, T.. Enabling Personalized Visualization of Large Business Processes through Parameterizable Views. 26th Symposium On Applied Computing (SAC'12), 2012, 1653-1660.
[20] Bishop, B, and McDaid, K.. Unobtrusive Data Acquisition for Spreadsheet Research. IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC) 2008, 139-142.