Vol.2009-SE-166 No.16
2009/11/6

Java WEB アプリケーションにおける通信特性の解析と可視化

高野 祐輝 † **篠田 陽一** § **鈴木 正人** †

北陸先端科学技術大学院大学 情報科学研究科 † / 情報科学センター §

概 要

ソフトウェアの開発を行う上でデータやメッセージのフローを見ることは，デバッグ，パフォーマンス解析，プログラム構造の理解などをする上で非常に重要であり，これは，Web アプリケーションにおいても例外ではない．そこで，我々は，Java 言語を用いた Web アプリケーションのための，解析および可視化手法を提案し，実証ソフトウェアの実装を行った．特に，本研究では，メソッド呼び出し，ループ，通信量，オブジェクト間の関係に焦点を置いた解析を行った．これらの解析および可視化を行うことによって，Web アプリケーションのプログラムの無駄な部分や非効率な部分の発見も容易になると期待される．

Analysis and Visualization of Communication Characteristics on Java Web Application

Yuuki Takano† *Yoichi Shinoda*§ *Masato Suzuki*†

School of Information Science† / Center for Information Science§,
Japan Advanced Institute of Science and Technology

Abstract

It is important to figure out flow of data and messages on software developments for analysis and understanding structures of programs. Web applications are no exception in that to see flow is important. So, we suggest analysis and visualization methods for web applications written by Java language in this paper. Furthermore, we implemented a software to demonstrate our suggestions. Our analysys and visualization methods is expected to help detecting needless and inefficient spots of web application programs.

1 はじめに

Web アプリケーションは Web ブラウザとサーバを用いて，アプリケーションが実現される．従来のデスクトップアプリケーションと違い，Web アプリケーションはクライアントサーバ実装の一つとなっており，クライアント側での作業量が少なく，かつ，クライアント間での独立性が高くなる一方，サーバ側での作業量は多くなるという特徴を持つ．

Web アプリケーションの実現には，様々な言語，および環境が用いられるが，Java 言語の場合 Java 言語，Web サーバ，JSP コンテナを用いて実現されるのが一般的である．アプリケーションの解析を行う場合，プログラムやデータやメッセージのフローの解析を行うのは重要である．特に Web アプリケーションで扱うデータのほとんどが文字という特性もあり，フローを把握することはパフォーマンス改善においても重要である．そこで，本研究では，Java 言語を用いた Web アプリケーションの開発に焦点を当て，プログラム及びデータ・メッセージのフロー解析及び可視化を行った．解析・可視化を行うことで不要なコードや，非効率な部分の発見が容易になり，Java 言語での Web アプリケーションの開発の一助となると考えられる．

本研究では，フローの解析では主にメソッド呼び出し，ループ，通信量に主眼を置いた解析と可視化を行った．メソッド呼び出しの解析では，単純なメソッドの呼び出しだけではなく，オブジェクト間のメッセージフローの解析も行う．これらの解析・可視化によって開発者がプログラムの理解を行うことが容易になると考えられる．

2 背景と目的

Web アプリケーションは，クライアントサーバ実装の一つであり，クライアント側での作業量が少な

く，かつ，クライアント間での独立性が高くなる一方，サーバ側での作業量は多くなるという特徴を持つ．Java 言語を利用した Web アプリケーションは，サーバ側に Web サーバと，Apache Tomcat [1] 等の JSP コンテナを利用し，クライアントには Web ブラウザを用いて実現する．

Web サーバと JSP コンテナを用いた Web アプリケーションの作成には，JSP と Java 言語の両方を利用して作成する．サーバ側の操作は，基本的にデータに対する操作・管理が主となるが，JSP と Java 言語ではこれらは Bean と呼ばれるクラスオブジェクトを操作することで行う．

ソフトウェアの開発を行う上でデータやメッセージのフローを見ることは，デバッグ，パフォーマンス解析，プログラム構造の理解などをする上で非常に重要であり，これは，Web アプリケーションにおいても例外ではない．Java Web アプリケーションで Bean のフローを知ることは，データ・メッセージのフローを知ることに繋がるため，Bean のフローを人間が理解しやすい形で表すのは非常に重要である．特に，Web アプリケーションの場合文字列を扱うことが多くなるので，引数に文字列を渡す場合に通信量が非常に大きくなってしまう場合がある．そのため，データ・メッセージのフローを把握することは，パフォーマンスを向上する際にも重要となる．

そこで我々は，Java Web アプリケーションにおいてデータ・メッセージのフローの理解を容易にすべく，Java Web アプリケーションのための構造理解と特徴抽出の方法の提案を行い，提案手法を実現するソフトウェアのプロトタイプ実装を行った．なお，実装は，Eclipse プロジェクト [2] の提供する構文解析ライブラリである ASTParser，Java クラスファイルの解析を行う bcel [3]，グラフ描画ライブラリの jung [4] を用いて行った．

3 Java Web アプリケーションの解析

3.1 構造理解と特徴抽出

Web アプリケーションを開発する際，Model-View-Controller(MVC) という 3 つのロジックに分割して実装することが推奨されている．Java 用のサーブレットコンテナを用いて MVC モデルでは，基本的に Model 部分は Java で記述し View は JSP で記述する．フレームワークを利用すると，Controller はフレームワークが処理する．従って，実際に処理を行う部分は Java 言語の部分であるため，基本的には Java 言語の特徴抽出を行えばよいことになる．

図 1 は Web アプリケーションにおける，メッセージフローを示している．Java 言語はオブジェクト指向言語であるため，オブジェクト間でメッセージのやりとりを行う．そのため，Web アプリケーションの構造を把握するためには，オブジェクト間におけるメッセージのフローを把握することが重要となる．また，オブジェクト間におけるフローのみでなく，メソッドレベルでメッセージのフローを把握することは，プログラム構造の理解に役立つ．Bean と呼ばれるコンポーネントも，基本的にはクラスオブジェクトであるため，オブジェクト間のメッセージフローやメソッドレベルでのフローを把握することがデータ・メッセージのフローを把握することに繋がる．

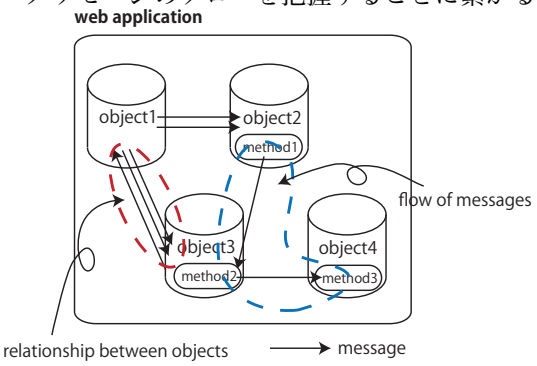

図 1: Web アプリケーションの構造

Java 言語の場合，メッセージはメソッド呼び出しを行うことによって行われる．従って，データ・メッセージのフローの解析を行うためには，メソッド呼び出しの解析を行えばよいことになる．Java 言語ではこのメッセージ呼び出しと同時に，また，メッセージがどのように呼び出されているかを知ることも重要であるため，for や while などループについても解析を行った．この解析を行うことで，ループ中に何度もメッセージが送信されている箇所などを発見できるようになる．

本研究では，オブジェクト間でのメッセージやりとりのフローをオブジェクト間のフローと呼ぶ．先に述べたように，Java 言語の場合メッセージはメソッド呼び出しを用いて行われる，従って，本研究ではオブジェクト間のフローを解析するために，クラス間におけるメソッド呼び出しに注目して解析を行った．

さらに，本研究では通信量についても解析を行った．通信量とはメッセージ送信にかかるコストの大きさのことであり，Java 言語ではメソッド呼び出しにおけるオーバーヘッドを示す．本研究では，メソッド引数の大きさをオーバーヘッドとし，バイト数を用いて通信量を表す．Java の基本型以外である場合，その大きさはあらかじめ決定されていないため，本研究で行うよな静的解析では具体的なバイト数を知ることが出来ない．そこで，引数がクラスの場合はクラス内のメンバ変数を解析して，クラスのバイト数を決定する．

ところで，Web アプリケーションの場合，メッセージのやりとりに文字列を用いることが非常に多い．これは，基本的に Web，HTTP はテキストを主体としたメディア，プロトコロルであるためである．そのため，Java 言語内でも String 型を引数にしたメソッドが多くなる．本研究では，通信量は際バイト数で表すが，String 型の場合はユーザが String 型のバイト数を指定して解析を行えるようにした．

3.1.1 メソッド呼び出しおよびループの解析

本ソフトウェアでは，メソッド呼び出しとループをノードで表し，メッセージのフローをグラフで表現する．

プログラム 1 では，foo メソッドと bar メソッドが定義されており，foo メソッド内で bar メソッドが呼び出されている．これをグラフで表現すると図 2 のようになる．メソッドを一つのノードで表し，その呼び出しを方向有りのエッジとして表すことで，メソッド呼び出しを表現している．

プログラム 1: メソッド呼び出しの解析

```
1 void foo() {
2   bar();
3 }
4
5 void bar() {
6 }
```

プログラム 2 では，foo メソッドが定義されており，foo メソッド内で for ループが実行される．また，for ループ内では配列アクセスが行われている．図 3 は，プログラム 2 のループを表現した物であり，for と data 配列へのアクセスがノードで表されている．ループ内で行われる処理であることを表すために，ループ内の処理は，ループから伸びる方向付エッジで表現される．

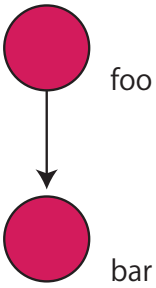

図 2: メソッド呼び出しの表現

基本的に本ソフトウェアでは，配列へのアクセスは考慮しないが，ループがある場合，どの配列へ繰り返しアクセスを行うかが重要となってくる．従って，Java プログラムの特徴を抽出するという観念から，ループ内にある配列アクセスに関しては特別に考慮している．

プログラム 2: ループの解析例

```
1 void foo(int[] data) {
2   int i;
3   for (i = 0; i < 10; i++) {
4     data[i] = i * 2;
5   }
6 }
```

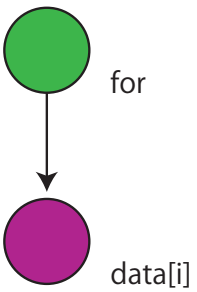

図 3: ループの表現

プログラム 3 は doGet メソッド内で，for ループを実行し，そのループ内で setAttribute メソッドを呼び出している．図 4 はプログラム 3 を表現したものである．本ソフトウェアでは，指定した Java ソースコード外で定義されているメソッドについては考慮しない．たとえば，プログラム 3 の 6 行目では，println メソッドを呼び出し標準出力に出力しているが，図 4 には現れていない．これは，println メソッドは解析するソースコードで定義されて居らず，ライブラリで定義されているからである．

一方，setAttribute メソッドは図 4 に現れている．setAttribute 等，attribute アクセスのメソッドは，Java サーブレットプログラムを作成する際に利用されるメソッドである．

基本的に HTTP はステートレスなプロトコルであるが，Java サーブレットコンテナの提供する attribute アクセスメソッドを用いることで，ステートを考慮した Web アプリケーションを作成することが出来る．

しかしながら，attribute アクセスをループ内で繰り返し行うことは，通信量が多くなり，パフォーマンスの低下に繋がってしまうので，プログラムのどの部分で attribute アクセスを行っているかを知ることは重要である．そこで，本プログラムでは，attribute メソッドの呼び出しに関して特別に考慮している．

プログラム 3 は setAttribute メソッドを for ループ内で呼び出しているため，結果的に attribute アクセスが大量に行われることになっている．図 4 では，for ノードから setAttribute メソッドのノードへエッジが伸びていることがわかる．attribute アクセスをループ中に大量に行うプログラムは効率が悪いため，代わりにリスト等を用いるべきである．

プログラム 3: attribute アクセスの解析

```
1 void doGet(HttpServletRequest request,
2             HttpServletResponse response) {
3   int i;
4   for (i = 0; i < 10; i++) {
5     request.setAttribute("user" + i, i);
6     System.io.println(i);
7   }
8 }
```

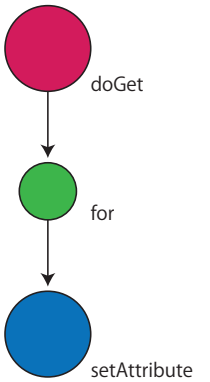

図 4: attribute アクセスの表現

3.1.2 通信量

本ソフトウェアで Java プログラムの解析を行う際には，メッセージのフローと同時に通信量の解析も行った．通信量とは，メソッド呼び出しに係るオーバーヘッドの大きさであり，具体的には，引数の合計サイズを意味する．

表 1 は Java のプリミティブ型が表現可能な範囲（バイト数）を表している．すなわち，引数に int が一つだけあるようなメソッドを呼び出した場合の通信量は，32 ビット，すなわち 4 バイトとなる．

表 1: プリミティブ型の表現範囲

型	バイト数
byte	1
char	2
short	2
int	4
long	8
float	4
double	8

引数に取る型がプリミティブ型のみであるメソッドの場合，通信量は表 1 を用いれば導出することが出来る．しかしながら，Java にはクラスがあり引数にクラスを取る場合も多くあるため，通信量を解析したい場合，クラスについても考慮する必要がある．

ところで，Java ではプリミティブ型以外の引数は参照渡しで渡される．仮想マシンの実装に依存するだろうが，メソッドの引数にクラスを取ったとしても，クラスのサイズが通信量に反映されるわけではなく，参照型変数のサイズが反映されることになる．しかしながら，参照型変数のバイト数は，JavaVM の実装に違ってくる．そこで，本ソフトウェアでは参照型変数のバイト数を int 型と同じ 32 ビットと想定して，通信量を表している．

基本的に，本ソフトウェアでは表 1 を元にして，通信量を表している．通信量は，グラフのエッジの横にバイト数で表している．ただし，表 1 以外の型である，boolean と String と，未解析のクラスは 4 バイトをデフォルト値として利用している．しかしながら，String やクラスは一意に大きさが決まらないため，4 バイトとして扱うには不適切な場合もある．そこで，本ソフトウェアでは，表 1 や，boolean，String，クラスのバイト数は随時変更可能にしてある．

3.1.3 枝刈り

メッセージのフローをグラフで表現した際，大規模なプログラムだとノードが多すぎて特徴的な部分を把握できなくなる状況が想定される．そこで，本ソフトウェアでは特に重要となるノードへの経路のみを残した，枝刈りを行えるようにした．枝刈りのアルゴリズムは以下の通りとなる．

アルゴリズム 1

key_types　**重要なノード種類**
visited　**既に訪れたノードの集合**

```
def pruning(node):
  if node.type in key_types:
    return true

  if node.children.length == 0:
    return false

  for child in node.children:
    if child in visited:
      return true
    else:
      visited.add(child)
      if pruning(child):
        return true
      else:
        node.children.remove(child)
        return false
```

アルゴリズム 1 は，対象となるグラフのルートを引数にとり，ルートから特定の種類のノードに至るまでのノード以外への経路を枝刈りする．

まず初めに，アルゴリズム 1 は，引数に取ったノードの種類が，key_types の集合に含まれるか確認し，もし含まれていない場合 true を返値として返しメソッドを終了する．ノードの種類が key_types 集合に含まれていなくて，かつ，そのノードが子ノードを持たない場合，false を返値として返しメソッドを終了する．

それ以外の場合は，ノードの子ノードを引数として，pruning メソッドを再帰的に呼び出す．ただし，呼び出す前に子ノードを visited 集合に追加する．もしも，対象となる子ノードが visited 集合に既に含まれていたら，true を返値として返しメソッドを終了する．再帰的に呼び出したメソッドの返値が true であった場合は，そのまま true を返値として返しメソッドを終了する．一方，返値が false であった場合は，子ノードの集合から対象となる子ノードを削除する．

アルゴリズム 1 で指定する key_types だが，このアルゴリズムで用いる再帰メソッドは，ノードの種類が key_types に含まれる集合の場合即座に true を返し，それ以上深く走査を行わない．したがって，key_types に指定するノードは，子を持たない葉である必要がある．

プログラム 4 は非常に単純なサーブレットプログラムである．doGet メソッドは，foo1 メソッドと，bar1 メソッドを呼び出す．また，foo1 メソッドは foo2 メソッドを呼び出し，bar1 メソッドは bar2 メソッドを呼び出す．foo2 メソッド内では何もしないが，bar2 メソッド内では setAttribute メソッドを呼び出している．

図 5 はプログラム 4 を doGet メソッドをルートとしてグラフ化したものである．一方，アルゴリズム 1 を用いて枝刈りを行った結果が，図 6 である．ただし，ここでは，attribute アクセスのメソッドを key_types の要素として指定している．

図 5 では，doGet メソッドをルートとし，foo2 メソッドと setAttribute メソッドが葉となるグラフが書かれていが，枝刈りによって，ルートから attribute アクセスを行うメソッド以外の経路が削除されている．これによって，例えば，for 文で何度も attribute アクセスメソッドを呼び出すような，非効率な部分も発見できるようになる．

プログラム 4: サーブレットプログラム

```
1  public class ServletTest
2            extends HttpServlet {
3    public void
4    doGet(HttpServletRequest request,
5          HttpServletResponse response)
6            throws ServletException {
7      foo1();
8      bar1();
9    }
10
11   public void
12   foo1() {
13     foo2();
14   }
15
16   public void
17   foo2() {
18   }
19
20   public void
21   bar1(HttpServletRequest req) {
22     foo4(req);
23   }
24
25   public void
26   bar2(HttpServletRequest req) {
27     LinkedList list = new LinkedList();
28
29     req.setAttribute("list", list);
30   }
31 }
```

3.1.4 オブジェクト間のフロー

メソッド間のフローのみではなく，オブジェクト間のフローを知りたい場合もある．本ソフトウェア

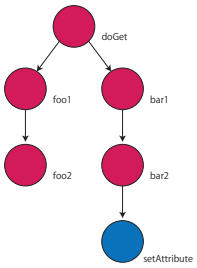

図 5: プログラム 4 のグラフ化

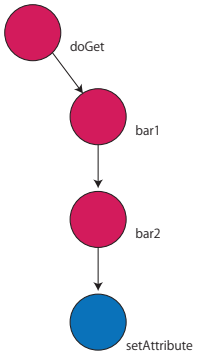

図 6: 枝刈り後

では，オブジェクトをノードで表し，オブジェクト間でのメッセージのやりとりをグラフで表現し，フローの理解を図る．

プログラム 5 では，cls1 クラスと cls2 クラスが定義され，cls1 クラスの method1 メソッド内から，cls2 クラスのメソッドを呼び出している．

プログラム 5: オブジェクト間のフロー

```
1  public class cls1 {
2    public void method1() {
3      cls2 c = new cls2();
4
5      c.methodA();
6      c.methodB();
7    }
8  }
9
10 public class cls2 {
11   cls2() {
12   }
13
14   public void methodA() {
15   }
16
17   public void methodB() {
18   }
19 }
```

図 7 はプログラム 5 に存在する，オブジェクト間のフローを表現している．なお，メソッド呼び出しとループのフローとの区別を分かりやすくするため，オブジェクト間のフローの可視化に用いるノードは円ではなく四角で表す．プログラム 5 では，cls1 クラスの method1 メソッドから，cls2 クラスの methodA, および methodB メソッドを呼び出している．さらに，cls2 を new で生成しているため，コンストラクタも呼び出している．そのため，図 7 では，cls1 クラスを表すノードから cls2 クラスを表すノードへエッジが伸びている．また，コンストラクタを含めて合計 3 カ所でメソッド呼び出しを行っているため，エッジの近くに 3 の数字が描かれている．

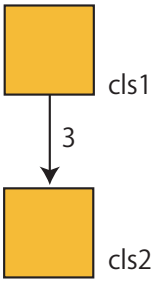

図 7: オブジェクト間フローの表現

3.2 描画

図 8 は，MyHotel にあるメソッドの一つを，本ソフトウェアを用いてグラフ化した様子を示している．MyHotel とは，Java 言語と JSP を用いて作成した，約 3000 行の Web アプリケーションである．

解析時にルートとしたメソッドは，ChangeRoomStatusServlet クラスの doGet メソッドである．赤色のノードはメソッド呼び出し，緑色はループ，紫色はレシーバへのアクセス，青色は attribute アクセスをそれぞれ示している．また，メソッド呼び出しを表すエッジには，呼び出すメソッド定義の引数のバイト数に応じた値がふられているのが分かる．

図 9 は，図 8 をアルゴリズム 1 を用いて枝刈りを行った結果である．ただし，本ソフトウェアでは key_types に attribute アクセス用のメソッドを指定しているため，ルートから attribute アクセスメソッドまでの経路以外のノードが枝刈りされる．

図 10 は，本ソフトウェアを用いて MyHotel のオブジェクト間フローをグラフ化したものである．黄色のノードがオブジェクトを表し，エッジの側にある数字がオブジェクトへメッセージを送信している箇所の数である．数が多いほど，フォントサイズを大きくするようにしてあるため，頻繁にメッセージが送信されるであろう箇所が分かりやすくなっている．

3.3 読み込みについて

本ソフトウェアで Java プロジェクトの解析を行う場合，各種パス等を渡す必要がある．解析に必要な

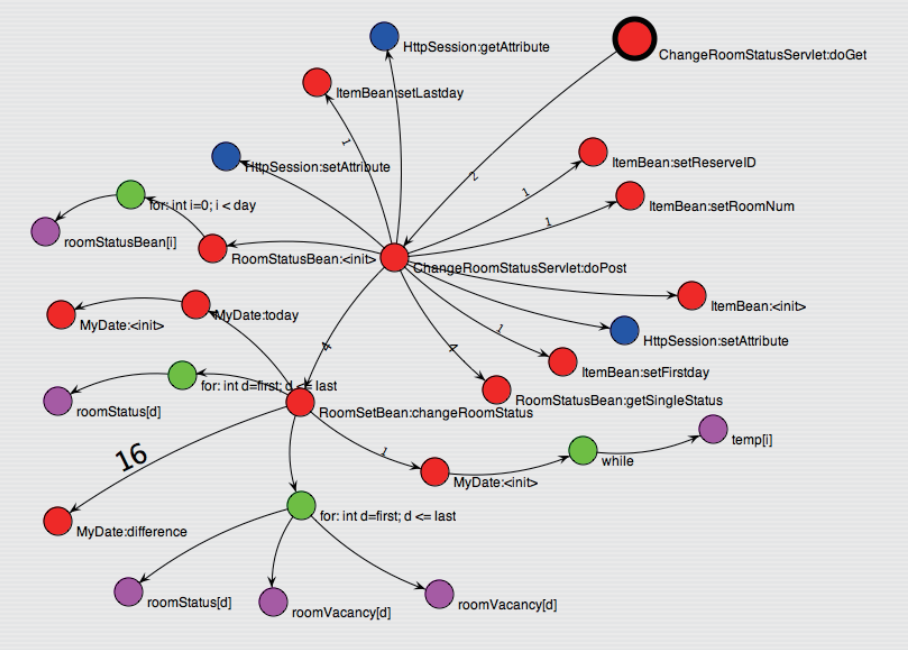

図 8: 実際の表示例

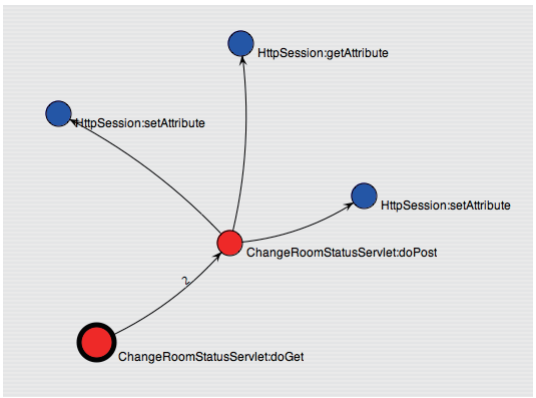

図 9: 枝刈りの表示例

パスは以下の通りである．

- プロジェクトルートディレクトリ
- Java ソースディレクトリ
- クラスパス

プロジェクトルートディレクトリは，名前の通り，プロジェクトのルートディレクトリである．

Java ソースディレクトリとは，.java ファイルが置いてあるディレクトリのトップである．

クラスパスは，Java ソースディレクトリをコンパイルして出来るオブジェクトファイルが置かれるディレクトリのトップである．

本ソフトウェアでは，パスを入力する手間を省くために，以上の順番でテキストで一行ずつ書いておくと，メニューから設定ファイルとして読み込むことが出来るようになる．

4 まとめ

ソフトウェアの開発を行う上でデータやメッセージのフローの把握を行うことは，プログラム構造の理解などをする上で非常に重要である．フローの把握を行うことで，プログラム中に存在する無駄な部分や，非効率な部分の発見も容易にもなると考えられる．そこで，本研究ではメソッド，for・while 構文，レシーバアクセス，attribute アクセス等をノードとしたグラフ構造で表し，Java プログラムのフローの可視化を行うための方法を提案し，実証ソフトウェアの実装を行った．

さらに，メソッドの重要度によって枝刈りを行うこ

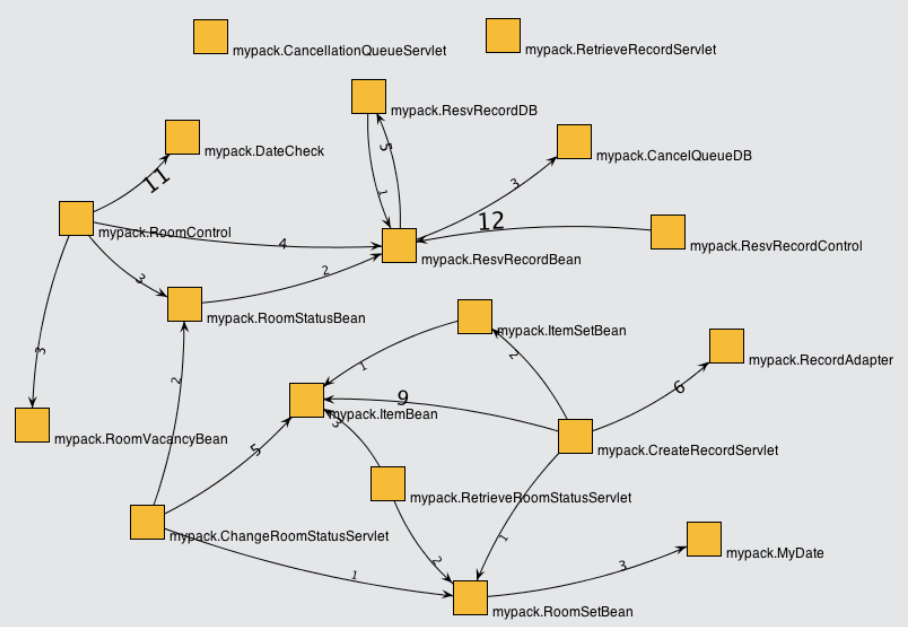

図 10: オブジェクト間フローの表示例

とによって，特に重要な部分のみに注目した解析が行えるようになった．特に，本研究では attribute アクセスについて枝刈りを行う方法を提案したが，これを行うことによって，直感的に attribute へのアクセスの理解できるようになった．これによって，attribute アクセスが for・while 文中にあるといった事などがわかり，非効率な部分の発見が容易にできるようりになった．また，メソッドの引数を元にて，通信量も可視化しているため，プログラム全体の通信量が，ある程度把握できるようになった．

特に Web アプリケーションは，文字列を用いたメッセージ送信を行うことが多く，その大きさはまちまちであり重要度も違ってくるため，通信量の解析を行うときに反映させなければならない．そこで，本研究では，文字列の重み付け，バイト数を解析中に変更できるようにした．これによって，より柔軟な通信量の解析が行えるようになったと言える．

これらの解析および可視化によって初学者などプログラミング経験の浅い者でも，ソースコードの理解を容易に行えるようになる事が期待される．さらに，プログラムの無駄な部分や非効率な部分の発見も容易になると期待される．

今後は本研究による提案・実証ソフトウェアで発見された通信特性の解析を行い，Web アプリケーションの再構成方法の提案などを行う予定である．

参考文献

[1] Apache Tomcat.

[2] Eclipse.org home. http://www.eclipse.org/.

[3] Jakarta bcel – バイトコード処理ライブラリ – - jakarta bcel project – bcel プロジェクト．http://bcel.jakarta.jp/.

[4] Java universal network/graph framework. http://jung.sourceforge.net/.