論文 / 著書情報
Article / Book Information

論題(和文)	実行トレース抽象化を目的とした参照関係・アクセス解析によるコアオブジェクト特定
Title(English)	
著者(和文)	野田訓広, 小林隆志, 渥美紀寿
Authors(English)	Kunihiro Noda, Takashi Kobayashi
出典(和文)	情報処理学会 情処研報, Vol. 2017-SE-195, No. 2, pp. 1-8
Citation(English)	, Vol. 2017-SE-195, No. 2, pp. 1-8
発行日 / Pub. date	2017, 3
権利情報 / Copyright	

Vol.2017-SE-195 No.2
2017/3/12

実行トレース抽象化を目的とした参照関係・アクセス解析によるコアオブジェクト特定

野田 訓広[1,a)]　小林 隆志[1,b)]　渥美 紀寿[2,c)]

概要：オブジェクト指向システムのプログラム理解では，実行履歴を解析し，シーケンス図等でシステムの振る舞いを可視化するアプローチが有効である．しかし，実行履歴には膨大な情報が含まれているため，復元されたシーケンス図は巨大で実用的なものとならない．この問題を解決するため，本稿では，実行トレース抽象化を目的とした，参照関係・アクセス解析によるコアオブジェクト特定手法を提案する．提案手法では，オブジェクトの参照関係およびアクセス頻度の特徴に基づき，各オブジェクトに対して重要度を推定する．重要度の高いオブジェクト（コアオブジェクト）を中心にオブジェクトをグループ化し，システムの振る舞いをグループ間相互作用として可視化する．これにより，システムの重要な振る舞いを含む読解可能なサイズのシーケンス図が生成される．複数のオープンソースソフトウェアに提案手法を適用し，シーケンス図のライフライン数の削減率，および，シーケンス図に含まれる振る舞いの特徴を分析することで，提案手法の有用性を議論する．

1. はじめに

プログラム保守においては，保守対象ソフトウェアの構造・振る舞いを十分に理解することが必要である．プログラム理解には，仕様書や設計書といったドキュメントが有用であるが，システムの度重なる改変や，時間・コストの制約を理由に，ドキュメントがシステムの最新状態を正確に反映していないことも多くある．

この問題を解決するため，実行トレースを解析することで，仕様復元 [1,2] や設計復元 [3] を行う研究が行われている．特に，プログラムの振る舞い理解支援に関しては，オブジェクトの相互作用をシーケンス図として可視化するアプローチが有効である [3]．一般に，実行トレースの情報量は膨大であるため，復元したシーケンス図も巨大なものとなる．実行トレースに対して何らかの抽象化を行い，実用的なサイズのシーケンス図を復元することが重要となる．

実行履歴から復元するシーケンス図は，システムの実行時間に比例して縦方向サイズが大きくなり，生成されたオブジェクト数に比例して横方向サイズが大きくなる．既存研究の多くは，縦方向のサイズ削減に焦点を当てた情報削減手法 [4–10] や，巨大な可視化情報の効果的な探索手法 [11–14] を提案している．実用性を確保するためには，横方向のサイズ削減も重要であるが，この点に焦点を当てた研究は少ない [7, 15–17]．このため，横方向のサイズ削減に焦点を当てた手法の開発・改良が必要である．

本稿では，実行履歴から復元するシーケンス図の横方向サイズの削減を目的とした，参照関係・アクセス解析によるコアオブジェクト特定手法を提案する．提案手法では，まず，戸田らの手法 [18] を利用して実行時に大量に生成される一時オブジェクトを特定・除去する．次に，非一時オブジェクトに対するアクセス頻度を解析し，オブジェクト毎の重要度を推定する．生存期間が長く高頻度でアクセスされるオブジェクトは，システムにおいて重要な役割を担うと想定される．我々はそのような重要な役割を担うオブジェクト（コアオブジェクト）を特定する．

重要度推定によりコアオブジェクトを特定した後，コアオブジェクトを中心にオブジェクトをグループ化する．システムの振る舞いをグループ間相互作用として可視化することで，コアオブジェクトを中心とした振る舞いを表現する，横方向のサイズが削減されたシーケンス図を生成する．

提案手法の有用性を評価するため，提案手法を複数のオープンソースソフトウェア (Open Source Software (OSS)) に適用した．提案手法により，設計を理解する上で重要なオブジェクトは保ったまま，シーケンス図に含まれるライフライン数を大きく削減（86%以上の削減率）することが

[1] 東京工業大学
Meguro-ku, Tokyo 152–8552, Japan
[2] 京都大学
Sakyo-ku, Kyoto 606-8501, Japan
[a)] knhr@sa.cs.titech.ac.jp
[b)] tkobaya@cs.titech.ac.jp
[c)] atsumi.noritoshi.5u@kyoto-u.ac.jp

でき，提案手法の有用性が確認できた．

本稿の主要な貢献点は以下の通りである．

- アクセス頻度に基づくオブジェクトの重要度推定式の提案．
- オブジェクトの参照関係グラフ (Reference Relation Graph (RRG)) 上での，コアオブジェクトを中心としたオブジェクトグループ化手法の提案．グループ間相互作用のみを可視化することで，復元するシーケンス図のサイズを大きく削減できる．
- 複数の OSS を用いた有用性評価．既存手法との比較を行い，提案手法がシーケンス図の横方向サイズの削減に関して優れた性能を持つことを確認した．

以降では，2 節で主要な関連研究について述べた後，3 節で提案手法の詳細を説明する．4 節で評価実験について述べ，5 節 で手法・評価の妥当性について議論する．最後に，6 節で本稿のまとめを述べる．

2. 関連研究

実行トレース抽象化手法の多くは，生成されるシーケンス図のメッセージ数削減（縦方向のサイズ削減）に焦点を当てている．繰り返し処理の圧縮 [4, 8, 13] や実装詳細の除去 [7] を行ったり，実行トレースを複数のフェイズ（機能・タスクのまとまり）に分割 [6, 9, 10, 19] したりすることでメッセージ数を削減する手法が提案されている．

一方，シーケンス図のオブジェクト数削減（横方向のサイズ削減）に焦点を当てた研究は少ない．提案手法は，オブジェクト数削減を行うタイプに分類される．以降，提案手法と，既存の横方向の情報削減を行う手法を比較する．

Hamou-Lhadj らは，fan-in/fan-out からメソッド毎の utilityhood を計算し，実装詳細（utilityhood 値が高）を除去する手法を提案している [7]．実装詳細のメソッドを除去することで，シーケンス図の縦横双方向の情報削減が行われる．Hamou-Lhadj らの手法は，コア/非コア（非実装詳細/実装詳細に対応）の判別という点で，我々の提案手法と共通点がある．しかし，現実のソフトウェアには，同じ fan-in/fan-out 数を持つメソッドが多数存在するために，Hamou-Lhadj らの手法では横方向のサイズ削減が適切に行えないことが多くある．4 節にて，OSS を用いて，Hamou-Lhadj らの手法と提案手法の詳細な比較を行い，横方向の情報削減に関して提案手法がより優れていることを示す．

Dugerdil らは，単位時間当たりのオブジェクト間の相互作用頻度に基づき，オブジェクトをクラスタリングする手法を提案している [15]．クラスタ間相互作用のみを可視化することで，抽象化レベルの非常に高いシーケンス図が生成される．Dugerdil らの手法は，プログラム理解の初期段階において，粗粒度のアーキテクチャレベルでの振る舞い（e.g., レイヤードアーキテクチャにおけるレイヤ間の相互作用）を理解したい場合に有効である．それに対して我々の提案手法は，より細粒度の振る舞いの理解支援に焦点を当てている．提案手法は，特定の機能・モジュールにおいて重要な役割を担うオブジェクトを特定し，そのオブジェクト間の相互作用を通して機能・モジュールの振る舞いを理解したいような場面で有効である．

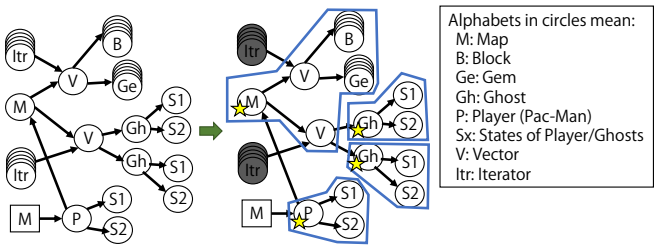

図 1 Running example of our technique (Pac-Man game example)

我々もこれまでに，設計意図の抽出とそれに基づく実行トレース抽象化手法を提案してきている [16, 17]．ソースコードから設計パターンを抽出し，各パターンによって実現される設計意図を考慮したオブジェクトグループ化方法を提案している．機械的な抽象化に比べ，設計意図を考慮しているためプログラム理解への有用度が高いことが見込まれる一方で，静的解析に頼った手法であるため，一定の false positive を生じる．振る舞い理解支援のためには，実行シナリオに特化した動的情報の解析も重要である．我々がこれまでに提案してきた設計パターンに基づく抽象化手法が静的解析に焦点を当てているのに対し，本稿で提案する手法は動的解析に焦点を当てたものである．

3. コアオブジェクト特定による実行トレース抽象化

3.1 手法概要

提案手法は以下のステップから成る．

(1) 参照エスケープ解析 (Reference Escape Analysis (REA)) による一時オブジェクトの削減
(2) アクセス頻度解析によるオブジェクトの重要度推定
(3) RRG 上でのオブジェクトグループ化
(4) グループ間相互作用としてシステムの振る舞い可視化

各ステップの詳細を，以降の節で説明する．

以降では，実行トレースが Behavior model (B-model) [20] に基づくイベントシーケンスとして表現されているものとする．B-model はオブジェクト指向システムの振る舞いを表現するモデルであり，複数のイベント要素から構成される．イベント要素の例としては，*ConstructorEntry* / *ConstructorExit*（コンストラクタ進入/退出），*VariableDefinition* / *VariableReference*（変数定義/参照）などがある．B-model を用いると，実行トレースは $\langle b_1, b_2, ..., b_n \rangle$ の形式で表現できる．ここで，b_i は B-model のイベント要素である．

図 1 に提案手法の例を示す．図 1 の有向グラフは，Pac-Man ゲームに対する RRG である．丸・矩形はそれぞれ，非

静的・静的オブジェクトを表す．エッジは，オブジェクトのフィールドを介した参照の方向を表す．Pac-Man ゲームでは，プレイヤーがマップ上を，ゴーストとの接触を避けながら宝石を集めて動き回る．マップオブジェクトは，ブロック・ゴーストオブジェクトをベクターオブジェクトを介して保持する．プレイヤーとゴーストには，複数の状態（通常状態・死状態等）が存在する．プレイヤーオブジェクトはシングルトンであり，マップの静的オブジェクトから参照される．提案手法の適用により，図 1 に示すように RRG が複数のサブグラフへ分解されることになる．グラフの分解プロセスについては，以降の節で詳細を説明する．

3.2 REA による一時オブジェクト除去

戸田らの手法 [18] を応用した参照エスケープ解析 (REA) により，一時オブジェクトを特定・除去する．

REA では，戸田らの手法と同様に，オブジェクトの参照関係及びアクセス期間を解析する．以下の性質を共に満たすオブジェクト o を，一時オブジェクトとして特定する．

- o は他のどのオブジェクトからも参照されていない
- 生存期間 $Lifetime(o)$ が閾値 L_t 以下

ここで，$Lifetime(o)$ は，実行トレース中で，o が最初にアクセスされてから最後にアクセスされるまでの期間である．

REA で解析する B-model のイベントは，*MethodEntry/Exit*，*ConstructorEntry/Exit*，および，フィールドに対する *VariableReference/Definition* である．提案手法では，解析対象ソフトウェアに，これらのイベントを検出するためのロギングコード計装が必要となる．上記以外の B-model のイベント（e.g., 局所変数に対する *VariableReference/Definition*，*LoopStart/End* 等）については，REA に必要がなく，それらを検出するための計装の必要もない．

図 1 の例では，イテレータオブジェクト及びマップの非静的オブジェクトが，他のどのオブジェクトからも参照されない．この時，イテレータオブジェクトは，生存期間が短いため，一時オブジェクトとして判定・削除される．一方，マップの非静的オブジェクトについては，生存期間が長いため，非一時オブジェクトと判定され削除されない．

3.3 アクセス頻度に基づくオブジェクトの重要度推定

非一時オブジェクトに対し，アクセス頻度を解析することでオブジェクトの重要度を推定する．システムにおいて重要な役割を担うオブジェクトにはアクセスが集中すると想定される．そこで提案手法では，以下の式により，アクセス頻度が高いほど重要なオブジェクトであると推定する．

$$Importance(o_i) = (w_w + w_r) \,/\, (w_w \frac{1}{WF(o_i)} + w_r \frac{1}{RF(o_i)})$$

$Importance(o_i)$ は，書き込みアクセス頻度 $WF(o_i)$ と，読み込みアクセス頻度 $RF(o_i)$ との重み付き調和平均である．$WF(o_i)$ と $RF(o_i)$ は以下のように定義する．なお，O は全オブジェクトの集合を表す ($O = \{o_i \mid 1 \le i \le n\}$)．

$$WC(o_i) = (o_i \text{ への連続しない書き込みアクセス回数})$$
$$RC(o_i) = (o_i \text{ への読み込みアクセス回数})$$
$$WF(o_i) = (WC(o_i) - WC_{\min}(O)) \,/\, (WC_{\max}(O) - WC_{\min}(O))$$
$$RF(o_i) = (RC(o_i) - RC_{\min}(O)) \,/\, (RC_{\max}(O) - RC_{\min}(O))$$

プログラム理解支援の観点から，状態変化を引き起こす書き込みアクセスは，読み込みアクセスよりも重要であると考え，重みは $w_w > w_r$ となるように選ぶ (e.g., $w_w = 3, w_r = 1$)．$WC(o_i)$ は，連続する書き込みアクセスはカウントしない．すなわち $WC(o_i)$ は，他オブジェクトから読み取られる o_i の状態変化の回数に等しい．$WF(o_i)$ と $RF(o_i)$ はそれぞれ，$WC(o_i)$ と $RC(o_i)$ を [0,1] に正規化した値である．読み込みアクセス回数は，書き込みアクセス回数に比べ非常に大きくなる傾向にあるため，[0,1] にアクセス頻度を正規化することが必要である．

なお，$WF(o_i)$ または $RF(o_i)$ が 0 の場合，$Importance(o_i) = 0$ とする．$WC_{\max}(O) = WC_{\min}(O)$ の場合は $WF(o_i) = 1$ とし，$RC_{\max}(O) = RC_{\min}(O)$ の場合は $RF(o_i) = 1$ とする．

最終的に，重要度の値が閾値 I_t より大きいオブジェクトを，コアオブジェクトであると判定する．

図 1 では，マップの非静的オブジェクト，プレイヤー，ゴーストが高頻度でアクセスされるため，コアオブジェクトであると判定されている．図 1 における星マークは，そのオブジェクトがコアオブジェクトであることを示す．

3.4 RRG 上でのオブジェクトのグループ化

重要度推定の後，コアオブジェクトを中心としてオブジェクトをグループ化する．コアオブジェクトと関連の強いオブジェクト群を特定し，それらをグループ化する．

関連の強いオブジェクトを特定するため，オブジェクトの生存期間と RRG を利用する．ここで，$RRG = (V, E)$ は有向グラフであり，V はノードの集合，E はエッジの集合である．V は O に等しい．o_i が o_j を参照する場合，o_i から o_j へのエッジ $e \in E$ が存在する．

RRG を探索することで，コアオブジェクトと同じライフサイクルを持つ（i.e., コアオブジェクトとコンポジション関係（≠集約関係）にある）オブジェクト群を見つける．同じライフサイクルを持つオブジェクト群は相互に関連が強いことが想定できるため，そのオブジェクト群をグループ化する．このグループ化により，RRG が複数のサブグラフに分解されることになる．各サブグラフのルートノードはコアオブジェクトである．サブグラフ内のオブジェクト群は，同じライフサイクルを持つ．一般に，オブジェクト間の参照関係だけでは（RRG だけでは）コンポジション関係と集約関係を区別することはできない．そのため，

Algorithm 1 Object Grouping in a RRG

Input: Core Objects: $CO = \{co_i \mid 1 \leq i \leq m\}$
Reference Relation Graph: $RRG = (V, E)$
Output: Object Groups: $\mathcal{G} = \{G_1, G_2, ..., G_m\}$
1: **for all** $co_i \in CO$ **do**
2: $G_i \leftarrow \{o_s \mid o_s \in V$
$\wedge\ o_s$ is reachable from co_i in RRG
$\wedge$ no paths from co_i to o_s in RRG contain $o_t \in V$
s.t. $(o_t \neq co_j (i \neq j) \wedge Lifetime(o_t) \leq Lifetime(co_i))\}$
3: **for all** $o_i \in V$ **do**
4: **if** o_i belongs to more than two groups **then**
5: $\mathcal{K} \leftarrow \{k \mid o_i \in G_k\}$
6: $LD(G_k, o_i) \leftarrow Lifetime(co_k) - Lifetime(o_i)$
7: $k_{\min} \leftarrow$ one of elements in $\arg\min_{k \in \mathcal{K}} LD(G_k, o_i)$
8: **for all** $l \in \{l \mid o_i \in G_l \wedge l \neq k_{\min}\}$ **do**
9: remove o_i from G_l

3.2 節で計算したオブジェクトの生存期間を利用することでコンポジション関係と集約関係を区別する．

Algorithm 1 に，グループ化アルゴリズムを示す．Algorithm 1 では，“コアオブジェクト” と，“コアオブジェクトから（直接的/間接的に）参照されるオブジェクトで，生存期間がコアオブジェクトと同じかそれ以下のもの” がコンポジション関係にあると判定している．

可視化の都合上，あるオブジェクトが複数グループに所属しないようにする必要がある．システムの振る舞いをグループ間相互作用として可視化するため，もし，あるオブジェクト o_i が複数のグループ G_s, G_t に所属し，$o_j \in G_u$ から o_i へのメッセージが存在すると，G_s か G_t どちらのグループが G_u からのメッセージを受信すべきか決定できなくなってしまう．これを避けるため，o_i が複数グループの所属候補となった場合は，o_i を，o_i と最も近い生存期間を持つコアオブジェクトのグループに所属させるようにする．

図 1 の例では，RRG が 4 個のサブグラフ（オブジェクトグループ）に分解される．

3.5 グループ間相互作用として振る舞いを可視化

オブジェクトグループ化後の可視化方法は，我々の既存研究 [17] と同様である．オブジェクトグループ G_i を 1 つのライフラインとして，シーケンス図上で描画する．G_i 内のコアオブジェクトのインスタンス id を，ライフラインの上部矩形内に表示する．グループ間相互作用のみを可視化し，グループ内相互作用は可視化しない．

図 1 の例では，各サブグラフがそれぞれ 1 つのライフラインとして描画され，サブグラフ間の相互作用のみがシーケンス図上に可視化される．

4. 実験

提案手法の有用性を評価するため，複数のオープンソースソフトウェアに手法を適用する．ロギングコードの計装には，*REMViewer* [21] の一部である *SELogger* を利用する．Hamou-Lhadj らの utilityhood に基づくトレース抽象化手法 [7] を baseline 手法とし，提案手法との比較を行う．

4.1 Research Questions

RQ_1 では，提案手法の主目的である，実行履歴から復元するシーケンス図の横方向の削減性能に関して調査する．

> ***RQ*$_1$:** 実行履歴から復元するシーケンス図の横方向の削減性能に関して，提案手法は baseline 手法に比べどれだけ優れているか？

次に，提案手法の中の 2 つの削減ステップ（一時オブジェクト削除・非コアオブジェクト削除）それぞれが，削減性能にどのような影響を与えているか更なる調査を行う．

> ***RQ*$_2$:** 一時オブジェクト削除・非コアオブジェクト削除はそれぞれ，提案手法の情報削減性能にどのような影響を与えているか？

提案手法はロギングコードの計装を必要とするため，実行時オーバーヘッドを伴う．実用性の観点から，RQ_3 では，どの程度のオーバーヘッドが生じるのかを調査する．

> ***RQ*$_3$:** 提案手法の実行時オーバーヘッドはどの程度か？

提案手法の目的は，システムの重要な振る舞いを含んだコンパクトなシーケンス図を実行履歴から復元し，プログラム理解支援に繋げることにある．RQ_4 では，生成されたシーケンス図中に，設計を理解する上で重要なオブジェクトがどの程度の割合で含まれているのか，そこからどのような振る舞いが読み取れるのかについて調査する．

> ***RQ*$_4$:** 生成されたシーケンス図には，設計を理解する上で重要なオブジェクトがどの程度含まれているか？そこからどのような振る舞いが読み取れるか？

4.2 実験の設定

4.2.1 対象システムと実行シナリオ

実験対象を**表 1** に示す．提案手法中の各パラメータは次の通り設定する．$L_t = Lifetime_{\max}(O) \cdot 0.7, w_w = 3, w_r = 1.$

4.2.2 評価尺度

各実行シナリオに対して，コアオブジェクト特定における ground truth (O_{gtcore}) を事前に定義する．実験対象システムのドキュメント・ソースコードから，設計上重要なクラス (C_{gtimp}) を抽出し，そのクラスのオブジェクトを，コアオブジェクト特定における ground truth とする．

実験対象システムに対する C_{gtimp} を**表 2** に示す．

jpacman に対しては，実行シナリオの説明中に現れるキーワード（プレイヤー・パワークッキー等）を重要クラスとして定義している．加えて，ゲームシステム全体の管理を行う（ゲーム上の要素の更新・再描画処理の起点となる）

表 1　対象システムと実行シナリオ

Project	Ver.	LOC	Scenario
jpacman [1]	rev.53	5.9K	アプリケーション起動，ゲーム開始，パックマンを操作してパワークッキーを取得・パワーステートへ移行，アプリケーション終了．
JHotDraw [2]	7.6	58.5K	アプリケーション起動，矩形を描画，角丸矩形を横に描画，アプリケーション終了．
Ant [3]	1.9.8	224.3K	Ant 自身をビルド（build.xml で定義された bulid タスクを実行）．

[1] `http://code.google.com/p/jpacman/` [2] `http://www.jhotdraw.org/` [3] `http://ant.apache.org/`

表 2　対象システムにおいて設計上重要なクラス

Project	Important classes (C_{gtimp})
jpacman	*Player, Ghost, EatGem, Player$State, Ghost$State, SimpleLevel*
JHotDraw	*DrawingView, Drawing, DrawingEditor, CreationTool, (Round)RectangleFigure*
Ant	*Project, Target, UnknownElement, RuntimeConfigurable, Task, IntrospectionHelper, ProjectHelper2*

SimpleLevel クラスを重要クラスに含めている．

他のプロジェクトについては，開発者向けドキュメント・ソースコードにおいて，design overview，most important classes などのキーワードを用いて言及されているクラスを，設計上重要なクラスとしている（抽出方法は，既存研究 [22, 23] と同様である）．今回の実験では，実行シナリオに基づき，開発ドキュメントで言及されているクラスセットを次のように一部除外・具体化している．*JHotDraw* については，選択・リサイズハンドルを用いた操作を行わないため *Handle* を除外，*Tool*・*Figure* を *CreationTool*・*(Round)Rectangle* に具体化した．*Ant* については，今回のシナリオで利用されない *ProjectHelperImpl* を除外した．

O_{gtcore} に基づき，評価尺度を以下のように定義する．

$$Recall = (\sum_{c \in C_{\text{gtimp}}} h(c, SD)) \;/\; |C_{\text{gtimp}}|$$

$$Precision = \#\{o \mid o \text{ is an instance of } c \in C_{\text{gtimp}}\} \;/\; \#lifelines$$

$h(c, SD)$ は，シーケンス図 SD に，クラス c の型（派生型含む）のコアオブジェクトが含まれているときに 1，それ以外の場合に 0 となる関数である．*Recall* は，シーケンス図に含まれているオブジェクトが，重要クラス C_{gtimp} をどの程度網羅するかを表す．*Precision* は，シーケンス図に含まれるライフラインの何割が，重要クラス C_{gtimp} のインスタンスに対応するかを表す．

4.2.3　計装範囲

対象システムに固有の振る舞いのみが可視化されるように，対象システムにおいて定義されたクラスにのみロギングコードを計装し，ライブラリコードには計装を行わない．ただし，コレクションライブラリに関しては以下の理由から計装を行う．コレクションデータを扱うために，コレクションライブラリ（*java.util.ArrayList* 等）が利用されることが多くある．提案手法はオブジェクトの参照関係を解析するため，コレクションライブラリを介した参照関係が追跡できないと，解析精度が著しく低下してしまう．これを避けるため，コレクションライブラリに関しては計装対象とする．なお，コレクションオブジェクト o_c へのアクセスは，o_c を所有する（フィールドを介して参照する）オブジェクトへのアクセスであると見なし，重要度推定を行う．

表 3　Collected runtime information

Project	Events	Messages	Loaded classes	Objects
jpacman	34,491K	16,179K	42	1,354
JHotDraw	906K	468K	216	3,302
Ant	44,586K	23,067K	250	40,305

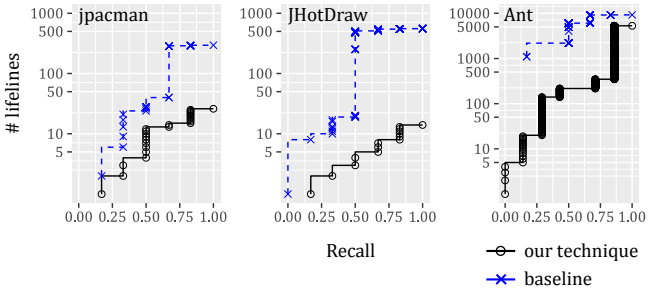

図 2　Recall vs. # lifelines

4.3　結果と評価

記録された実行時情報を**表 3** に示す．メッセージ数とオブジェクト数が，シーケンス図のサイズに影響を与える．

4.3.1　RQ_1 への回答

提案手法と baseline 手法とで，シーケンス図の横方向サイズの削減性能を比較する．プログラム理解に有益となる O_{gtcore} をできる限り削減せずに，いかにオブジェクト数を減らすことができるかが重要である．

推定したオブジェクト重要度の降順（baseline 手法では utilityhood 値の昇順）にオブジェクト（メソッド）を並べ，コア/非コアを区別する閾値を変化させることで，生成されるシーケンス図の抽象度が変化する（図中のライフライン数が変化する）．閾値を変化させたときの，シーケンス図に含まれるライフライン数と *Recall* の関係を**図 2** に示す．

図 2 から，提案手法は baseline 手法に比べ，*Recall* の上昇に対するライフライン数の上昇率が小さいことが分かる．*Recall* = 1 の場合のライフライン数を比較してみると，**表 4** のようになる．*Recall* = 1 においては，提案手法のライフライン数は baseline 手法の 2.5%–57.0%に抑えられている．このことから，実行履歴から復元するシーケンス図の横方向サイズの削減性能に関して，提案手法は baseline 手法よりも優れていることが分かる．

表 4　Final Results of Trace Summarization (*Recall* = 1)

	Our technique		Baseline	
Project	# lifelines	Comp.[1]	# lifelines	Comp.[1]
jpacman	28	97.93%	292	78.43%
JHotDraw	14	99.55%	559	83.07%
Ant	5,283	86.89%	9,196	77.18%

[1] Compression ratio calculated by
1 − (# lifelines in the resulting diagram)/(# all objects)

表 5　Results of pruning temporaries

Project	# temporaries	# non-temporaries	Comp. ratio[1]
jpacman	280	1,074	20.68%
JHotDraw	1,582	1,720	47.91%
Ant	18,732	21,573	46.48%

[1] Compression ratio calculated by 1 − (# non-temp.)/(# all objects)

また，閾値の選択に関して柔軟性の違いが見られる．図 2 から分かるように，baseline 手法では，閾値を少し変化させると，ライフライン数が急激に変化する．これは，同一の fan-in/fan-out 数を持つメソッドが多数存在するため，utilityhood 値に基づくランキング表において，同順位の項目が多数発生してしまうことに起因する．一方，提案手法では，閾値を変化させても大きくライフライン数は変化しない．これは，オブジェクト重要度によるランキング表では比較的同順位が発生しづらいためである．提案手法は，開発者の理解度に合わせたより柔軟な閾値変更（シーケンス図の抽象化粒度変更）が可能であることが分かる．

実行履歴から復元するシーケンス図の横方向の削減性能に関して，提案手法は baseline 手法よりも優れている．*Recall* = 1 のとき，提案手法のライフライン数は baseline 手法の 2.5%–57.0%に抑えられる．

加えて，提案手法は，開発者の理解度に合わせたより柔軟な抽象度変更（閾値変更）が可能である．

4.3.2　RQ_2 への回答

一時オブジェクト削除･非コアオブジェクト削除それぞれの削減性能を確認する．**表 5** に一時オブジェクト削除の結果を示す．非コアオブジェクト削除に関しては，*Recall* = 1 の場合の結果（表 4）を参照して議論する．

一時オブジェクト削除により，オブジェクト数が 20.68%–47.91%削減されたが，依然として，1,000 以上のオブジェクトが残存した．このことから，一時オブジェクト削除は，シーケンス図の横方向サイズの削減に貢献するが，それ単独では削減性能が十分でないことが分かる．一方，非コアオブジェクト削除に関しては，オブジェクト数を 86.89%–99.55% と大きく削減することができている．このことから非コアオブジェクト削除が，シーケンス図の横方向サイズの削減に大きく貢献していることが分かる．

ここで，非コアオブジェクト削除単体での効果を調べるために，一時オブジェクトを含む全オブジェクトに対する重要度推定を行った．その結果，いくつかのプロジェクトでは，重要度のランキング上位に，一時オブジェクトが大量に出現する結果となった．例えば，*JHotDraw* の場合には，重要度のランキング 19 位付近に，*ReversedList* クラスのインスタンスが 67 個出現した．大きなコレクションデータの探索に利用する一時オブジェクトなど，いくつかの一時オブジェクトは，高頻度にアクセスされる傾向にあり，一時オブジェクト削除がそれらのノイズとなるオブジェクトを除外する役割を果たしていることが確認できた．

表 6　Execution time

Project	Base	With logging	Overhead
jpacman	5.677s	9.956s	75.4%
JHotDraw	2.489s	5.811s	133.5%
Ant	16.781s	36.610s	118.2%

一時オブジェクト削除・非コアオブジェクト削除の 2 ステップは共に，提案手法の情報削減性能に貢献している．どちらか 1 ステップのみの実施では不十分であり，2 ステップを組み合わせて情報削減することが，実行履歴から復元するシーケンス図の有用性を高めるために必要である．

4.3.3　RQ_3 への回答

表 6 に実行時オーバーヘッドを示す．各実行シナリオに対し，それぞれ 10 回繰り返し測定を行い，平均値を記載している．‘Base’ 列・‘With logging’ 列はそれぞれ，ロギングコードの計装無し・有りの状態での実行時間を示している．計測時に利用した計算機は，Intel Core i-7-4790K 4.00GHz, 32.0GB RAM である．*SELogger* のロギングオプションは，実行トレースの記録に 4 バックグラウンドスレッドを利用，非圧縮形式でトレースを記録するように設定した．

提案手法は 75.4%–133.5%の実行時オーバーヘッドを伴う．これは，近年のスケーラブルな動的解析手法と比べ，比較的小さいものである．例えば，feedback-directed instrumentation により crash paths を計算する手法 [24] では，多くの場合に 100%–800%のオーバーヘッドを伴い，スケーラブルにスレッド間共有データを検出する record and replay システム [25] では，平均で 480%のオーバーヘッドを伴う結果が報告されている．

提案手法において開発者は，振る舞い可視化のために，計装されたアプリケーションを 1 度実行すればよいだけである．多くの場合において，提案手法のオーバーヘッドは許容できるものであると考える．

提案手法は 75.4%–133.5%の実行時オーバーヘッドを伴う．これは，近年のスケーラブルな動的解析手法と比べ，比較的小さなものである．多くの場合において，提案手法のオーバーヘッドは実用上許容できると考える．

4.3.4　RQ_4 への回答

生成されたシーケンス図において，O_{gtcore} のオブジェク

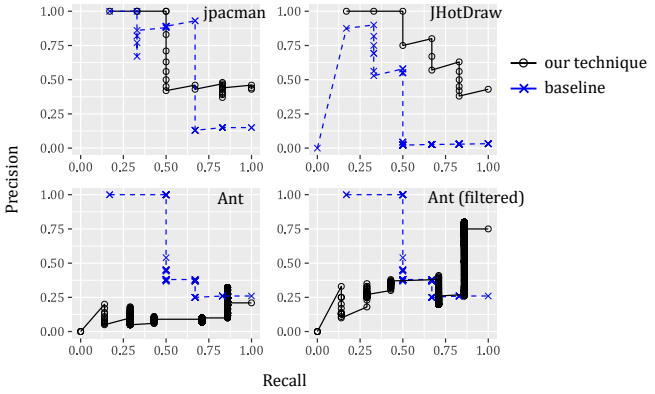

図 3　*Precision-Recall* curve

トの割合が高いほど，設計を理解する上での有用性が高いと考える．*Precision-Recall* の関係を**図 3** に示す．O_{gtcore} は設計を理解する上で特に重要なクラスであるため，実用上は高い *Recall* となることが望ましい．

図 3 から，*Ant* 以外のプロジェクトにおいては，*Recall* > 0.8 となるような高 *Recall* の位置において，提案手法が baseline 手法よりも，精度よく O_{gtcore} を含んだシーケンス図を復元できていることが分かる．

Ant に対しては，図 3 左下グラフから分かるように，全体として提案手法の方が精度が悪い．*Recall* > 0.8 となるような高 *Recall* の位置においては，ほぼ同精度であるが，わずかに提案手法の方が精度が悪い．これは，*Ant* がプロジェクト内に，コレクションデータを保持・イテレートするための独自クラスを定義しており，それらのクラスへのアクセス頻度が高くなることが原因である．コレクションのコンテナとなるようなクラスに対する重要度が不当に高くならないよう，今後，手法の改善が必要である．コレクションコンテナのようなクラスは簡単にフィルタリングすることができ，例えば，*Iterator*・コレクションクラスを継承しているクラスと，*Ant* 内のリソースを扱うコレクション・セレクタのコンテナクラスを除外することで，*Precision* は図 3 右下グラフのように改善する．

jpacman に関して，生成したシーケンス図の一部を**図 4** に示す．図 4 において，O_{gtcore} のオブジェクト間の振る舞いから，次のようなシステムの典型的な振る舞いが読み取れる．*SimpleLevel* オブジェクトは，周期的に *update()* メッセージを *Map*・*Gem*・*Ghost* 等のアクターオブジェクトへ送信する．*Player* オブジェクトと他オブジェクトとの衝突発生時，*onCollision()* メソッドが呼び出され，*Player* オブジェクトの状態が変化する．このことから，例えば，開発者が衝突時の振る舞いを変更（衝突によりプレイヤーの位置を後方にずらす等）しようとした場合に，開発者は変更に必要な知識を図 4 から読み取れることが分かる．

このように，生成されたシーケンス図からは，O_{gtcore} のオブジェクトの相互作用として，システムの典型的な振る舞いを把握できることが確認できた．

高 *Recall* の位置において，提案手法は baseline 手法とほぼ同精度，もしくは，より精度高くシーケンス図を復元できる（*Recall* = 1 の時，*Precision* は 0.21–0.43）．しかし，コレクションを扱うようなクラスが解析対象に多量に含まれていると，ノイズが増え精度が低下する．

復元したシーケンス図からは，O_{gtcore} のオブジェクトの相互作用を確認することで，システムの典型的な振る舞いを把握できる．

5. 妥当性への脅威

実験における各実行シナリオは比較的短いものが多いが，これは不自然な設定ではない．仮に，長い実行トレースしか得られない場合は，興味のある区間（振る舞い理解をしたい区間）のみを，イベントに記録された絶対時間を参考に切り出し，提案手法を適用すればよい．もしくは，2 節で述べた，フェイズ分割手法を利用して実行トレースを分割し，短い実行トレースを得ることもできる．

jpacman に関しては，著者が ground truth (O_{gtcore}) を定めており，誤りが存在する可能性がある．また，本稿では，提案手法により，開発者のプログラム保守工数がどの程度削減されるかまでは明らかにしていない．具体的な保守タスクを設定し，被験者実験を通して手法・評価尺度の有用性・妥当性を評価することが今後必要である．

6. まとめ

実行トレースからシーケンス図を復元し，プログラム理解支援に繋げる手法が有用であるが，復元されたシーケンス図には膨大な情報が含まれており，そのままでは実用面での問題が生じる．本稿では，実行トレース抽象化を目的とした，参照関係・アクセス解析によるコアオブジェクト特定手法を提案した．提案手法では，実行時に大量に生成される一時オブジェクトを特定・除去した後，アクセス頻度に基づいてオブジェクトの重要度を推定する．重要度の高いコアオブジェクトを中心とした振る舞いのみを可視化することにより，実行トレースから復元するシーケンス図の横方向のサイズを大きく削減することができる．

複数のオープンソースソフトウェアに手法を適用することで評価を行った．適用結果から，提案手法は既存手法と比べ，横方向サイズの削減性能に優れており，結果として得られるシーケンス図は，システムの典型的な振る舞い理解に有用であることが確認できた．

今後は，今回利用していない他の静的・動的特徴なども複合し，重要度推定の精度をより高めていくことを考えている．また，手法のより正確・詳細な評価のために，より多くのバリエーションのオープンソースソフトウェアに手法を適用することや，被験者実験が必要である．

謝辞　本研究の一部は科研費 #24300006, #26280021,

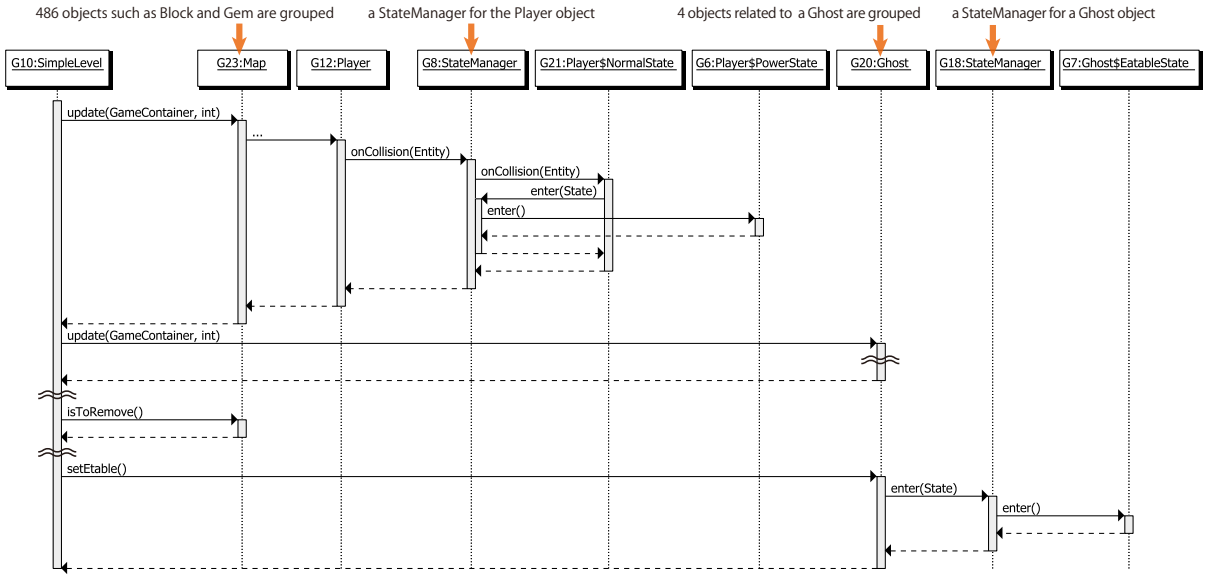

図 4 Part of the resulting diagram in the *jpacman* case

#15H02683, #15K15973 の助成を受けた．

参考文献

[1] Fahland, D., Lo, D. and Maoz, S.: Mining branching-time scenarios, *ASE*, pp. 443–453 (2013).
[2] Lo, D. and Maoz, S.: Scenario-based and Value-based Specification Mining: Better Together, *ASE*, pp. 387–396 (2010).
[3] Bennett, C., Myers, D., Storey, M.-A., German, D. M., Ouellet, D., Salois, M. and Charland, P.: A survey and evaluation of tool features for understanding reverse-engineered sequence diagrams, *J. Softw. Maint. Evol.*, Vol. 20, No. 4, pp. 291–315 (2008).
[4] Taniguchi, K., Ishio, T., Kamiya, T., Kusumoto, S. and Inoue, K.: Extracting Sequence Diagram from Execution Trace of Java Program, *IWPSE*, pp. 148–154 (2005).
[5] Myers, D., Storey, M.-A. and Salois, M.: Utilizing Debug Information to Compact Loops in Large Program Traces, *CSMR*, pp. 41–50 (2010).
[6] Watanabe, Y., Ishio, T. and Inoue, K.: Feature-level Phase Detection for Execution Trace Using Object Cache, *WODA*, pp. 8–14 (2008).
[7] Hamou-Lhadj, A. and Lethbridge, T.: Summarizing the Content of Large Traces to Facilitate the Understanding of the Behaviour of a Software System, *ICPC*, pp. 181–190 (2006).
[8] Srinivasan, M., Yang, J. and Lee, Y.: Case studies of optimized sequence diagram for program comprehension, *ICPC*, pp. 1–4 (2016).
[9] Ishio, T., Watanabe, Y. and Inoue, K.: AMIDA: a Sequence Diagram Extraction Toolkit Supporting Automatic Phase Detection, *ICSE Companion*, pp. 969–970 (2008).
[10] Pirzadeh, H., Shanian, S., Hamou-Lhadj, A., Alawneh, L. and Shafiee, A.: Stratified Sampling of Execution Traces: Execution Phases Serving As Strata, *Sci. Comput. Program.*, Vol. 78, No. 8, pp. 1099–1118 (2013).
[11] Sharp, R. and Rountev, A.: Interactive Exploration of UML Sequence Diagrams, *VISSOFT*, pp. 1–6 (2005).
[12] Cornelissen, B., Zaidman, A. and van Deursen, A.: A Controlled Experiment for Program Comprehension through Trace Visualization, *IEEE Trans. on Softw. Eng.*, Vol. 37, pp. 341–355 (2011).
[13] Myers, D. and Storey, M.-A.: Using Dynamic Analysis to Create Trace-Focused User Interfaces for IDEs, *FSE*, pp. 367–368 (2010).
[14] Grati, H., Sahraoui, H. and Poulin, P.: Extracting Sequence Diagrams from Execution Traces Using Interactive Visualization, *WCRE*, pp. 87–96 (2010).
[15] Dugerdil, P. and Repond, J.: Automatic Generation of Abstract Views for Legacy Software Comprehension, *ISEC*, pp. 23–32 (2010).
[16] Toda, T., Kobayashi, T., Atsumi, N. and Agusa, K.: Grouping Objects for Execution Trace Analysis Based on Design Patterns, *APSEC*, pp. 25–30 (2013).
[17] Noda, K., Kobayashi, T. and Agusa, K.: Execution Trace Abstraction Based on Meta Patterns Usage, *WCRE*, pp. 167–176 (2012).
[18] 戸田達也，小林隆志，渥美紀寿，阿草清滋：オブジェクトの動的特徴に着目した実行トレースの抽象化，信学技報，SS2013-86, Vol. 113, No. 489, pp. 85–90 (2014).
[19] Pirzadeh, H., Hamou-Lhadj, A. and Shah, M.: Exploiting text mining techniques in the analysis of execution traces, *ICSM*, pp. 223–232 (2011).
[20] Noda, K., Kobayashi, T., Yamamoto, S., Saeki, M. and Agusa, K.: Reticella : An Execution Trace Slicing and Visualization Tool Based on a Behavior Model, *IEICE trans. on information and syst.*, Vol. 95, No. 4, pp. 959–969 (2012).
[21] Matsumura, T., Ishio, T., Kashima, Y. and Inoue, K.: Repeatedly-executed-method Viewer for Efficient Visualization of Execution Paths and States in Java, *ICPC*, pp. 253–257 (2014).
[22] Şora, I.: Finding the right needles in hay helping program comprehension of large software systems, *ENASE*, pp. 129–140 (2015).
[23] Zaidman, A. and Demeyer, S.: Automatic Identification of Key Classes in a Software System Using Webmining Techniques, *J. Softw. Maint. Evol.*, Vol. 20, No. 6, pp. 387–417 (2008).
[24] Madsen, M., Tip, F., Andreasen, E., Sen, K. and Møller, A.: Feedback-directed Instrumentation for Deployed JavaScript Applications, *ICSE*, pp. 899–910 (2016).
[25] Huang, J.: Scalable Thread Sharing Analysis, *ICSE*, pp. 1097–1108 (2016).