Vol.2011-SE-171 No.11
2011/3/14

サンプルプログラム自動生成のためのリソースを考慮した依存グラフ

藤浦祥雅[†1] 大久保弘崇[†2]
粕谷英人[†2] 山本晋一郎[†2]

本稿ではリソースを多用するプラットフォームアプリケーションのサンプルプログラム自動生成手法を提案する．既存のプログラム依存グラフを拡張し，本研究で定義するリソースの依存グラフと結合することにより，リソースを含むプログラムを依存グラフとして表現する．また既存技術のプログラムスライシングを発展させた，本依存グラフのスライシング手法も合わせて提案する．リソースを考慮した依存グラフとスライシングの応用として，サンプルプログラム自動生成ツールを実装した．本ツールはソースプログラムを入力し，キーワード (クラス名や関数名) 選択すると，キーワードのサンプルプログラムとしてキーワードに関連のあるコードを出力する．本稿はケーススタディとして携帯電話プラットフォーム Google Android に対して適用した．

Dependence Graph Considerering Resource for Automatic Sample Program Generation

Yoshimasa Fujiura,[†1] Hirotaka Ohkubo,[†1]
Hideto Kasuya[†1] and Shinichiro Yamamoto[†1]

This paper propose an automatic generation of sample programs based on platform applications with resources. We extend the program dependence graph by adding resource dependency defined in this paper. Slicing of our Dependence Graph, which is an extension of the program slicing, is also addressed. As an application of the Dependence Graph and Slicing, we have implemented Sample Program Generator. By inputting a source program and selecting a keyword such as class-name, the tool generates sample codes corresponding to the keyword. We present a case study using Google Android platoform and indicate the efficiency of the Tool.

1. はじめに

今日のソフトウェア開発において，開発効率化のために特定分野の基本的な機能を提供するプラットフォームを利用することは必須となっている．プラットフォームを利用したソフトウェア開発では，リソースを多用する傾向がある．ここでいうリソースとは，画像や文字列などプログラム内から参照される静的データのことを指す．リソースの多用化により静的データの記述場所が明確になったが，一方でプログラム内の間接参照が増加し，コード間の関係が複雑化した．そのためプログラムの保守や改変の際に，注目するコードに関連するコード及びリソースを把握することが困難となる．

プログラム理解を目的とした既存研究として，プログラム依存グラフ[9),12)] とプログラムスライシング[11)] がある．プログラム依存グラフはコードをデータ依存関係と制御依存関係を用いて関連付けたグラフである．プログラムスライシングはプログラム依存グラフを用いて，注目するコード (スライス基準) に関連するコード群 (スライス) を探索する手法である．この二つの既存研究は，ソフトウェア工学の幅広い分野で適用され，発展研究が多くなされてきた．しかしリソースを利用するソフトウェアに関するこれらの研究なされていない．

本研究は既存のプログラム依存グラフを拡張し，リソースを考慮した依存グラフ Source and Resource Dependence Graph (SRDG) と SRDG のスライシング手法 (SRDG スライシング) の提案する．SRDG は本研究で定義するリソースの依存グラフと既存のプログラム依存グラフを結合した依存グラフである．SRDG スライシングは，既存スライシング技術を用い，SRDG 用に拡張したプログラムスライシング手法である．リソースを含むプログラムに対して本手法と既存手法の適用を比較すると，プログラムとして記述されていた部分がリソースとして記述できるようになってきた背景から，多くの部分がリソースとして記述されるプログラムに適したものといえる．

また本手法の応用として，ソースプログラムを入力し，キーワードを指定することで，キーワードに関連した部分だけのサンプルプログラムを自動で生成するツールを実装し，携帯電話プラットフォーム Google Android[3)](以降 Android) へ適用した．

†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

© 2011 Information Processing Society of Japan

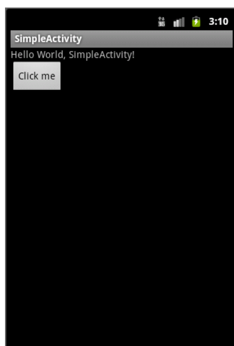

図 1　Andorid アプリケーションの画面

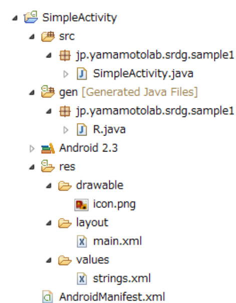

図 2　SimpleActivity アプリケーションのディレクトリ構成 (Andorid)

図 3　画面構造のリソース (main.xml)

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <LinearLayout
3      xmlns:android="http://schemas.android.com/apk/res/android"
4      android:orientation="vertical"
5      android:layout_width="fill_parent"
6      android:layout_height="fill_parent"
7      >
8          <TextView
9              android:layout_width="fill_parent"
10             android:layout_height="wrap_content"
11             android:text="@string/hello"
12             android:id="@+id/text"
13             />
14         <Button
15                 android:layout_width="wrap_content"
16                 android:layout_height="wrap_content"
17                 android:text="@string/button"
18                 android:id="@+id/button"
19                 />
20 </LinearLayout>
```

2.　プラットフォームにおけるリソース

プラットフォームは特定分野のソフトウェア開発効率化を目的としたソフトウェアであり，特定分野における利用頻度の高い機能を基本的な機能として提供する．そのためプラットフォーム上で動作するアプリケーションは，固有の機能を実装するだけでよいため，効率的にアプリケーションを開発できる．近年のプラットフォームの傾向として，画面構成などのユーザインタフェースをリソースとして扱えるように開発規則を定めていることが多く，リソースの担う役割が大きいことが挙げられる．ここでいうリソースとはプログラム内で扱う静的データのことであり，XML などの構造化言語や CVS 形式などで記述される．

リソースを利用することで静的データとソースコードを分離することができる．たとえば表示される文字列をリソースとしてプログラムから参照することで，実行時の言語環境に合わせた文章を表示するようできる．さまざまな形式のデータがリソースとして必要になり，プラットフォームにおける開発では多くのリソースファイルを扱うようになってきている．

Android におけるリソースの利用例

プラットフォームを用いたアプリケーション開発の事例として，Android におけるリソースの利用を示す．Android ではプログラム言語に Java，リソース記述言語に XML を用いる．リソースとして記述できる内容には，画面構成，文字列，色，スタイルなどがある．図 1 の画面はリスト 3 の XML 記述から作られる．

図 1 の画面はテキストを表示する部分 (Hello World, SimpleActivity!) とボタン (Click me) で構成され，これらを一方向に並べるようレイアウトされている．これを XML で記述すると，テキストを表示する部分は TextView 要素 (リスト 3，*l*.8–*l*.13)，ボタンは Button 要素 (リスト 3，*l*.14–*l*.19) で記述される．そして，一方向に並べるレイアウトである LinearLayout 要素 (リスト 3，*l*.2–*l*.7, *l*.20) に 2 つの要素が入れ子になっている．このように，XML を用いて画面構造を階層的に記述することができる．

次にリソース間の参照について説明する．リスト 3 には図 1 に表示されている文字列は含まれていない．それらは別のリソースにあり，strings.xml という文字列を格納するリソースファイルの文字列を参照している．strings.xml には，識別子付き文字列が格納されている．ここで識別子は string 要素の属性 name の値である．リソース間の参照には特別な構文が用いられる場合が多い．Andorid において，リソースからリソースへの参照は “@*resourece*/*id*” という構文を用いる．リスト 3 の 11 行目は，属性 name の値が hello である string 要素の文字列を参照しており，TextView に「Hello World, SimpleActivity!」と表示される．

プログラムからリソースを参照することもできる．リスト 4 にリソースを参照する Java

© 2011 Information Processing Society of Japan

図 4 Java プログラム (SimpleActivity.java)

```
1  public class SimpleActivity extends Activity {
2      TextView text;
3
4      public void onCreate(Bundle savedInstanceState) {
5          super.onCreate(savedInstanceState);
6          setContentView(R.layout.main);
7
8          Button button = (Button) findViewById(R.id.button);
9          button.setOnClickListener(mListener);
10         text = (TextView) findViewById(R.id.text);
11     }
12 }
```

プログラムの例を示す.

リスト 4 の 6 行目で，リスト 3 の内容を参照し表示画面として設定している．リスト 3 を指す部分は “R.layout.main” で，これもリソース間の参照同様，プラットフォームで定められた特殊な構文を用いて参照される．プログラムからリソースを参照する場合には，“`R.`*resourece*`.`*id*” という構文を用いる．この場合，layout ディレクトリに含まれる main.xml を参照している．図 2 に Android アプリケーションのディレクトリ構成を示す.

Andorid ではリソースとプログラムを分割して配置する．プログラムは src ディレクトリ以下，リソースは res ディレクトリ以下に配置するよう定められている．さらに，res ディレクトリ以下では用途に合わせてリソースを配置する場所が定められている．以下に各リソースの用途を示す．icon.png と main.xml に関しては，プラットフォームで定められた名称ではないため，特別な用途はない.

drawable 画像ファイルを配置するディレクトリ
layout 画面レイアウトに関するリソースファイルを配置するディレクトリ
values 値に関するリソースファイルを配置するディレクトリ
strings.xml 文字列に関するリソースを記述するファイル

リソースファイルだけでなく，ファイル内の要素も参照することができる．リスト 4 の 8 行目では，findViewById() メソッドを用いて Button 要素を button オブジェクトに変換している．これにより，リソースで定義した部品にプログラム中で振る舞いを付加できる．`R.`*resourece*`.`*id* の *id* に対応する部分は，リスト 3 の 18 行目にあたる．要素に識別子を付加する方法も，プラットフォーム独自の構文がある．Android の例では “`@+id/`*id*” という構文を用いることで，要素に *id* という識別子を付加することができる.

2.1 リソースを多用したソフトウェアの問題点

プログラムからリソースとして静的データを追い出し，多種多様なリソースとして分離したことで，コードを含めそれぞれの役割にあった記述場所が細分化された．このことにより記述場所が明確になり，また構造を持つデータを適した言語で記述できるようになったことで，ソフトウェア開発が効率的になった．しかしリソースの多様化と記述場所の細分化により，プログラムからリソースの直接参照だけであったアプリケーションが，前節で示したようにプログラムからリソース，リソースからリソースといったように内部の参照が増え，プログラム内のフローやコード間の関係が複雑になった．このことにより，プログラムの保守や改変の際に重要であるプログラム理解が困難になった.

3. 関 連 研 究

3.1 プログラム依存グラフ

プログラム依存グラフ[9] は Ferrante らが提案したプログラム表現方法である．Ferrante らはプログラム内の手続きをデータ依存関係と制御依存関係によって繋ぎ，有向グラフとして表現し，計算的に関連する部分を連結した．PDG は最適化などのコード変換に適したプログラム表現方法として提案されたが，Weiser が提案したプログラムスライシング[11] と共に用いることで，プログラム理解[5]，保守[7]，デバッグ[8]，リバースエンジニアリング[4] など広い分野で利用されている．また Horwitz らがオブジェクト指向言語のための依存グラフとして PDG を拡張した SDG[12] を提案しており，文献 12) のほかにも様々な特徴を持つ PDG，SDG を発展させた依存グラフが多くの研究者によって提案されている.

3.1.1 Program Dependence Graph (PDG)

PDG は Ferrante ら[9] が提案した，コード間の依存関係を表現した有向グラフであり，手続き型言語の依存グラフとして用いられる．ノードは文，エッジは依存関係を表す．エッジは 2 種類存在し，実線はデータ依存関係を示し，破線は制御依存関係を示す．データ依存関係はある変数に定義されたデータを参照している場合に用いられる関係である．制御依存関係は if などの分岐，ループにより実行される文が変化する場合に用いられる関係である.

3.1.2 Software Dependence Graph (SDG)

SDG は Horwitz ら[12] が提案した，ソフトウェアの依存関係を表現する有向グラフである．SDG はソフトウェアのメインプログラムを表現する program dependence graph とソフト

© 2011 Information Processing Society of Japan

ウェアの補助手続を表現する procedure dependence graph(ProcDG) といくつかのエッジを含む．SDG により PDG では表現できない，補助手続内の依存関係を含めたプログラムを表現できる．オブジェクト指向言語においては，メソッドが ProcDG に相当し，表現されるメソッドの集合として SDG が扱われる．SDG は ProcDG を用いることで，手続きの呼び出しや引数を詳細に表現する．手続きのエントリである *ENTRY* ノードと呼び出しである *CALL* ノードを持ち，パラメタの中と外を分離するための *PARAM CALLEE* ノードと *PARAME CALLER* ノードなどコード以外の拡張ノードが定義された．CALLEE ノードは仮引数を表し，CALLER ノードは実引数を表す．またパラメタの参照，代入は，in と out によって表される．*in* は参照のみ可能な変数を示し，*out* は代入のみ可能な変数を示す．*in out* は参照と代入が共に可能な変数を示す．

3.2 プログラムスライシング

プログラムスライシングは Weiser によって提案されたプログラムを分割する手法である[11]．最初はプログラムのデバッグを支援するために考えられていたが，現在ではソフトウェア工学の幅広い分野で応用されている有用な技術である．文献 11) でのスライシングは振舞いに関わらず，与えたスライス基準が変化しないスライスを得る静的スライシングであるが，Agrawal らは実行トレースを加え，スライス基準に関して与えた実行トレースと振舞いをするスライスを得る動的スライシングを提案した[2]．動的スライシングでは，静的スライシングよりも小さなスライスを得ることができるため，デバッグにおけるバグの探索がより容易になる．SDG のスライシングの拡張として，文献[1),4),5),14)] ではクラスやオブジェクトを識別したり，意味や文脈を考慮したスライシングが提案されている．

4. リソースを考慮した依存グラフ

前章で述べたように，PDG と SDG などの依存グラフは広く活用される有用な技術である．しかし，これらの依存グラフはプログラミング言語の表現手法であるため，リソースに関しては考慮されていない．そのため，本研究では，考案したリソースの依存グラフと既存の依存グラフを合わせた，Source and Resource Dependence Graph (SRDG) を提案する．

SRDG は DR, RG, RDG, SG の 4 つの依存グラフで構成される．`DG` はディレクトリ構造の依存関係を表現する有向グラフであり，`RG` はリソースファイル内の依存関係を表現する有向グラフである．`RDG` は DG と RG を結合したグラフであり，リソースの依存関係を表現する有効グラフである．また `SG` はプログラムソース内の依存関係を表現する有効グラフである．`SRDG` は，SG と RDG を結合することで作成する．

4.1 DG (Directory Graph)

DG はディレクトリとファイルに関する依存グラフである．DG は 1 つのプログラムに対し 1 つ作られる．DG のノードはディレクトリを表す `dir` とファイルを表す `file` であり，エッジはディレクトリ間の関係を表す dir(親ディレクトリ) → dir(子ディレクトリ)，ディレクトリとファイルの関係を表す dir → file である．DG のノードを DNode と呼ぶ．ここで a → b は “b は a に依存する” を表現する．以降，矢印は依存関係を示し，“被依存 → 依存” を表現する．

4.2 RG (Resource Graph)

RG は 1 つのリソースファイルの内部に関する依存グラフである．RG のノードを RNode と呼ぶ．RG はリソースファイルの内部の依存関係を表現する．リソースは様々なファイル形式が混在するため，各ファイル形式に合わせて RG のノードとエッジを定義する．RG が定義できるファイル形式は人が可読なファイルとする．つまり，アーカイブファイルやバイナリファイル, 画像ファイルは対象としない．また RG はプログラムのリソースファイル毎に作られる．それらのファイルはすべて対応する DG の file ノードを持つ．

本研究では Android に対する適用実験として支援ツールを実装している．Android でもちいられるリソース記述形式である XML 形式を例として RG を具体化した．XML 形式の RG のノードは，XML 要素を表す `elm`, 属性名を表す `attr`, 属性値を表す `val`, テキスト値を表す `text` である．XML 形式の RG のエッジは，要素間の依存関係を表す elm(親要素) → elm(子要素)，要素と属性名の依存関係を表す elm ↔ attr, 属性名と属性値の依存関係を表す attr ↔ val, 要素とテキスト値の依存関係を表す elm ↔ text である．

4.3 SG (Source Graph)

SG はプログラムソース内の依存関係を表現するグラフである．プログラムソースはさまざまな言語が用いられるため，RG と同様に各言語に適した依存グラフを用いる．SG のノードを SNode と呼ぶ．Android では，プログラムソースは Java で記述されるため，本研究では Java の依存グラフである SDG を SG として用いる．

4.4 RDG (Resource and Directory Graph)

RDG は RG と DG をマージしたグラフである．RDG のノードを RDNode と呼ぶ．RDG では RG と DG のノード，エッジの他に 2 種類のエッジを追加する．1 つは RG のルートノード (ファイルのルート XML 要素の elm) と DG の file の依存関係，もう 1 つはリソース間の参照の依存関係である．

リソース間の参照はプラットフォーム固有の方法が用いられるため，各プラットフォーム

© 2011 Information Processing Society of Japan

の知識を用いる．Android の例では，“@<resource>/<id>” の形式でリソースを参照することができる．この知識に基づき，リソース間の依存関係を構築する．

4.5 SRDG (Software and Resource Graph)

SRDG は SG と RDG をマージしたグラフである．RDG と同様，SRDG は SG と RDG のノード，エッジの他に，SNode が記述される DNode との依存関係と，プログラムとリソースの参照関係のエッジが追加される．

プログラムからリソースを参照する場合，特殊な構文もしくはそれ自体を示す文字列や数値を参照している．Android の場合，“ R.<resourece>.<id>” という構文，実際には R.java という Java 言語記述されたリソースのインデックスファイルの内部クラスのフィールドを参照することによって実現している．そのためプログラムとリソースとのエッジは，SG 内で特定の値を参照している SNode とそのノードが参照しているリソースを表す RDNode を関連付ける．またリソースからプログラムの場合には，プログラム内で定義される関数名，クラス名がリソースの値として記述されるため，クラス名が記述される RDNode とそのクラス名を持つ SNode を関連付ける．

5. SRDG スライシング

SRDG におけるスライシングは SG と RDG に対し，それぞれ異なる手法のスライシングを行うことで達成する．これは既存の依存グラフを SG として用いるため，SG として実装した既存の依存グラフに適したプログラムスライシングを適用するためである．本研究で提案する RDG のスライシングは，スライス基準と対応するノードからエッジを逆に辿ることで到達可能なノードすべてを RDG のスライスとする．

以下に SRDG のスライシング手順を示す．SRDG のスライス基準は SG のノード (SNode) と RDG のノード (RDNode) の 2 種類があるため，それぞれ異なるスライシング手法を用いる．SG のスライシングには，SG として適用した既存プログラム依存グラフにあった既存プログラムスライシングを用いる．

スライス基準が SNode(RDNode) の場合の SRDG スライスシングは以下の手順で行う．

(1) SNode(RDNode) をスライス基準としスライシング
(2) 得られたスライスである SNode(RDNode) から関連のある RDNode(SNode) を取得
(3) 得た RDNode(SNode) をスライス基準としスライシング
(4) 得られたスライスである RDNode(SNode) から関連のある SNode(RDNode) を取得
(5) 手順 1 から手順 4 を繰り返し，新たに得られるスライス基準がなくなった時点でスライシングを終了し，工程で得られたスライスすべてを SRDG のスライスとする．

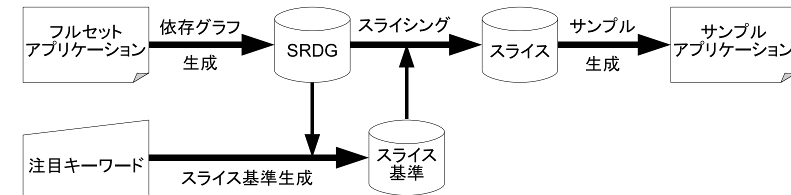

図 5 ツールの構成
Fig. 5 Struct of Our Tool

6. サンプルプログラム自動生成への応用

プラットフォームを利用する際には提供される API について学習する必要があり，実際に API を利用しているプログラムソースを読むことは有用である．実行可能なプログラムによる学習では実行時の振る舞いの確認ができ，API の実例として利用できる．しかし，注目する API を利用しており，かつ学習に適した規模のプログラムは少数であり，大半の API のサンプルプログラムは存在しない．そのため，プログラムソースから学習をする場合には，学習に適さない規模のプログラムを読むこととなるが，注目するコードをすべて発見することが困難となる．

本ツールではプラットフォーム学習支援を目的とし，オープンソースなアプリケーションからユーザが注目する点 (キーワード) と関連する部分のみで構成される，学習に適した規模のプログラムをサンプルプログラムとして生成する．本ツールの構成を図 5 に示す．また本ツールは Android に対して適用し，既存の依存グラフとスライシングに Java 静的解析ライブラリである WALA[13] を用いて，統合開発環境 Eclipse[6]q のプラグインとして実装を行った．本ツールは全体で約 6000 行である．

本ツールは 4 つの部分で構成される．まず，依存グラフ生成部でオープンソースなアプリケーションを入力し，SRDG を生成する．次に，キーワード と SRDG をスライス基準探索部 に入力し，キーワードに適した SRDG のノードを探索する．そして，スライシング部において，得られたノードをスライス基準として SRDG をスライシングする．最後にサンプルプログラム生成部 で，得られたスライスからサンプルプログラムを生成する．

6.1 依存グラフ生成部

依存グラフ生成部は入力されたアプリケーションから SRDG を生成する部分である．本

© 2011 Information Processing Society of Japan

ツールでは WALA が提供する SDG を SG として用いる．WALA ではソースコードの抽象構文木からバイトコード命令に変換し SDG を作成するため，SDG の粒度はバイトコード命令である．

SRDG の例として 2 節で用いたアプリケーションを図 6 を示す．

6.2 スライス基準探索部

スライス基準探索部では，ユーザが入力したキーワードからスライシングの軸となる注目点を探索する部分である．ユーザはキーワードを用いることにより，スライス基準を複数同時に選択することを可能とする．

キーワードには，SNode を指定するものとしてクラス名，メソッド名，オブジェクト名，オカレンスがあり，RDNode を指定するものとしてリソースファイル名，要素名，テキスト値がある．

RDNode を指定するキーワードは，キーワードと RDNode で一対一もしくは一対多に対応する．オカレンス以外の SNode を指定するキーワードは，キーワードと SNode で多対多に対応する．このとき，オカレンスと SNode は一対一で対応する．そのため，クラス名，メソッド名，オブジェクト名を組み合わせることで，スライス基準の指定を容易にする仕組みをとる．

6.3 スライシング部

スライシング部では，スライス基準探索部で得たスライス基準と依存グラフ生成部で得た SRDG を用いて，スライシングを行う部分である．ここでは例として，“Hello World, SimpleActivity!” をキーワードとして選択した結果を図 7 に示す．‘Hello World, SimpleActivity!” をキーワードとした場合，スライス基準探索部では，DNode の text(太線の赤四角) がスライス基準として得られる．

6.4 サンプルプログラム生成部

サンプルプログラム生成部では，スライシング部で得たスライスを基のサンプルプログラムを生成する部分である．スライシング部で得られたスライスは，SRDG のノードであるため，入力プログラムのコードやファイルなどの内，ノードと対応している部分を複製することでサンプルプログラムを生成する．SRDG のノードからの複製は以下の基準で行う．

DNode　DNode と対応するファイル，ディレクトリを複製

RNode　RNode と対応する要素や属性名，属性値，テキスト値を複製

SNode　SNode と対応する抽象構文木 (Abstract Syntax Tree．以後 AST) のノード (以後 ASTNode) を取得し，その ASTNode をルートとする部分木と AST のルートからその ASTNode へのパス上にある ASTNode をすべてを複製

表 1　実験結果

	削減率			適合率			再現率		
	DG	RG	SG	DG	RG	SG	DG	RG	SG
c0(クラス)	0.59	0.14	0.68	1.00	1.00	0.96	0.89	0.2	0.92
c1(クラス)	0.56	0.14	0.27	1.00	1.00	0.90	1.00	0.4	0.74
c2(メソッド)	0.67	0.48	0.61	1.00	1.00	0.946	1.00	0.6	0.98
c3(メソッド)	0.56	0.14	0.71	1.00	1.00	1.00	0.79	0.2	0.8
c4(XML 要素)	0.85	0.45	0.59	1.00	0.85	0.911	1.00	0.55	0.98
c5(XML 要素)	0.85	0.45	0.59	1.00	0.85	0.911	1.00	0.55	0.98
c6(ファイル)	0.81	0.28	0.58	1.00	1.00	0.915	1.00	0.7	0.98
平均値	0.7	0.3	0.6	1.0	0.9	0.9	0.9	0.5	0.8

本ツールでは Java の AST を扱うライブラリとして Java Development Tool(JDT) を用いる．[10]

7. 評価実験

本手法の評価として，本ツールをプラットフォームに同梱されている公式プログラムに 1 つ対して実行した出力結果の削減率，適合率，再現率について調査した．スライス基準はクラス 2 つ，メソッド 2 つ，XML 要素 2 つ，ファイル 1 つの 7 種類である．

削減率 は入力したプログラム (入力コード数) に対する出力されたプログラム (出力コード数) の割合を示し，削減率 = 出力コード数 / 入力コード数で算出する．適合率は出力されたプログラムに対するキーワードに関連するコード (出力関連コード数) の割合を示し，適合率 = 出力関連コード数 / 出力コード数で算出する．再現率は入力プログラム内のキーワードに関連するコード (入力関連コード) に対する出力関連コードの割合を示し，再現率 = 出力関連コード数 / 入力関連コード数で算出する．

また各割合に対し，1) ディレクトリとファイルの数 (DG)，2) コメントを除いたリソースファイル内の XML 要素の数 (RG)，3) コメントを除いたプログラムソース内の文字数 (SG)，の 3 項目に分けて調査した．

入力プログラムはディレクトリとファイルの数が 27，XML 要素の数が 29，プログラムソース内の文字数が 3773 である．出力結果を表 1 に示す．

表 1 より，削減率に関しては DG, RG, SG それぞれ 0.7, 0.3, 0.6 と，元のプログラムの約半分の規模のプログラムが出力されが，これは表 1 に示されているとおり，多くの加筆が必要とされるため実際には 0.6 から 0.8 割程度の規模のプログラムが出力されると予想され

© 2011 Information Processing Society of Japan

る．あまり高い削減率が示されていないが，本実験では小規模のプログラムに対して行ったため，規模によっては半分以下の規模が抽出できると考える．

適合率に関してはほぼ 1.0 と高い数値を示している．上記でも述べたように，加筆の部分が多いが，ノイズは少ないことがわかる．

再現率に関しては RG が 0.5 とかなり低い値を示しているが，他の 2 つは 0.8 以上と高い値を示している．RG の再現率の低い値は SG の加筆部分と関係がある．SRDG スライシングは SG のスライシングと RG のスライシングを交互に行うため，一度の小さな損失が，最終的に大きな影響を与える．表 1 では，SG スライシングで取得できなかった部分に RG スライシングのおける重要なスライス基準が含まれていたため，このような結果になった．このことは SNode をスライス基準とした c0–c3 よりも RDNode をスライス基準とした c4–c6 の方が再現率は高いことが示している．

8. おわりに

本稿では，リソースを含むプログラムの解析基盤技術としてリソースを考慮したプログラム依存グラフ SRDG とそのスライシング手法について提案した．SRDG は既存のプログラム依存グラフと本研究で提案するリソース依存グラフ RDG と間のノードを対応付け，リソースを考慮した依存グラフを定義した．またリソースとプログラムの依存グラフをそれぞれ交互にスライシングを繰り返すことによる SRDG のスライシングを行う手法を提案した．

また本手法の応用として携帯電話プラットフォーム Android に対し，サンプルプログラム自動生成ツールを実装した．本ツールでは入力したアプリケーションから SRDG を生成し，スライシングを行うことで，ユーザが注目する部分と関連するコードを含むサンプルプログラムを生成することができる．

応用ツールを用いた評価実験においては，再現率 0.6 程度，リソース内部の依存グラフ RG に関して注目と 0.5 以下，最低値では 0.2 と低い値を示しており，網羅的にスライス基準と関係あるコードを抽出することができていなかった．しかし，削減率 0.5 以上，適合率 0.9 以上と高い数値を得ることができ，抽出されたコードのほとんどがスライス基準と関係のあるコードであり，ノイズとなるコードが少ないことを実験から得ることができた．RG の再現率においても，RDNode をスライス基準としてスライシングする場合では，平均 0.6 と数値が高くなる．このことから，リソースからスライシングを行う場合においては特に有効であると考える．

本稿では SRDG の有用性を示すために評価実験を行った．評価実験では SNode をスライス基準とする際に，特に低い再現率を示しており，十分な数の関連コードを取得することができていないことがわかった．そのため，本手法の精度の向上を目指す．本評価実験では公式のサンプルプログラムとして提供される小規模なプログラムに対して本手法を適用したが，本手法の実用性を示すためにより大規模なプログラムを対象として実験を行う．また，本ツールは Android に対して実装したが，他のプラットフォームに対しても適用し，本手法の一般性を検証する必要がある．

参 考 文 献

1) A., V. G.: The Semantic Approach to Program Slicing, *SIGPLAN NOTICES*, Vol.26, No.6, pp.107–119 (1991).
2) Agrawal, H. and Horgan, J.R.: Dynamic program slicing, *SIGPLAN Not.*, Vol.25, pp.246–256 (1990).
3) Andorid, G.: .
4) Beck, J. and Eichmann, D.: Program and interface slicing for reverse engineering, *Proceedings of the 15th international conference on Software Engineering*, ICSE ’93, Los Alamitos, CA, USA, IEEE Computer Society Press, pp.509–518 (1993).
5) de Lucia, A., Fasolino, A.R. and Munro, M.: Understanding Function Behaviors through Program Slicing, *Program Comprehension, International Workshop on*, Vol.0, p.9 (1996).
6) Eclipse: .
7) Gallagher, K.: Using program slicing in software maintenance, *IEEE Trans. Software Eng.*, Vol.17, No.8, pp.751–761 (1991).
8) H., A.: Debugging with dynamic slicing and backtracking, *Software Practice and Experience*, Vol.23, No.6, pp.589–616 (1993).
9) J., F.: The Program Dependence Graph and Its Use in Optimization, *ACM Trans. Prog. Lang. Syst*, Vol.9, No.3, pp.319–349 (1987).
10) JDT: .
11) M., W.: Program Slicing, *IEEE Trans. Softw. Eng.*, Vol. 10, No. 4, pp. 352–357 (1984).
12) S., H.: Interprocedural Slicing Using Dependence Graphs, *Proceedings of the SIGPLAN ’88 Conference on Programming Language Design and Implementation*, pp. 35–46 (1988).
13) WALA: .
14) Zhenqiang, C.: Slicing Object-Oriented Java Programs, *ACM SIGPLAN*, Vol.36, No.4, pp.33–40 (2001).

© 2011 Information Processing Society of Japan

Vol.2011-SE-171 No.11
2011/3/14

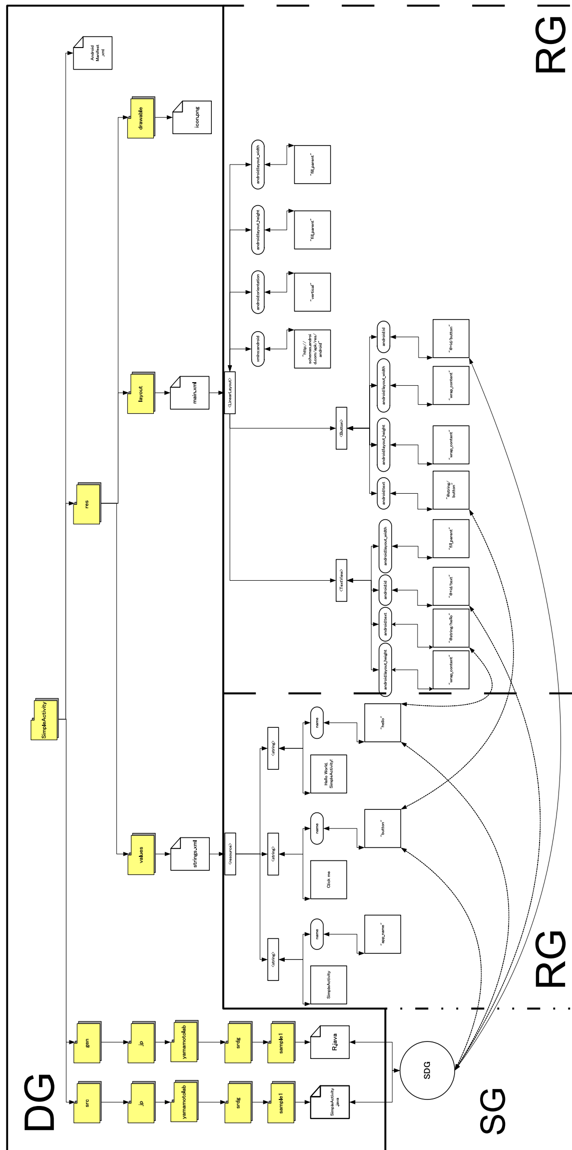

図 6 SimpleActivity アプリケーションの SRDG
Fig. 6 SRDG of SimpleActivity application

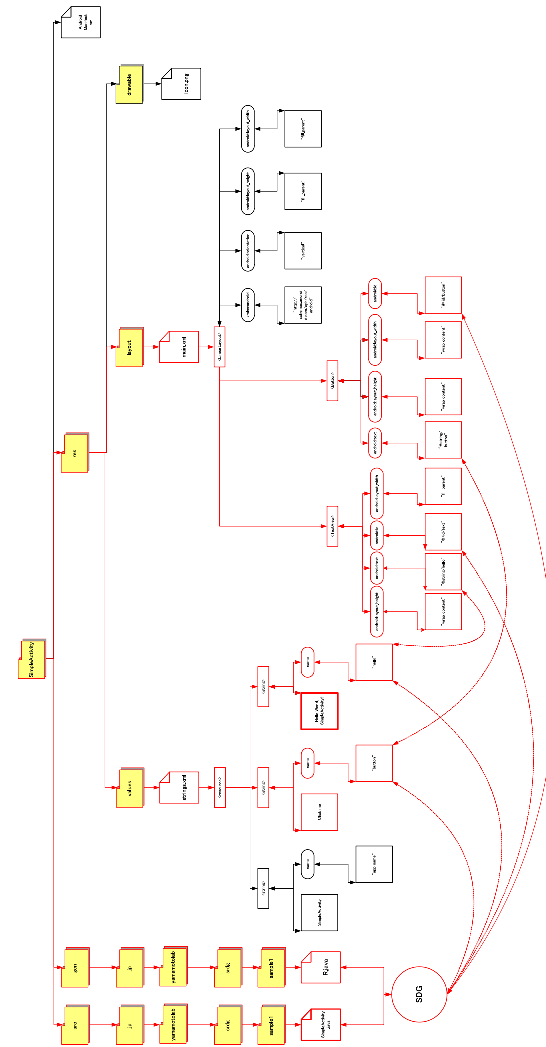

図 7 スライシングの例
Fig. 7 Example of Our Slicing

© 2011 Information Processing Society of Japan