オブジェクト指向言語のための細粒度システム依存グラフ

蜂巣吉成[†] 山本晋一郎[††] 阿草清滋[†††]

本論文ではオブジェクト指向言語のためのシステム依存グラフOSDGを提案する．OSDGでは解析を式の粒度で行うため，既存の依存解析では困難な関数合成やオブジェクトを介したメソッド呼び出しにおけるデータ依存関係を正確に扱うことができる．一般に細粒度の解析を行うとデータ量が増大するが，本論文では粒度を変換することにより，データ量を削減する手法を提案する．これにより，着目しているクラスは細粒度で詳しく表現し，それ以外のクラスは粗粒度で簡潔に表現することが可能になる．また，OSDGの応用例として，オブジェクトの依存関係を定義し，その表現方法を述べる．

OSDG: Object-oriented System Dependence Graph

YOSHINARI HACHISU,[†] SHINICHIROU YAMAMOTO[††] and KIYOSHI AGUSA[†††]

In this paper, we propose a new control and data dependence graph for an object-oriented language. We call it Object-oriented System Dependence Graph (OSDG). OSDG shows dependence at fine grained level such as expressions, and enables to analyze composite function and method invocation on an object precisely. We also propose an idea translating a fine grained graph into coarse grained one to decrease amount of data. It allows to show a detail graph of a concerning class and to show rough graphs of other classes. To show effectiveness of OSDG, we define *object dependence* relation using OSDG.

1. はじめに

制御依存やデータ依存解析はプログラムの理解支援や最適化に有効であり，従来の手続き型言語に対する解析手法は数多く研究されている[1)~3)]．しかし，オブジェクト指向言語に対する依存解析の研究は少ない[4),5)]．

文献[4)]は手続き型言語における Program Dependency Graph （PDG）を基にした Object-oriented Program Dependency Graph （OPDG）を提案している．OPDG は Class Hierarchy Subgraph, Control Dependence Subgraph, Data Dependence Subgraph （DDS）の3層からなるが，DDSの定義が今後の課題となっているなど不十分である．

文献[5)]は System Dependence Graph （SDG）[1)]をオブジェクト指向言語に適合するように改良し，ステートメントの粒度で解析を行い，メソッド間の解析を可能にしている．しかし，SDGは関数合成などの複雑なメソッド呼び出しが扱えない，メンバ変数の直接参照を考慮していないなどの問題点がある．

本論文ではSDGの問題点を明らかにして，SDGを拡張したOSDG （Object-oriented System Dependence Graph）を提案する．OSDGでは式の粒度で解析を行うため，従来の解析では困難な関数合成やオブジェクトに対するメソッド呼び出し（たとえば，f(g(a+1, b), obj.h(0))），同一変数が複数回出現するステートメント（たとえば，a = f(a) + g(a);）のデータフロー関係などを正確に扱うことができる．一般に細粒度の解析を行うとデータ量が増大するが，本論文では粒度を変換することによりデータ量を削減する手法を提案する．これにより，着目している部分は細粒度で詳しく表現し，それ以外は粗粒度で簡潔に表現することが可能になる．

また，OSDGの応用例としてオブジェクトの依存関係を定義する．一般にオブジェクト指向言語ではオブジェクトを生成し，オブジェクト間の相互作用によって計算を進めるため，オブジェクトの依存関係の把握

† 名古屋大学工学部
School of Engineering, Nagoya University
†† 愛知県立大学情報科学部
Faculty of Information Science and Technology, Aichi Prefectural University
††† 名古屋大学情報メディア教育センター
Center for Information Media Studies, Nagoya University

表1 SDGの節点
Table 1 Node of SDG.

節点名	説明
クラスエントリ節点 (C)	クラスを表す
メソッドエントリ節点 (M)	メソッドを表す
ステートメント節点 (S)	メソッド内のステートメントを表す
呼び出し節点 ($CALL$)	ステートメント中のメソッド呼び出しを表す
引数節点 (P)	メソッド呼び出し時における引数を表す．引数節点はさらに次の4種類に分けられる
formal_in	メソッドの仮引数および，メソッドで使われるメンバ変数
formal_out	メソッドによって変更される可能性のある参照渡しの仮引数とメンバ変数
actual_in	メソッド呼び出しの実引数
actual_out	メソッド呼び出しによって変更される可能性のある実引数
多相選択節点 (PC)	多相メソッド呼び出しを表す

表2 SDGの辺
Table 2 Edge of SDG.

辺名	説明
クラスメンバ辺 ($C \times M$)	クラスとメソッドの所属関係を表す
制御依存辺 ($M \times S$, $S \times S, S \times CALL$)	ステートメントの制御依存関係を表す
データ依存辺 ($S \times S, P \times S, S \times P$)	ステートメントのデータ依存関係を表す
引数データ依存辺 ($P \times P$)	actual_in, actual_out引数節点間のデータ依存関係を表す
メソッド呼び出し辺 ($CALL \times M$, $CALL \times PC, PC \times M$)	ステートメントがメソッドを呼び出すという関係を表す
引数受渡し辺 ($P \times P, S \times CALL$)	メソッド呼び出しにおける実引数と仮引数の対応，およびメンバ変数の対応を表す

は，プログラムの理解や最適化を行うために重要である．しかし，オブジェクトは操作（メソッド呼び出し）により内部の状態（メンバ変数）が変化するため，従来の依存関係でこれを表現するのは難しい．OSDGでは式の粒度で解析を行うため，オブジェクトの依存関係を表現することが可能である．

以下に本論文の構成を示す．2章でSDGの概略を述べ，その問題点を明らかにする．3章でOSDGを定義する．4章で，OSDGの定性的・定量的な評価を行い，OSDGの応用例としてオブジェクトの依存関係を定義する．

2. System Dependence Graph for Object-oriented Software

この章では文献[5]で提案されているSystem Dependence Graph for Object-oriented Software（SDG）を説明し，その問題点を明らかにする．

2.1 基底クラスのグラフ

Class Dependence Graph（ClDG）はSDGの部分グラフである．ClDGは6種類の節点（**表1**）と，6種類の辺（**表2**）から構成される有向グラフである．ClDGの根節点はクラスエントリ節点であり，各クラスのClDGを組み合わせることにより，SDGは作成される．

2.2 派生クラスのグラフ

派生クラスのClDGは，派生クラスで定義されたメソッドと基底クラスのメソッドの部分グラフを組み合わせて作成する．派生クラスのクラスエントリ節点から，派生クラスで定義されたメソッドのメソッドエントリ節点と基底クラスで定義されたメソッドのメソッドエントリ節点に，それぞれクラスメンバ辺が作成される．

2.3 メソッド呼び出し

ステートメントがメソッドを呼び出しているとき，呼び出し節点とメソッドエントリ節点の間にメソッド呼び出し辺を作成する．また，呼び出しステートメント側の実引数を表す引数節点（actual_in）とメソッドエントリ側の仮引数を表す引数節点（formal_in）との間に引数受渡し辺を作成する．同様にactual_outとformal_out間に引数受渡し辺を作成する．

多相呼び出しのメソッドの場合は，多相選択節点を作成して呼び出し節点とメソッド呼び出し辺で結ぶ．さらに多相選択節点と呼び出される可能性のあるすべてのメソッドのメソッドエントリ節点とメソッド呼び出し辺で結ぶ．

また，メソッドが値を返す場合，メソッドのreturn文を表すステートメント節点から呼び出し節点に引数受渡し辺を引く．

SDGの例を**図1**に示す．なお図ではメソッドm1, m3の詳細は省略している．

2.4 SDGの評価

SDGの注目すべき点として

(1) メソッド間の解析を行うための引数の受渡し方法を提案している，
(2) 多相メソッド呼び出しを扱うための方法を定めている，

などがあげられる．

一方，問題点として次の点があげられる．

(1) 関数合成などの複雑なメソッド呼び出し（e.g. f(g(a+1, b), obj.h(0))）を考慮してない．
(2) 他のクラスのメンバ変数の参照を考慮してい

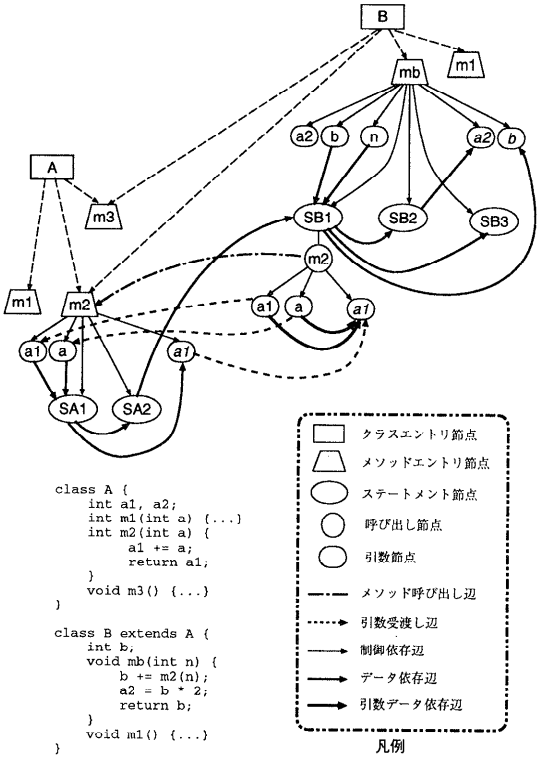


図1 SDG の例
Fig. 1 Example of SDG.

ない.

(3) クラス・メソッドの情報が局所化されないためグラフの再利用性が悪い. つまり, 基底クラスでメソッドを追加すると派生クラスの CIDG も変更する必要があり, また, メソッドの仮引数のデータ依存関係を変更するとメソッド呼び出し側の引数データ依存辺を変更しなければならない.

3. Object-oriented System Dependence Graph

本章では 2 章であげた SDG の問題点を解決するために, SDG を拡張した OSDG (Object-oriented System Dependence Graph) を提案する. OSDG は構文木を基に制御依存関係やデータ依存関係を表現する.

3.1〜3.4 節では OSDG のクラス間の関係, 制御依存関係, データ依存関係, メソッド呼び出し関係に対して直観的な説明を行い, 3.5 節で OSDG の定義を与える. OSDG では依存関係を式の粒度で扱うため必要なデータが増大するが, 3.6 節で OSDG のグラフの粒度を変換し, データを削減する方法について述

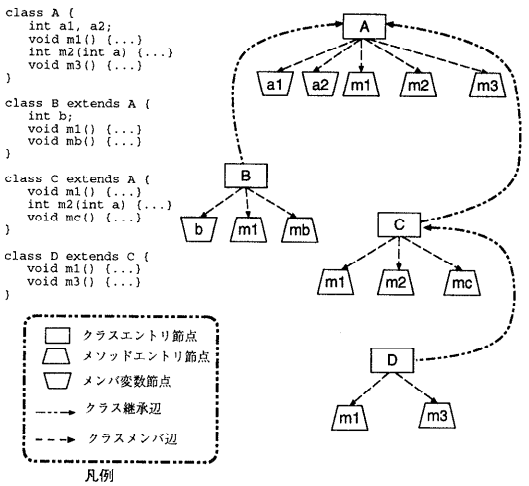


図2 クラス間の関係グラフ
Fig. 2 Graph bewteen classes.

べる.

なお, この章では Java から例外処理 (throw, try, catch など) と同期 (synchronized) に関する部分を除いた言語を対象とする. この対象言語にはオブジェクト指向言語の特徴であるカプセル化と継承, 多相性が含まれているため, ここでの議論は一般のオブジェクト指向言語にも適用することができる☆. メソッド呼び出しの観点からこれらの特徴をみると, カプセル化はオブジェクトに対するメソッド呼び出しとして, 多相性は動的なメソッド呼び出しとして表現される. また, オブジェクトは一般に参照として扱われるので, 本論文では参照渡しによるメソッド呼び出しも扱う.

3.1 クラス間の関係

OSDG では派生クラスで基底クラスのメソッドを保持せずに, 派生クラスのクラスエントリ節点から基底クラスのクラスエントリ節点に継承関係を表す辺を作成する. このため, SDG に比べクラスメンバ辺の数が少なくなる. また, 基底クラスにメソッドを追加した場合に, 派生クラスのクラスメンバ辺を追加する必要がなく, 再利用性に優れる.

また, 他クラスからのメンバ変数参照に対応するためメンバ変数節点を作成し, クラスエントリ節点からメンバ変数節点にクラスメンバ辺を作成する (図 2).

3.2 制御依存

SDG の制御依存の表現と同じである.

3.3 データ依存

SDG を含めた従来の依存グラフでは, f(g(0)) のようにメソッドの返り値を引数としたメソッドの呼び出

☆ SDG の対象言語 (C++) も同じ特徴を持つ.

しを扱えない．また，

S1: a = 1;
S2: a = f(a) + g(a);
S3: b = a;

という部分プログラムがあった場合，従来のデータ依存グラフは変数 a に対して，(S1, S2)，(S2, S3) というデータ依存辺が引かれるが，S2 には a が 3 回出現するため，S1 の a の定義が S2 のどの a で使用されているか機械的に判断できない☆．

これらの問題は本来，式の粒度で扱うべきデータ依存関係をステートメントの粒度で扱っていることが原因である．OSDG ではデータ依存を式の粒度で扱う．

3.3.1 式とデータ依存

まず，式におけるデータ依存関係に必要な定義を示す．

定義 1（末端式） 式を構文木で表したとき葉となる式を末端式と呼ぶ． □

定義 2（変数参照式） 式 E が変数を定義または使用しており，かつ式 E が末端式のとき，式 E を変数参照式と呼ぶ． □

式におけるデータフロー関係，データ依存関係を以下のように定義する．

定義 3（式におけるデータフロー関係） 次の条件を満たすとき，変数 x について変数参照式 E1，E2 にデータフロー関係があるという．

(1) 式 E1 が変数 x を定義している．
(2) E1 を含むステートメントを S1 とすると，S1 において E1 を評価した後で変数 x を定義することがない．
(3) E2 が変数 x を使用している．
(4) E2 を含むステートメントを S2 とすると，S2 において E2 を評価する前に変数 x を定義することがない．
(5) S1 から S2 への制御フローがあり，S1 から S2 の間に変数 x を定義するステートメントが存在しない．

□

定義 4（式におけるデータ依存関係） 次の条件を満たすとき，変数 x について変数参照式 E1，E2 にデータ依存関係があるという．

(1) 式 E1，E2 がデータフロー関係にある，または
(2) 式 E1，式 E2 が代入式を構成し，E1 が右辺，E2 が左辺である．

また，E1，E2 のデータ依存関係は E1 $\xrightarrow{d}$ E2 と表現する． □

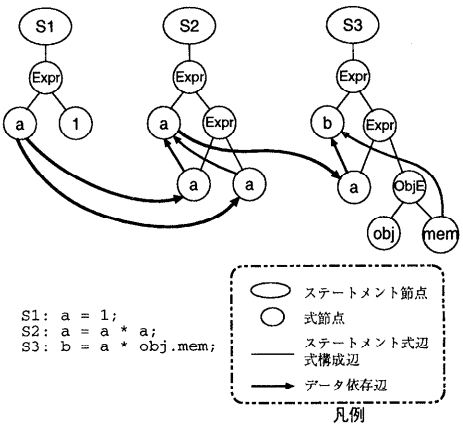


図 3 式におけるデータ依存グラフ
Fig. 3 Data dependence graph at the expression level.

式におけるデータ依存グラフの例を**図 3** に示す．図で "ObjE" と標識づけられた式節点は obj.mem などのオブジェクトに対する操作を表す式を，"Expr" はその他の末端式でない式を表す．

3.4 メソッド呼び出し

OSDG ではメソッド呼び出しを式節点で表現する．式の粒度で解析を行うため，f(g(a+1, b), obj.h(0)) などの関数合成やオブジェクトに対するメソッド呼び出しも正確に扱うことができる．

3.4.1 引数節点

SDG ではメンバ変数と仮引数の節点を同等に扱っていた．OSDG ではこれを分類する．

OSDG では引数節点を次の 8 種類に分類する．

(1) formal_in メソッドの仮引数
(2) member_in メソッドが参照するメンバ変数
(3) formal_out メソッドが値を変更する参照渡しの仮引数
(4) member_out メソッドが値を変更するメンバ変数
(5) return メソッドが返す値
(6) actual_in メソッドの実引数
(7) actual_out メソッドによって変更される参照渡しの実引数
(8) returned メソッドによって返される値

つまり，SDG の formal_in，formal_out をメンバ変数と仮引数でさらに分類し，新たに return，returned の節点を加える．メソッド定義側では return 節点で値を返し，メソッド呼び出し側では returned 節点で

☆ 関数 f，g とも引数が値渡しの場合，f(a)，g(a) どちらの a も S1 の a の定義に依存する．しかし，関数 f が参照渡しで，かつ演算子 "+" が左結合の場合，f(a) の a は S1 の a の定義に依存するが，g(a) の a は f(a) に依存する

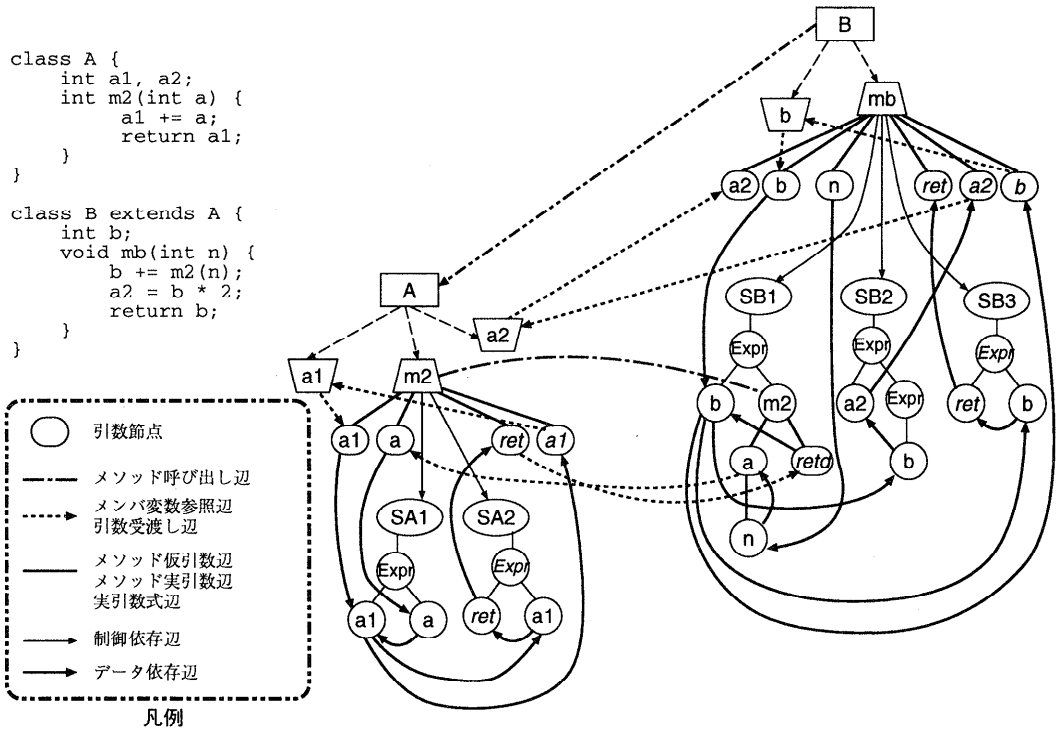


図4 メソッド呼び出しのグラフ1
Fig. 4 Graph for method invocation 1.

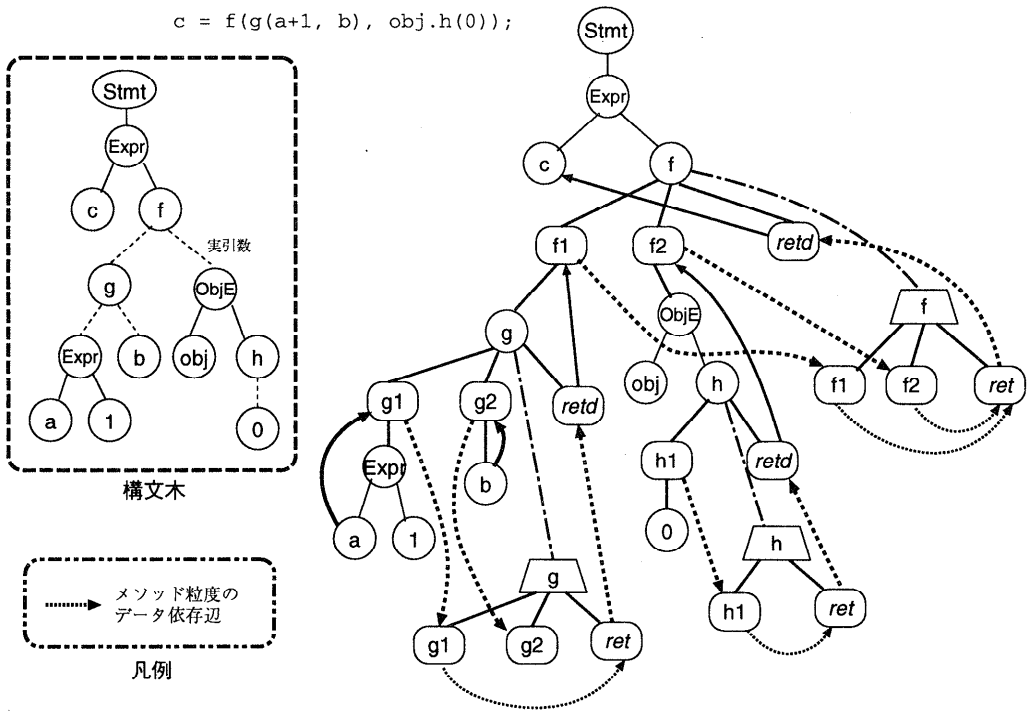


図5 メソッド呼び出しのグラフ2
Fig. 5 Graph for method invocation 2.

値を受け取る．なお，return 文 “**return EXPR;**” は，返り値を表す仮想的な変数 *ret* を用いて “**return (***ret* **= EXPR)**” 文として扱うこととする．

member_in，member_out は同じクラスのメンバ変数だけでなく，他のクラスのメンバ変数を参照している場合にも用いる．member_in，member_out 節点からそれが参照するメンバ変数節点にはメンバ変数参照辺が作成される．

3.4.2 メソッド呼び出しのグラフ

メソッド呼び出しは次のようにして構文木から OSDG に変換する．

(1) 呼び出すメソッド本体の formal_in，formal_out，return に対応する actual_in，actual_out，returned 節点を作成する．

(2) メソッド呼び出しを表す式節点と (1) で作成した actual_in，acutal_out，returned とを辺で結ぶ．この辺をメソッド実引数辺と呼ぶ．

(3) actual_in，actual_out とメソッドの実引数を表す式節点とを辺で結ぶ．この辺を実引数式辺と呼ぶ．

(4) メソッド呼び出しを表す式節点とメソッド本体のエントリ節点を辺で結ぶ．この辺をメソッド呼び出し辺と呼ぶ．

(5) メソッド呼び出しの actual_in，actual_out，returned 節点と，それに対応するメソッド本体の formal_in，formal_out，return 節点とを辺で結ぶ．この辺を引数受渡し辺と呼ぶ．

なお，引数受渡し辺は actual_in $\xrightarrow{d}$ formal_in，formal_out $\xrightarrow{d}$ actual_out，return $\xrightarrow{d}$ returned のデータ依存関係も表現する．

静的メソッド呼び出しの場合の OSDG の例を**図 4**，**図 5** に示す．たとえば図 4 のメソッド mb の返り値を計算するのに必要な変数は，OSDG のデータ依存辺，引数受渡し辺の到達可能性を調べることにより，メソッド mb の引数 n，クラス A のメンバ変数 a1，クラス B のメンバ変数 b であることが分かる．

図 5 ではメソッド呼び出しの返り値を別のメソッド呼び出しの引数に使用している．図 5 の仮引数節点間の辺は仮引数間のデータ依存関係を表す（3.6.2 項参照）．

また多相メソッド呼び出しは SDG と同じく多相選択節点を用いて表現する（**図 6**）．

SDG では実引数データ依存辺を用いて実引数間のデータ依存関係を表現している．しかし，この手法ではメソッドの引数のデータ依存関係が変更された場合，すべての呼び出し側の実引数データ依存辺も変更しなければならない．

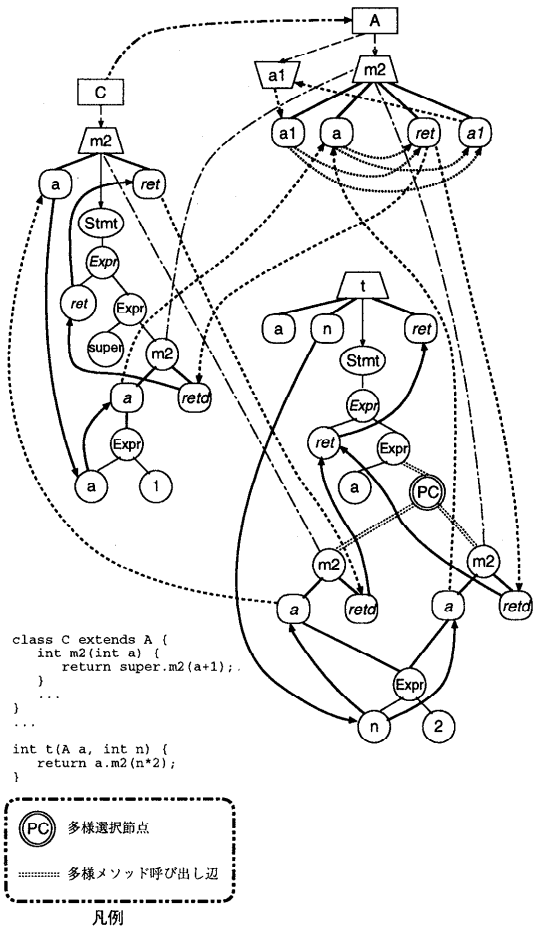


図 6 多相メソッド呼び出しのグラフ
Fig. 6 Graph for polymorphic method invocation.

OSDG では変数参照式のデータ依存辺や引数受渡し辺の到達可能性を調べることにより，実引数として渡される式のデータ依存関係を求めることができるため，SDG の実引数データ依存辺を削除した．なお，OSDG では 3.6.2 項のメソッド粒度のグラフにより，仮引数間のデータ依存関係を表すことができる．

3.5 OSDG のグラフ定義

以上のことを整理して OSDG のグラフを定義する．式の構造や演算子の優先順位を扱うために OSDG では構文木を基にグラフを構成する．

OSDG の節点は構文木の節点 5 種類とそれ以外の 3 種類に分類される（**表 3**）．なお，仮引数節点と実引数節点をあわせて引数節点（P）と呼ぶ．

OSDG の辺は構文木の辺 4 個とそれ以外の 9 個に分類される（**表 4**）．

表3 OSDGの節点
Table 3 Node of OSDG.

構文木に基づく節点	
節点名	説明
クラスエントリ節点（C）	クラスを表す
メソッドエントリ節点（M）	メソッドを表す
メンバ変数節点（V）	メンバ変数を表す
ステートメント節点（S）	ステートメントを表す
式節点（E）	式を表す．SDGの呼び出し節点はOSDGでは式節点となる．なお，メソッド呼び出しを表す式節点を特に $CALL$ と表記することがある
それ以外の節点	
節点名	説明
多相選択節点（PC）	多相性によって呼び出されるメソッドが一意に決まらないことを表す
仮引数節点（P_f）	メソッドの仮引数を表す formal_in，formal_out，メンバ参照を表す member_in，member_out，返り値を表す return で構成される
実引数節点（P_a）	メソッド呼び出しの実引数を表す actual_in，actual_out，メソッドから返される値 returned で構成される

表4 OSDGの辺
Table 4 Edge of OSDG.

構文木に基づく辺	
辺名	説明
クラス継承辺（$C \times C$）	クラス間の継承関係を表す
クラスメンバ辺（$C \times M, C \times V$）	クラスとメソッド，メンバ変数の所属関係を表す
ステートメント式辺（$S \times E$）	ステートメントと式の構成関係を表す
式構成辺（$E \times E$）	式の構成関係を表す
それ以外の辺	
辺名	説明
メソッド仮引数辺（$M \times P_f$）	メソッドとメソッドの仮引数節点の関係を表す
メンバ変数参照辺（$P_f \times V$）	メンバ変数参照を表す仮引数節点（member_in，member_out）とメンバ変数の関係を表す
メソッド実引数辺（$CALL \times P_a$）	メソッド呼び出し式と実引数節点との関係を表す
実引数式辺（$P_a \times E$）	実引数節点と引数として渡される式の対応を表す
引数受渡し辺（$P \times P$）	仮引数節点（formal_in，formal_out，return）と実引数節点（actual_in, actual_out, returned）の対応を表す
メソッド呼び出し辺（$CALL \times M$）	メソッド呼び出しにおける，メソッド呼び出し式とメソッドエントリ節点の関係を表す
多相メソッド呼び出し辺（$E \times PC, PC \times CALL$）	多相メソッド呼び出しにおける，メソッド呼び出し式と多相選択節点の関係を表す
制御依存辺（$M \times S, S \times S$）	制御依存関係を表す
データ依存辺（$E \times E, P \times E, E \times P$）	データ依存関係を表す

3.6 OSDGの粒度変換

OSDGでは構文を式の粒度で表現しているため，従来より正確にデータ依存関係を表現できるが，必要なデータ量が増大する．また一般にはステートメントレベルの情報で十分な場合や，メソッド内部をブラックボックス化したい場合も多い．

本節では，OSDGのグラフの粒度をステートメント粒度，メソッド粒度に変換する方法を示す．本手法により，着目している部分を式の粒度で詳しく表現し，それ以外はステートメントやメソッドの粒度で簡潔に表現することが可能になる．

3.6.1 ステートメント粒度

次のようにして式の粒度のグラフをステートメントの粒度のグラフに変換できる．このグラフはステートメント間のデータ依存関係やステートメントが呼び出すメソッド，メソッド間のデータの受渡しなどを表現しており，従来のSDGと同等の表現力を持つ．

(1) 変数参照式eについて
  (a) eあるいはeを含む式がメソッド呼び出しの実引数のとき，eに対するデータ依存辺の始点または終点を実引数節点にする．
  (b) それ以外のとき，eに対するデータ依存辺の始点または終点をeを含むステートメントに変える．

(2) (1)の結果，同じ始点，終点を持つ辺が複数ある場合は，1つの辺だけを残してその他の辺を削除する．

(3) メソッド呼び出し式とそれを含むステートメントの間に辺を引く．この辺はステートメント内で呼び出されるメソッドを表す．

(4) 次の節点と辺を削除する．
  - メソッド呼び出し式を除く式節点
  - ステートメント式辺，式構成辺，実引数式辺，始点または終点が式節点のデータ依存辺

3.6.2 メソッド粒度

次のようにして式の粒度のグラフをメソッド粒度のグラフに変換できる．これはメソッド内部をブラックボックス化し，仮引数間のデータ依存関係を表したものである．

(1) member_in，formal_inの仮引数節点inに対し

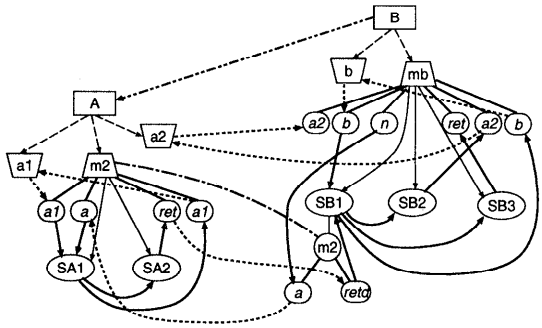


図7 ステートメント粒度のグラフ
Fig. 7 Graph at the statement level.

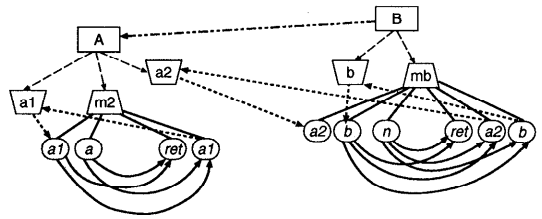


図8 メソッド粒度のグラフ
Fig. 8 Graph at the method level.

て，データ依存辺，または引数受渡し辺を用いて到達可能な仮引数節点 out があるとき，in と out を辺で結ぶ．この辺は仮引数間のデータ依存関係を表す．

(2) 次の節点と辺以外の節点・辺を削除する．
- クラスエントリ節点，メソッドエントリ節点，メンバ変数節点，仮引数節点
- クラス継承辺，クラスメンバ辺，メソッド仮引数辺，メンバ変数参照辺，(1) で作成した仮引数データ依存辺

図4をステートメント粒度にしたグラフを**図7**に，メソッド粒度にしたグラフを**図8**に示す．また，図5，図6は式の粒度とメソッドの粒度のグラフを合わせたグラフになっている．

4. 評　価

この章では OSDG の評価として，SDG との比較，OSDG の空間的・時間的効率と OSDG の応用例としてオブジェクトの依存関係について述べる．

4.1 SDG との比較

2.4 節であげた SDG の問題点は OSDG では次のように解決された．

(1) OSDG では式の粒度で解析を行うため，複雑なメソッド呼び出しや，同一変数が複数回出現するステートメントのデータフロー関係を表現できる（3.3 節参照）．

(2) 他のクラスのメンバ変数の直接参照は，メンバ変数節点と member_in，member_out 節点間のメンバ変数参照辺で扱うことができる（3.1 節参照）．

(3) 情報の局所化，および再利用性を高めるためにクラス継承辺を新たに作成した．このため，基底クラスにメソッドを追加したときに，派生クラスのグラフを変更する必要がない（3.1 節参照）．また，SDG の引数データ依存辺を削除し，メソッド粒度のグラフで仮引数間のデータ依存関係を表現した．このため，メソッドの引数のデータ依存関係が変更されても，メソッドを呼び出した側のグラフを変更する必要がない（3.4.2 項参照）．

4.2 空間的・時間的効率

細粒度リポジトリに基づいた CASE ツールプラットフォーム Japid[6] を利用して，OSDG を作成するツールを試作し，その空間的・時間的な評価を行った．

Japid では Java 言語の構文を Class，Routine☆，Expression，Operator など 15 の実体と 45 の関連でモデル化している．Japid はソースプログラムを構文解析した結果をデータベース化し，CASE ツール作成を支援する．Japid ではソースプログラムを細粒度，すなわち式や演算子の粒度で解析しているため，OSDG の作成に適している．

ここでは OSDG のうちメソッド間の関係，つまり多相選択節点，メソッド呼び出し辺，多相メソッド呼び出し辺，引数受渡し辺などを除いた部分を作成し，その空間的・時間的効率を調べた．対象プログラムは JDK1.1.6 の String クラス（1531 行，13 コンストラクタ，48 メソッド）を選び，Sun Ultra1 OEM ワークステーション（300 MHz UltraSPARC-II, 128 MB RAM）で実行した．

まず，Japid により 4245 個のオブジェクト（節点）と 6128 個の関連（辺）が生成されたことを確認した．この中には Operator などの OSDG には直接関係のない節点や辺も含まれている．次に Japid を利用して OSDG を作成し，データベース☆☆に保存した．ステートメント，メソッド粒度の辺も含めて全体で 4936 個のオブジェクトと 8338 個の関連が生成され，6.7 MB ほどメモリを消費した．また生成には 16.9 秒（データベース読み込み時間 9.6 秒を含む）要した．最後に

☆ Routine は Method と Constructor を汎化したものを表す．
☆☆ 現在は単なるファイルとして管理されている．

表5 OSDGの構築に必要な時間・メモリ
Table 5 Time and memory for constructing OSDG.

構築したグラフ	時間（秒）	メモリ（MB）	節点	辺
式粒度	7.1	5.2	4018	2810
ステートメント粒度	4.6	4.2	2448	2054
メソッド粒度	3.3	2.9	1684	918
初期データベース読み込み	9.6	4.3	4245	6128
OSDGの作成と保存	16.9	6.7	4936	8338

保存したデータベースから各粒度のグラフを構成し，実行時間，使用メモリ，節点と辺の数を計測した．結果を**表5**に示す．この結果から細粒度ながら十分実用的な性能であること，粒度の変換によりグラフの構築にかかる時間が短縮され，必要なメモリを削減できることが分かる．

4.3 オブジェクトの依存関係

一般にオブジェクト指向言語ではオブジェクト間の相互作用によって計算を進めるため，オブジェクトの依存関係を把握することは重要である．しかし，オブジェクトは操作（メソッド呼び出し）により内部の状態（メンバ変数）が変化するため，従来の依存関係では表現できない．

ここでは以下のようにオブジェクトの依存関係を定義する．オブジェクトが生成された時点から，オブジェクトの依存関係に従ってフォワード・スライシングを行うことにより，オブジェクトの生存期間やオブジェクトに対する操作の系列が得られる．これを利用してプログラムの理解支援や最適化などを行うことができる．

定義5（オブジェクト定義） オブジェクトのメンバ変数の少なくとも1つが定義されたとき，オブジェクトが定義されるという． □

定義6（オブジェクト使用） オブジェクトのメンバ変数の少なくとも1つが使用されたとき，オブジェクトが使用されるという． □

定義7（オブジェクトフロー関係） オブジェクトxについてオブジェクト参照式O1，O2が次の条件を満たすとき，O1とO2にデータフロー関係があるという．なお，オブジェクト参照式とはオブジェクトを定義または使用している末端式のことをいう．

(1) 式O1がオブジェクトを定義している．
(2) O1を含むステートメントをS1とすると，S1においてO1を評価した後でオブジェクトxを定義することがない．
(3) O2がオブジェクトxを使用している．
(4) O2を含むステートメントをS2とすると，S2においてO2を評価する前にオブジェクトxを定義することがない．
(5) S1からS2への制御フローがあり，S1からS2の間にオブジェクトxを定義するステートメントが存在しない． □

定義8（オブジェクト依存関係） オブジェクト参照式O1，O2がオブジェクトフロー関係にあるとき，O1，O2はオブジェクト依存関係にあるという． □

なお，オブジェクトの定義・使用はメンバ変数への直接参照とメソッド呼び出しによって参照される間接参照に分類される．直接参照による定義・使用はメンバ変数の定義・使用を調べることで簡単に求めることができる．間接参照の場合は，メソッド呼び出しによるメンバ変数への副作用を調べなければならない．OSDGではオブジェクトに対するメソッド呼び出しを式の粒度で扱っており，メソッドによるメンバ変数の定義・使用をmember_in，member_out節点で表現しているため，間接参照によるオブジェクトの定義・使用を次のようにして求めることができる．

定義9（OSDGを用いたオブジェクト定義・使用） オブジェクトに対するメソッド呼び出し式O.m()について，

(1) メソッドmのメソッドエントリ節点とmember_out節点間に仮引数辺が存在するとき，オフジェクト参照式Oはオブジェクト定義である
(2) メソッドmのメソッドエントリ節点とmember_out節点間に仮引数辺が存在せず，メソッドエントリ節点とmember_in節点間に仮引数辺が存在するとき，オブジェクト参照式Oはオブジェクト使用である □

たとえば，図5ではobj.h(0)のオブジェクト参照式objはメソッドhにmember_in節点h1が存在するが，member_out節点は存在しないため，オブジェクト使用である．

5. おわりに

本論文ではSDGを拡張したOSDGとグラフの粒度変換の手法を提案した．OSDGは式の粒度で解析を行うため，従来の依存グラフに比べ正確な解析を行うことができる．注目度に応じて異なる粒度のグラフを組み合わせることができ，理解しやすい簡潔なグラフを作成することができる．また，実際にOSDGを作成するツールを試作し，細粒度ながら実用的な性能が得られること，粒度選択によりグラフ作成に必要な時間やデータ量が削減できることを示した．さらにオブジェクトの依存関係を定義し，OSDGで表現する方法を示した．

今後の課題として，OSDG作成ツールの完成，スライシングを行うツールの作成などがあげられる．

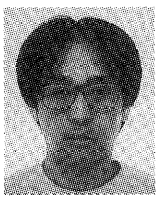
メソッド呼び出しを含んだOSDGを作成する際には，多相メソッド呼び出しの数を減らすことで，より正確な解析結果が得られる．与えられたプログラムを解析して多相メソッド呼び出しを静的な呼び出しに変換する手法としてRapid Type Analysisなどが知られている[7]．これらの手法と組み合わせることで，OSDGの多相選択節点を減らすことができる．

スライシングについては本論文では触れなかったが，ステートメント粒度のOSDGに対して，文献[5]で述べられているSDGのスライシング・アルゴリズムを用いることができる．また，オブジェクトの依存関係を利用してオブジェクトのフォワード・スライシングを行うツールを作成し，実際にそれを理解支援や最適化に利用したいと考えている．

参考文献

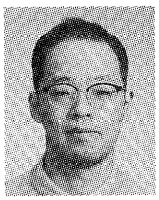
1) Horwitz, S., Reps, T. and Binkley., D.W.: Interprocedural Slicing Using Dependence Graphs, *ACM Trans. Programming Languages and Systems*, Vol.12, No.1, pp.26–60 (1990).
2) Ferrante, J., Ottenstein, K. and Warren, J.: The Program Dependence Graph and Its Use In Optimization, *ACM Trans. Programming Languages and Systems*, Vol.9, No.3, pp.319–349 (1987).
3) Kennedy, K.: A Survey of Data Flow Analysis Techniques, *Program Flow Analysis: Theory and Applications*, Jones, N.D. and Muchnick, S.S. (Eds.), pp.5–54, Prentice-Hall (1981).
4) Krishnaswamy, A.: Program Slicing: An Application of Object-oriented Program Dependency Graphs, Technical Report, TR94-108, Dept. of Computer Science, Clemson Univ. (1994).
5) Larsen, L. D. and Harrold, M.: Slicing Object-Oriented Software, *Proc. 18th International Conference on Software Engineering* (1996).
6) 蜂巣吉成，山本晋一郎，濱口　毅，阿草清滋：Java言語のための細粒度リポジトリ，コンピュータシステム・シンポジウム論文集，pp.147–154 (1996).
7) Bacon, D.F. and Sweeney, P.F.: Fast Static Analysis of C++ Virtual Function Calls, *Proc. ACM 1996 Conference on Object-Oriented Programming Systems, Languages, and Applications*, pp.324–341 (1996).

(平成10年9月9日受付)
(平成11年1月8日採録)



蜂巣　吉成

1971年生．1994年名古屋大学工学部情報工学科卒業．1996年同大学院工学研究科情報工学専攻博士前期課程修了．現在，同大学院博士後期課程在学中．項書換え計算，ソフトウェア開発環境に関する研究に興味を持つ．



山本晋一郎（正会員）

1962年生．1987年名古屋大学工学部卒業後，同大学院に進学．1991年同大学助手，1996年講師．1998年愛知県立大学情報科学部助教授．プログラミング言語処理系，ソフトウェアの形式的開発手法，ソフトウェア開発環境に関する研究に従事．最近は，細粒度のソフトウェア・リポジトリに基づいたCASEツール・プラットフォームに関する研究を進めている．電子情報通信学会，日本ソフトウェア科学会各会員．

阿草　清滋（正会員）

1947年生．1970年京都大学工学部電気工学第二学科卒業．1972年同大学院工学研究科電気工学第二専攻修士課程修了．同博士課程へ進学．1974年より同情報工学科助手．同講師，助教授を経て1989年より名古屋大学教授．現在，同情報メディア教育センター教授．工学博士．専門分野はソフトウェア工学，ソフトウェア開発方法論，知的開発環境，ソフトウェアデータベース，仕様化技法，再利用技法，マンマシンインタフェース．電子情報通信学会，ソフトウェア科学会，IEEE，ACM各会員．