地域分析
第49巻第２号
2011年３月

Java プログラムの実行状態の視覚化について(2)*

吉田　聡

【要旨】

Java プログラムのテストやデバッグを行う場合，そのプログラムが正しく動作しているかどうかを検証するために，実機でのプログラムの動作情報を取得する必要がある。しかし，一般に実機には Java プログラムの開発環境がなく，実機上で動作情報を取得することができないため，プログラムが要求どおりに動作しなかった場合，そのプログラムでのバグが発生した部分を特定することが困難となる。

これらのことから，Java プログラムにおけるイベントの発生や処理のタイミング，メッセージの送信内容などの情報をリアルタイムに取得し，サーバに送信する環境について提案し，オブジェクト指向型プログラムにおけるテスト段階から実用段階へ移行する際にバグの発生を防ぐことを可能とした。

本論文ではさらに，メソッド内の変数やフィールドへのアクセス内容や条件分岐や繰り返し文のアクセス内容，例外発生時の処理内容といった，手続き型プログラムにおける動作情報を実行時に取得する環境について提案する。これによって，イベントの発生や処理のタイミング，およびインスタンスの生成，インスタンス間のメッセージの交信内容の動作情報取得に加えて，手続き型プログラムの

動作情報も取得することができるようになる。

【キーワード】
Java アプリケーション，手続き型プログラム，モニタリング，開発支援環境

Ⅰ　はじめに

携帯情報端末等で動作する Java プログラムのテストやデバッグを行う場合，そのプログラムが正しく動作しているかどうかを検証するために，実機でのプログラムの動作情報を取得する必要がある。ここでの動作情報とは，イベントの発生や処理のタイミング，およびインスタンスの生成，インスタンス間のメッセージの交信内容，さらに変数やフィールドへのアクセス内容や条件分岐や繰り返し文のアクセス内容，例外発生時の処理内容などである。

しかし，一般に実機には Java プログラムの開発環境がなく，実機上でプログラムの動作情報を取得することができないため，プログラムが正しく動作しているかどうかを検証するためには，実機への入力内容に対する実行結果などから判断せざるを得ない。このため，プログラムが要求どおりに動作しなかった場合，直接プログラムのテストやデバッグを行うことができないため，そのプログラムでのバグが発生した部分を特定することが困難となる。

これらのことから，作成されたプログラムの動作情報を取得するために，その動作情報をリモートで取得し，解析する技術が必要となる。このとき，メッセージの交信内容は，送信先のインスタンスに対して直接送信するため，イベントやメッセージの内容を直接取得することができない。

以上のことから，実機で動作する Java プログラムのメソッドを変更することなく，イベントの発生や処理のタイミング，メッセージの送信内容などの情報をリアルタイムに取得し，サーバに送信する環境について提案してきた[1), 2)]。これによって，オブジェクト指向型プログラムにおけるテスト段階から実用段階へ移行する際のバグの発生を防ぐことが可能となった。

本論文ではさらに，メソッド内の変数やフィールドへのアクセス内容や条件分岐や繰り返し文のアクセス内容，例外発生時の処理内容といった，手続き型プログラムにおける動作情報を実行時に取得する方法について提案する。

Ⅱ　Java プログラムモニタリングシステム

1　システムの概要

Java プログラム間でのメッセージの交信が正しくなされているかを監視するために，Java プログラムの動作情報をリアルタイムに取得し，その情報をサーバに送信するモニタリングシステムを開発してきた。モニタリングシステムの機能モデルを図 1 に示す。

図 1　モニタリングシステムの機能モデル

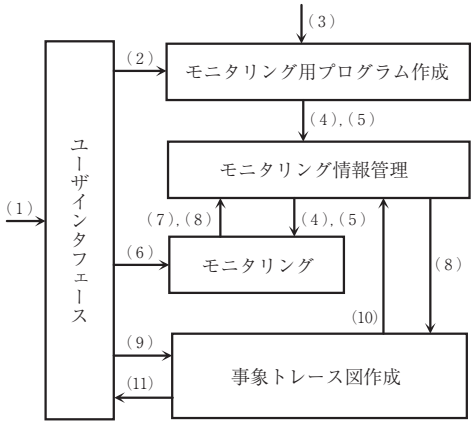


図 1 において，「ユーザインタフェース」はモニタリングシステムのユーザからの要求を受けて，システム内部に様々な指示を行う部分である。システムの画面表示などはすべてユーザインタフェースによって行われる。「モニタリング用プログラム作成」は，作成済みの Java プログラムをもとにモニタリングを行うのに必要なコードを作成し，プログラム中に挿入する部分である。「モニタリング情報管理」は，作成されたモニタリング用プログラムや各プログラムの ID などを管理する部分である。「モニタリング」は，モニタリング用プログラムを実行させて，モニタリング結果を生成する部分である。なお，モニタリングの対象となるプログラムが携帯情報端末などのリモート環境である場合，この部分は端末上に組み込まれ，実行時にモニタリング結果をインターネット経由でサーバ側へ送信する。「事象トレース図作成」は，モニタリング結果をもとに事象トレース図[3)] を作成する部分である。

また，図 1 の（1）～（11）は次の通りである。

（1）ユーザからの要求

（2）モニタリング用プログラム作成要求

（3）作成済み Java プログラム

（4）モニタリング用プログラム

（5）プログラム管理情報

（6）モニタリング要求

（7）モニタリング用プログラム要求
（8）モニタリング結果
（9）事象トレース図作成要求
（10）モニタリング結果要求
（11）事象トレース図

2　オブジェクト指向プログラムのモニタリング

本システムは，始めに作成済みのJavaプログラムに対応したモニタリング用プログラムを作成する。メッセージの交信についての情報を取得するためのモニタリング用プログラムは，作成済みのプログラムを内包したプロクシプログラムおよびゴーストプログラムから構成される。Javaプログラムでこれらのプログラムを実装する際には，作成済みのプログラムおよびゴーストプログラムをプロクシプログラムの内部クラスとして実装する。プロクシプログラムおよびゴーストプログラムを用いてメッセージの交信を中継することで，作成済みのプログラムの内容を変更することなく，メッセージの交信についての情報を実行時に取得することが可能となる。

イベント処理についての情報を取得するためのモニタリング用プログラムは，作成済みのプログラムのほかにモニタリング用のイベントリスナーから構成される。モニタリング用のイベントリスナーがイベントを受信し，作成済みのプログラムにそのイベントを渡すことで，メッセージの交信同様に作成済みのプログラムの内容を変更することなく，イベント処理についての情報を実行時に取得することが可能となる。

Ⅲ　手続き型プログラムにおける動作情報の取得

1　変数の定義と代入の情報取得

メッセージ交信についての情報を取得するために，プロクシプログラムの考え方を導入してきた。これによって，メソッドの内容を変更することなくメッセージ交信の情報を取得することが可能になった。

しかし，この方法ではメソッド内部の動作情報を取得することはできない。そこで，本システムでは，メソッド内の変数やフィールドへのアクセス内容や条件分岐や繰り返し文のアクセス内容，例外発生時の処理内容などの動作情報を取得する場合に限り，メソッド内部に動作情報を取得するための処理を記述し，これをモニタリング用プログラムとして扱う。

具体的には，作成済みのメソッド内に本システムが用意した“Monitor”クラ

スの“rec_write”メソッドを起動することによって動作情報の取得が可能となる。“rec_write”メソッドの引数は処理の種類（変数への代入，条件分岐，繰り返し，例外処理など）および参照される変数の値である。本システムは“rec_write”メソッドが起動されると，動作情報を log.csv ファイルに書き込む。

本システムは，変数宣言についての情報を取得するために，ソースコードにおいて変数宣言された次の行にモニタリング要求のメッセージ送信を 1 行追加する。図 2 に変数宣言の情報取得を示す。

図 2　変数の定義の情報取得

```
int num1, num2;
        (a)

int num1, num2;
Monitor.rec_write("variable","int,num1,num2");
        (b)
```

図 2 における（a）は元のプログラムで，（b）は動作情報取得の処理を加えたプログラムである。変数宣言が行われた直後に，本システムに対してモニタリング要求のメッセージを送信する。このメッセージの引数は，変数宣言を表す文字列“variable”および変数の型と変数の名前である。これによって，変数宣言の情報が実行時に記録される。

次に，変数への代入についての動作情報の取得について述べる。本システムは，変数への代入についての情報を取得するために，ソースコードにおいて変数への代入が行われた次の行にモニタリング要求のメッセージ送信を 1 行追加する。図 3 に変数への代入の情報取得を示す。

図 3　変数の代入の情報取得

```
num = 50;
        (a)

num = 50;
Monitor.rec_write("substitution","num,"+num);
        (b)
```

図 3 における（a）は元のプログラムで，（b）は動作情報取得の処理を加えたプログラムである。変数への代入が行われた直後に，本システムに対してモニタリング要求のメッセージを送信する。このメッセージの引数は，変数への代入を表す文字列“substitution”および変数の名前と代入される値である。これによって，変数への代入の情報が実行時に記録される。

2　条件分岐処理の情報取得

条件分岐処理についての情報を取得するためには，条件分岐そのものが行われたという情報と，正しく分岐されたという情報を取得する必要がある。本システムは，条件分岐についての情報を取得するために，ソースコードにおいて条件分岐についての文の直前の行，分岐先の先頭部分および条件分岐の終了部分にそれぞれモニタリング要求のメッセージ送信を追加する。図 4 に if 分岐の情報取得を示す。

図 4　if 分岐の情報取得

```
if(X){
  // X が true であるときの処理
}
else if(Y){
  // X が false で，Y が true であるときの処理
}
else{
  // X，Y が false であるときの処理
}
          (a)

Monitor.rec_write("START_if","X");
if(X){
Monitor.rec_write("if_TRUE","X");
  // X が true であるときの処理
Monitor.rec_write("END_if_TRUE","X");
}
else if(Y){
Monitor.rec_write("elseif_TRUE","Y");
  // X が false で，Y が true であるときの処理
Monitor.rec_write("END_elseif_TRUE","Y");
}
else{
Monitor.rec_write("if_FALSE");
  // X，Y が false であるときの処理
Monitor.rec_write("END_if_FALSE");
}
          (b)
```

図 4 における（a）は元のプログラムで，（b）は動作情報取得の処理を加えたプログラムである。if 分岐が行われる直前に，本システムに対してモニタリング要求のメッセージを送信する。このメッセージの引数は，if 文の実行を表す文字列“START_if”および分岐条件である。

同様に，分岐先の処理開始時および終了時でも，本システムに対してモニタリング要求のメッセージを送信する。分岐先の処理開始時におけるメッセージの引数は，条件が成立した場合（TRUE）には文字列“if_TRUE”および分岐条件

であり，成立しなかった場合（FALSE）には文字列“if_FALSE”である。分岐先の処理終了時におけるメッセージの引数は，条件が成立した場合には文字列“END_if_TRUE”および分岐条件であり，成立しなかった場合には文字列“END_if_FALSE”である。

else-if による分岐の場合，分岐直後に本システムに対してモニタリング要求のメッセージを送信する。分岐先の処理開始時におけるメッセージの引数は文字列“elseif_TRUE”および分岐条件であり，処理終了時におけるメッセージの引数

図 5　switch-case 分岐の情報取得

```
switch(X){
  case num1:
    // X が num1 であるときの処理
    break;
  case num2:
    // X が num2 であるときの処理
    break;
  case num3:
    // X が num3 であるときの処理
    break;
  default:
    // case 値がないときの処理
    break;
}
          (a)
Monitor.rec_write("START_switch","X");
switch(X){
  case num1:
Monitor.rec_write("case_START","num1");
    // X が num1 であるときの処理
Monitor.rec_write("case_END");
    break;
  case num2:
Monitor.rec_write("case_START","num2");
    // X が num2 であるときの処理
Monitor.rec_write("case_END");
    break;
  case num3:
Monitor.rec_write("case_START","num3");
    // X が num3 であるときの処理
Monitor.rec_write("case_END");
    break;
  default:
Monitor.rec_write("default_START");
    // case 値がないときの処理
Monitor.rec_write("default_END");
    break;
}
          (b)
```

は文字列“END_elseif_TRUE”および分岐条件である。これによって，if 文による条件分岐の情報が実行時に記録される。

次に，switch-case による分岐についての動作情報の取得について述べる。図 5 に switch-case 分岐の情報取得を示す。

図 5 における（a）は元のプログラムで，（b）は動作情報取得の処理を加えたプログラムである。switch 文が実行される直前に，本システムに対してモニタリング要求のメッセージを送信する。このメッセージの引数は，switch 文の実行を表す文字列“START_switch”および分岐条件である。同様に，case 文による処理開始時および終了時でも，本システムに対してモニタリング要求のメッセージを送信する。case 文の処理開始時におけるメッセージの引数は文字列“case_START”および分岐条件であり，処理終了時におけるメッセージの引数は文字列“case_END”である。default 文の処理開始時におけるメッセージの引数は文字列“default_START”であり，処理終了時におけるメッセージの引数は文字列“default_END”である。なお，処理終了時におけるメッセージは break 文の直前に記述する。

これによって，switch-case 文による条件分岐の情報が実行時に記録される。

3　繰り返し処理の情報取得

繰り返し処理についての情報を取得するためには，繰り返し処理のための条件判断が行われたという情報と，条件判断が行われてから正しくループの開始または終了が行われた情報を取得する必要がある。本システムは，繰り返し処理についての情報を取得するために，ソースコードにおいて繰り返し文などの直前の行，ループの先頭部分および終了部分にそれぞれモニタリング要求のメッセージ送信を追加する。図 6 に for 文の情報取得を示す。

図 6　for 文の情報取得

```
for(X;Y;Z){
  // Y が true であるときの処理
}
          (a)

Monitor.rec_write("START_for_loop","X");
for(X;Y;Z){
Monitor.rec_write("STARTloop","Y");
  // Y が true であるときの処理
Monitor.rec_write("ENDloop","Z");
}
Monitor.rec_write("END_for_loop","Y");
          (b)
```

図 6 における（a）は元のプログラムで，（b）は動作情報取得の処理を加えたプログラムである。for 文が実行される直前に，本システムに対してモニタリング要求のメッセージを送信する。このメッセージの引数は，for 文の実行を表す文字列“START_for_loop”および for 文の前処理である。同様に，ループ処理開始時および終了時でも，本システムに対してモニタリング要求のメッセージを送信する。ループ処理開始時におけるメッセージの引数は文字列“STARTloop”および繰り返し処理の継続条件である。ループ処理終了時におけるメッセージの引数は文字列“ENDloop”および後処理である。ただし，後処理において変数への代入が行われる場合には

Monitor.rec_write("ENDloop","Z");

の部分を

Monitor.rec_write("ENDloop"," 代入される変数名 "+ 代入される変数名);

に変更して記述する。

また，繰り返し処理の継続条件が“偽”となり，繰り返し処理そのものが終了する際にも本システムに対してモニタリング要求のメッセージを送信する。このときのメッセージの引数は文字列“END_for_loop”および繰り返し処理の継続条件である。

次に，while 文の動作情報取得について述べる。図 7 に while 文の情報取得を示す。

図 7　while 文の情報取得

```
while(X){
  // X が true であるときの処理
}
          (a)

Monitor.rec_write("START_while_loop");
while(X){
Monitor.rec_write("STARTloop","X");
  // X が true であるときの処理
Monitor.rec_write("ENDloop");
}
Monitor.rec_write("END_while_loop","X");
          (b)
```

図 7 における（a）は元のプログラムで，（b）は動作情報取得の処理を加えたプログラムである。while 文が実行される直前に，本システムに対してモニタリング要求のメッセージを送信する。このメッセージの引数は，while 文の実行を表す文字列“START_while_loop”および while 文の前処理である。同様に，ルー

プ処理開始時および終了時でも，本システムに対してモニタリング要求のメッセージを送信する。このメッセージは，for 文によるループと同様である。繰り返し処理の継続条件が“偽”となり，繰り返し処理そのものが終了する際におけるモニタリング要求のメッセージの引数は文字列“END_while_loop”および繰り返し処理の継続条件である。

次に，do-while 文の動作情報取得について述べる。図 8 に do-while 文の情報取得を示す。

図 8　do-while 文の情報取得

```
do{
  // 処理
}while(X);
          (a)

Monitor.rec_write("START_do_loop");
do{
Monitor.rec_write("STARTloop");
  // X が true であるときの処理
Monitor.rec_write("ENDloop");
}while(X);
Monitor.rec_write("END_do_loop","X");
          (b)
```

図 8 における（a）は元のプログラムで，（b）は動作情報取得の処理を加えたプログラムである。do-while 文が実行される直前に，本システムに対してモニタリング要求のメッセージを送信する。このメッセージの引数は，do-while 文の実行を表す文字列“START_do_loop”である。同様に，ループ処理開始時および終了時でも，本システムに対してモニタリング要求のメッセージを送信する。このメッセージは，for 文および while 文によるループと同様である。繰り返し処理の継続条件が“偽”となり，繰り返し処理そのものが終了する際におけるモニタリング要求のメッセージの引数は文字列“END_do_loop”および繰り返し処理の継続条件である。

4　例外処理の情報取得

Java では，文法上に誤りのないプログラムにおいて実行時に例外が発生した際には，それに対応した例外処理が行われる [4]。具体的には，例外が発生しうる処理を try ブロックに記述して，例外が発生した際に行われる処理を catch ブロックに記述する。なお，catch ブロックは，（　）内に例外に対応するクラスを記述することにより，例外の種類に応じて複数定義することができる。try ブロッ

クと catch ブロックの処理終了後，finally ブロックの処理を実行する。図 9 に例外処理の情報取得を示す。

図 9　例外処理の情報取得

```
try{
  // try ブロック
}catch(E e){
  // catch ブロック
}finally{
  // finally ブロック
}
          (a)

try{
Monitor.rec_write("STARTtry");
  // try ブロック
Monitor.rec_write("ENDtry");
}catch(E e){
Monitor.rec_write("STARTcatch","E");
  // catch ブロック
Monitor.rec_write("ENDcatch","E");
}finally{
Monitor.rec_write("STARTfinally");
  // finally ブロック
Monitor.rec_write("ENDfinally");
}
          (b)
```

図 9 における（a）は元のプログラムで，（b）は動作情報取得の処理を加えたプログラムである。try ブロックの開始時および終了時に，本システムに対してモニタリング要求のメッセージを送信する。このメッセージの引数は, try ブロックの開始時は文字列“START_try”であり，終了時は文字列“END_try”である。同様に，catch ブロックの開始時および終了時に，本システムに対してモニタリング要求のメッセージを送信する。このメッセージの引数は，catch ブロックの開始時は文字列“STARTcatch”および例外のクラス“E”であり，終了時は“ENDcatch”および例外のクラスである。さらに，finally ブロックの開始時および終了時に，本システムに対してモニタリング要求のメッセージを送信する。このメッセージの引数は, finally ブロックの開始時は文字列“STARTfinally”であり，終了時は文字列“ENDfinally”である。

Ⅳ　モニタリング結果

ここでは，モニタリング用プログラムの実行結果を述べる。メソッド内部に記

述された“rec_write”メソッドの呼び出し処理により，本システムは手続き処理の内容を csv ファイルに書き込む。図 10 に，for 文を利用して 1 ～ 10 の総和を求めるプログラムと実行結果を示す。

図 10　モニタリング用プログラム

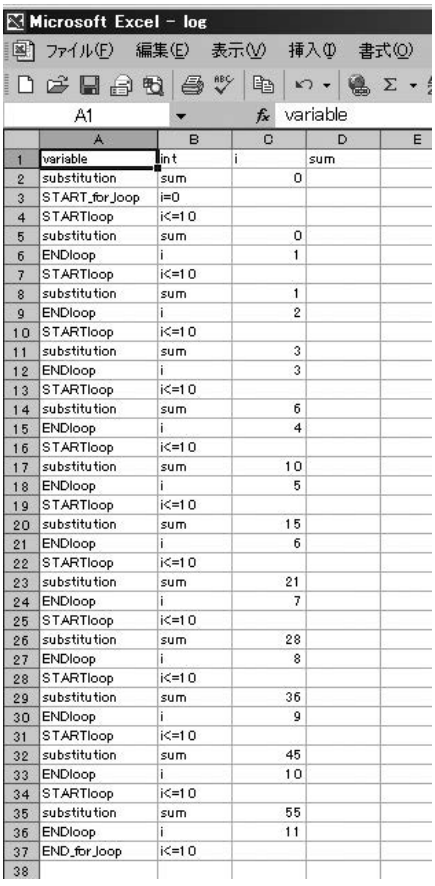
```
class Test01{
  public static void main(String[] args){
    int i, sum;
    sum = 0;
    for(i=0;i<=10;i++){
      sum = sum + i;
    }
    System.out.println("総和 = " + sum);
  }
}
                (a)

class Test01{
  public static void main(String[] args){
    int i, sum;
    Monitor.rec_write("variable","int,i,sum");
    sum = 0;
    Monitor.rec_write("substitution","sum,0");
    Monitor.rec_write("START_for_loop","i=0");
    for(i=0;i<=10;i++){
      Monitor.rec_write("STARTloop","i<=10");
      sum = sum + i;
      Monitor.rec_write("substitution","sum,"+sum);
      Monitor.rec_write("ENDloop","i,"+(i+1));
    }
    Monitor.rec_write("END_for_loop","i<=10");
    System.out.println("総和 = " + sum);
  }
}
                (b)
```

図 10 において，（a）は元のプログラムで，（b）は動作情報取得の処理を加えたプログラムである。（b）のプログラムには，図 6 で示した for 文を取得するためのコードが挿入されていることがわかる。

次に，図 11 に図 10 のプログラムのモニタリング結果を示す。

図 11　モニタリング結果



Microsoft Excel - log

ファイル(F)　編集(E)　表示(V)　挿入(I)　書式(O)

A1　fx　variable

	A	B	C	D	E
1	variable	int	i	sum	
2	substitution	sum	0		
3	START_for_loop	i=0			
4	STARTloop	i<=10			
5	substitution	sum	0		
6	ENDloop	i	1		
7	STARTloop	i<=10			
8	substitution	sum	1		
9	ENDloop	i	2		
10	STARTloop	i<=10			
11	substitution	sum	3		
12	ENDloop	i	3		
13	STARTloop	i<=10			
14	substitution	sum	6		
15	ENDloop	i	4		
16	STARTloop	i<=10			
17	substitution	sum	10		
18	ENDloop	i	5		
19	STARTloop	i<=10			
20	substitution	sum	15		
21	ENDloop	i	6		
22	STARTloop	i<=10			
23	substitution	sum	21		
24	ENDloop	i	7		
25	STARTloop	i<=10			
26	substitution	sum	28		
27	ENDloop	i	8		
28	STARTloop	i<=10			
29	substitution	sum	36		
30	ENDloop	i	9		
31	STARTloop	i<=10			
32	substitution	sum	45		
33	ENDloop	i	10		
34	STARTloop	i<=10			
35	substitution	sum	55		
36	ENDloop	i	11		
37	END_for_loop	i<=10			
38					

モニタリング結果は，Microsoft Excel などで読み取り可能な CSV ファイルとして出力される。ここでは，Excel での画面を用いてモニタリング結果を説明する。

図 11 の 1 行目は，変数宣言として int 型の変数 i と sum が定義されたことを示している。2 行目および 5 行目は，変数 sum に 0 が代入されたことを示している。3 行目は，for ループの開始および前処理として i=0 の処理が実行されたことを示している。4 行目は for ループの継続条件（i<=10）が判定され，ループの内部処理の実行が開始されたことを示している。6 行目はループの終了であり，後処理として i に 1 が代入されたことを示している。同様に，これらの動作情報が 36 行目まで反映され，37 行目にて for ループの継続条件（i<=10）が成立しないためにループ処理が終了したことを示している。

以上のことから，Java プログラムにおける手続き処理についての動作情報を実行時に取得可能であることがわかる。

Ⅴ　関連研究

関連するツールや研究として，アスペクト指向（AspectJ）[5), 6)]，Java Platform Debugger Architecture（JPDA）や Dynamic Proxy API などの環境[7)]，Java プログラムトレースツール[8)]，プログラム自動可視化ツール Avis[9)] などがある。

AspectJ は，アスペクト指向プログラミングの環境であり，アドバイス，ポイントカット，ジョインポイントなどを用いて，既存のクラスのソースコードを変更することなく動作情報を実行時に取得することが可能である。

しかし，この環境で取得可能な情報は，オブジェクトの生成，メソッドの起動，フィールドへのアクセスなどであり，変数の参照，条件分岐，繰り返し処理などの手続き型プログラムの動作情報の取得には対応していない。また，実行環境も限定されており，携帯情報端末などの実機上での動作情報を取得することも困難である。

JPDA は JavaVM のデバッグのための API であり，Java Virtual Machine Debug Interface（JVMDI）などを用いることによってスレッドやクラスの状態，メソッドの ID や名称といった情報の取得を可能にしている。また，テストやデバッグの対象となるクラスからインタフェースが分離されている場合には，Dynamic Proxy API を用いてログ出力機能を付加することが可能である。

Java プログラムトレースツールは Java Virtual Machine Profiler Interface（JVMPI）を利用したトレース作成ツールであり，このツールを用いることでクラス, インスタンス, メソッドの ID や名称などの情報の取得を可能にしている。

しかし，イベントの発生や処理のタイミング，メソッドの引数や戻り値の値などの動作情報を取得したり，取得した情報をリモートで検証したりする方法については確立されていない。さらに，携帯情報端末で動作する Java プログラムのテストやデバッグには対応していない。

Avis は，Java プログラムの構文解析を行い，プログラムのコントロールフローグラフ，実行パス，UML におけるシーケンス図などを自動生成する環境である。この環境は，プログラムの実行前にあらかじめ構文解析を行ってから，解析結果を元に動作情報を提示するものであり，実行時にプログラムの動作情報を取り込むことができない。

以上のことから，実機を用いたテストや実際のデータを利用したテストに，本システムが有効であることがわかる。

Ⅵ　おわりに

本論文では，Java プログラムにおける変数やフィールドへのアクセス内容や条件分岐や繰り返し文のアクセス内容，例外発生時の処理内容などの動作情報を実行時に取得する方法について提案した。

これによって，これまで提案してきたイベントの発生や処理のタイミング，およびインスタンスの生成，インスタンス間のメッセージの交信内容の動作情報取得に加えて，手続き型プログラムの動作情報も取得可能であることを示した。

今後は，イベントの発生や処理のタイミングやメッセージの送信内容と，メソッド内の動作情報を同時に取得する方法について検討するとともに，システムの実用化を目指していく予定である。

【参考文献】

1）高橋浩也，吉田聡，大原茂之，“Java アプレットリモートメンテナンス技術について”，情報処理学会第 63 回全国大会，6H-7，2001.

2）吉田聡，“Java プログラムの実行状態の視覚化について”，愛知学院大学産業研究所所報地域分析，Vol.46，No.2，pp.57-70，2008.

3）本位田真一，山城明宏，“オブジェクト指向システム開発 分析・設計・プログラミングへの実践的アプローチ”，日経 BP 社，1994.

4）村上雅章訳，『最新 Java 仮想マシン仕様』，ピアソンエデュケーション出版，2000.

5）千葉滋，『アスペクト指向入門』，技術評論社，2005.

6）天野まさひろ，鷲崎弘宜，立堀道昭，『AspectJ によるアスペクト指向プログラミング入門』，ソフトバンクパブリッシング，2004.

7）http://java.sun.com/javase/technologies/core/toolsapis/jpda/

8）小野寛文，相台れいこ，古宮誠一，村尾洋，今城哲二“オブジェクト指向プログラムのデバッグ／テスト支援システムの研究開発”，情報処理学会第 64 回全国大会，2002.

9）喜多義弘，徳永友樹，片山徹郎，冨田重幸“プログラミング教育支援のためのプログラム自動可視化ツール Avis の試作”，ソフトウェアエンジニアリング最前線 2007，pp223-226，近代科学社