プログラミング学習者へのメソッド記述支援情報の提供

山本 貴士* 酒井 三四郎**
*静岡大学大学院 情報学研究科
**静岡大学情報学部
432-8011 静岡県浜松市城北 3-5-1
e-mail: cs9097@cs.inf.shizuoka.ac.jp

概要

現在、プログラム記述支援に関してさまざまな研究が行われている。しかし、これらの研究の多くは、フレームワークなどの構造的な面での支援に目的を置き、既存のクラス、メソッドを利用して、ソースコードを記述するときの支援機能は不足している。クラス、メソッドに関して知識の乏しい学習者は、１ステートメントを完成させるだけでも複数のドキュメントやサンプルプログラムを参照しなくてはならないことはプログラム記述のモチベーションの低下につながる。本研究では学習者に対して、java 言語の持つ標準 API に含まれるクラス、メソッドを利用するための知識を効率よく獲得できるような情報を提供する。しかし、java 1.3 から java 1.4 に移ったとき、その標準 API の数は 1000 クラス近く増加した。このように増加していくクラスに対して、一つ一つのクラス、メソッドに関して情報を登録していくのは非常にコストが高い。そこで、既存のソースコードに対して構文解析・意味解析をおこない、その結果から記述支援情報を生成する。また、その情報をユーザーに提供するためのインターフェースを持つシステムを開発し、そのシステムの有効性と提供情報の信頼性について評価を行った。

1. はじめに

近年、オブジェクト指向言語が広く普及するようになった。その結果、既存のクラスを利用してプログラム記述を行うことは非常に重要な要素となっており、ソフトウェアの生産性を大きく左右する。オブジェクト指向言語を代表するjava言語は膨大な数のクラスライブラリを持っているが、プログラマはこのクラスライブラリの中から自分が必要としているクラスやメソッドを選び出さなくてはならない。しかし、既存ツールに搭載されているメソッド記述時の支援機能は常に記述者側から先行して情報を発信しなくてはならないため、初めて利用するクラスやメソッド等の場合に記述時の負担の低減を期待できない。

そこで、本研究では初めて利用するクラスやメソッドの場合でも利用可能で、且つ「検索の負担の低減」、「記述時の意思決定支援」が可能となる記述支援情報を提供するシステムの開発を目的とする。しかし、java 1.3からjava 1.4に移ったとき、その標準APIの数は1000クラス近く増加した。このように増加していくクラスに対して、一つ一つのクラス、メソッドに関して情報を登録していくのは非常にコストが高い。そこで記述支援になりうる情報を自動生成し、データベースに保存していく。このようなシステムを開発していき、開発したシステムの有効性と提供する記述支援情報の信頼性について評価を行い考察する。

以降、2節で既存の記述支援機能の問題点とあらたな記述支援機能の提案、3節で提案するシステムの構築について述べ、4節で開発したシステムの評価・考察を行う。最後に、5節で今後の課題について述べる。

2. 背景

プログラム記述に関してはさまざまな支援が行われている。しかし、これらの多くはフレームワークなどの外枠をつくるといった構造的な面として支援であり、メソッド単体をソースコードに記述するときの支援機能は不足していると思われる。プログラム学習者にとっては、前者の構造的な面での支援機能より、後者を支援する機能のほうが必要である。以降、既存の支援機能を例にあげ、その問題点を指摘し、問題点を解決可能な支援機能の提案を行う。

2.1 ドキュメントのキーワード検索

近年注目を浴びているオブジェクト指向言語であるjavaには標準でドキュメントが付属しているが、すべてHTMLで記述されているため、ブラウザの持つ文字列検索やハイパーリンク、また履歴機能等を利用することができる。しかし、これらの機能の中で1ページ単位でしか利用することのできない文字列検索は非常に貧弱であるといえる。この機能に関して、microsoft[1]の開発した統合開発環境の持つドキュメントライブラリであるMSDNライブラリでは検索機能を強化し、全文探索を行える機能やあらかじめ登録されたキーワードによってクラスやメソッドを検索することが可能となっている。また、java標準ドキュメントに比べて内容も充実している

2.2 コード補完機能

多くの統合開発環境[1],[2]に搭載されているコード補完機能とは、記述者側からのある入力に対して次に続くべく文字列を補完し、ステートメントを完成させるために必要な情報をシステム側が提供する機能である。

2.3 既存の支援機能の問題点

二つの支援機能を例に挙げたがこれらの機能は記述者側からの情報発信を1メソッドごとに行わないといけない。すなわち、キーワード検索ではキーワードを、コード補完機能では最初の文字列をそれぞれ記述者側から発信しなくてはならない。

つまり、これらの機能は、どのようなメソッドを自分が利用したいのかを意思決定していないと利用することができない。しかし、学習者はこのような意思決定を容易にはできない。さらに、メソッドは単独では意味をなさず、複数のメソッドを用いてその機能を果たすものも存在するため、学習者は、サンプルプログラムを探しに走り回ったり、キーワードを捻出したりして見つけ出さなければならない。これは非常に負担が大きく、学習のモチベーションの低下につながる。

2.4 問題点の解決

膨大なクラスライブラリから、自分が必要とするクラス、メソッドを探す負担は非常に大きい。そこで、検索の負担を低減させるために、検索場所の限定と検索の繰り返しの除去を行うことができれば、負担を低減させることが可能であると考えられる。

そこで提案するシステムでは、あるメソッドに対して、併用する可能性の高いメソッドの情報を提供することにした。このような情報を提供することにより、一つ目のメソッドを発見することができれば、それと併用するメソッドも同時に知ることができることになり、検索の繰り返しの除去が可能となる。また、同時に、自分が調べればよいメソッドを知ることが可能となるので、キーワードの捻出などの負担を取り除き、検索場所を膨大なクラスライブラリの中からではなく、システムから提供された情報の中から探し出せるようにすることができる。

3. システムの構築

3.1 システムの提供する情報

開発するシステムでは併用するメソッドの情報を提供するが、実際に提供する情報は下記の４つである。

1. メソッドの記述法
2. 併用されるメソッドを含むステートメントとその関係
3. java標準ドキュメントへのリンク
4. サンプルソースコードへのリンク(閲覧可能な場合)

併用されるメソッドを含むステートメントとその関係とは、対象としたメソッドに対して、併用して利用されるメソッドが、「同一オブジェクトを参照するメソッド」、「同一オブジェクトが引数として参照されるメソッド」または、「そのオブジェクトのクラスと同一パッケージに存在するメソッド」で分別する(本システムではコンストラクタもメソッドとして扱う)。さらに、そのメソッドを利用するより前また

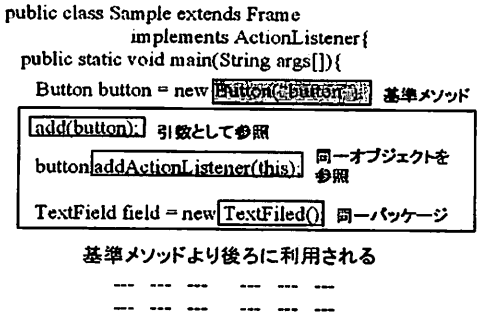

図3.1 メソッド間の関係

は後に利用されるかでも区別を行う。実際の例を図3.1で示す。

また、ここでステートメントとしているのは、複数のメソッドを組み合わせてできているステートメントが存在するため、単体のメソッド情報を扱うと不具合が生じる場合があるためである。しかし、このような情報を一つ一つのメソッドに対して登録していくのでは非常にコストが高い。そこで、既存のソースコードを多数解析し、統計的な情報を採取し記述支援情報を自動で生成することにする。この方法の利点として、一つの利用法に縛られずに他の利用法も知ることが可能であることや、利用されやすいメソッド、されにくいメソッド等の判別も可能となる。

3.2 システム構成

構築するシステムの構成を図3.2に示す。このシステムは主に、XML変換部、解析・データ抽出部、データベース、インターフェース部から構成されている。各部位については後述していく。

3.3 XML変換ツール

上述の情報を取得するために、ソースコードの解析を行う必要がある。しかし、プログラムは構文解析をして初めてその構造が明らかになる。また、メソッドはどのクラスのどのメソッドなのかを判定しなくてはドキュメントへのリンクを生成することができない。そこで、ソースコードをXMLに変換する方法をとることにした。XMLに変換することにより、構造

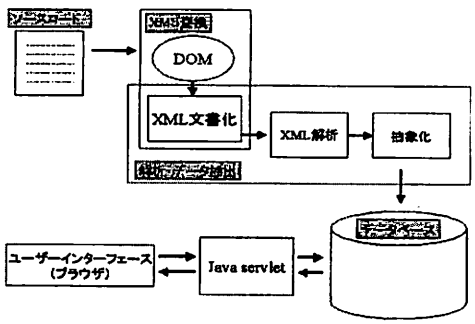

図3.2 システムの構成

をツリーで表すことができる。さらに、属性としてさまざまな付加情報を付け加えることが可能なので今後の解析が容易になる。ソースコードのXML化には多くの先行研究[3],[4],[5]が存在しているが、クラス間の外部参照が不可能であったり、フォーマットが公開されていないといった理由から、より解析しやすい独自のフォーマットを用意することにした。開発にはJavaCC[6]を利用している。

開発したXML変換ツールでは図3.3の枠内のステートメントを図3.4のように変換する。また、タグ・属性の説明を表3.1と表3.2に示す。

3.4 重みの付加と抽象化

・重みの付加

出力されるステートメントには重みを付加しなければ、そのステートメントを本当に記述すべきなのか戸惑うことになる。本研究では多くのソースコードを解析して記述支援情報を生成するシステムなので、あるメソッドに対し、「どのメソッドがどれほど併用されているのか」という統計的な情報を得ることが可能である。この統計情報の中の利用頻度とメソッド間の関係をポイント化してステートメントの重みとする。しかし、ソースコードを変換したXMLドキュメントをそのまま登録していくだけでは、ほとんどのステートメントは異なるものと判断され、重みは増えていかない。そこでステートメントの抽象化が必要となる。

・抽象化

以下の2行において、addメソッドに限れば

```
Public class Sample{
  public static void main(String args[]){
    Frame f = new Frame();
    f.setTitle("Sample");
    f.setSize(300,200);
    f.show();
  }
}
```

図3.3 javaプログラム

```
<local_var beginline="6" endline="6">
  <type class_ref="java.awt.Frame">Frame</type>
  <var_name local_id="1">f</var_name>
  <operator>=</operator>
  <constructor_call new="true" class_ref="java.awt.Frame"
  id="c3">Frame</constructor_call>
    <arguments />
</local_var>
<statement beginline="7" endline="7">
  <var class_ref="java.awt.Frame" var_ref="1">f</var>
  <method_call class_ref="java.awt.Frame" returntype="void"
  id="m20">.setTitle</method_call>
  <arguments>
    <arg type="java.lang.String">
      <value>String</value>
    </arg>
  </arguments>
</statement>
```

図3.4 XML変換の例

表3.1 タグの説明

タグ	ソースコードでの役割
Local_var	変数宣言
Statement	ステートメント
Type	型参照
Car_name	変数(宣言)
Var	変数(参照)
Operator	演算子
Constructor_call	コンストラクタ呼び出し
Method_call	メソッド呼び出し
Arguments	引数群
Arg	引数

表3.2 属性の説明

属性	ソースコードでの役割
Beginline, endline	ソースコードでの位置
Local_id	ローカル変数のID
Id	クラス内のメソッド識別
Type	型参照
Returntype	返り値の型
Class_ref	クラス参照
Var_ref	ローカル変数参照

テキストフィールドのオブジェクトをコンテナに加えるという同一のクラスに存在する同一のメソッドを使用しているが、XMLに変換したものをそのままデータベースに登録しても、データベース上ではまったく別のステートメントであると判断されてしまう。

・TextField text = new TextField;add(text);

・add(new TextField());

ステートメントに重みをつけていくために、このような意味的に同じステートメントを同一であると判断しデータベースを更新していくことにする。そのためには、適切な加工、「抽象化」を行わなければならない。実際に行っている抽象化としては以下の4つである。

1. 変数名の一般化　2. 引数の置き換え　3. 型の引き上げ　4. 型の付加

(型の付加とは、すでに宣言してある変数に対してコンストラクタを実行するときでも型を付け足すことによって変数の宣言文と同意に扱う)

3.5 ユーザインターフェース

インターフェースにはjava標準ドキュメントとの親和性や、同一ワークグループでの情報共有の利点からwebブラウザを用いることにした。図3.5に実行例を示す。

図のように、併用されるメソッド情報をカテゴリ化し、さらにドキュメントへのリンク、ソースコードへのリンクとともに提供することにより、記述者は出力された内容からのみ、自分に必要なメソッドを探せばよいため「検索の負担」が大幅に改善され、同時に「意思決定の支援」が可能になると考えられる。また、メソッド間の関係のカテゴリにはメソッドの前後間の関係もあるため、足りないステートメントの探索も容易になる。また、「次へ」のリンクより、そのステートメント内で最後に利用されているメソッドに対しての情報取得を行うことができる。

4. 評価・考察

4.1 評価実験

システムに登録する情報として、java標準API内にあるjava.awtやjavax.swingパッケージを利用しているソースコードを約250ファイルほど解析を行い、データを抽出した。このデータをもとに、出力される情報の妥当性と

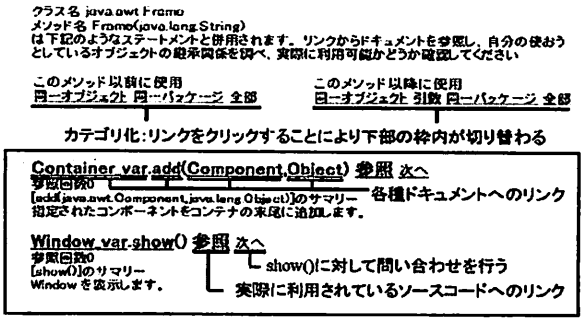

図3.5 実行例

システムの発信するプログラム解析情報から生成された記述支援情報の有効性について検証を行った。

4.1.1 出力結果の妥当性に関しての意見調査

システムを実際に使ってもらい、図4.1の内容の項目に関して意見をもらった。被験者はjava言語に対して習得度の高い大学生と大学院生7人である。アンケート結果を表4.1に示す。

4.1.2 提供情報の有効性についての意見調査

実験では被験者はjava.awtやjavax.swingのパッケージの利用に慣れていない大学生6人で、システムを実際に利用してもらい、図4.2に示すアンケートを実施した。先ほどの実験と異なるグループにしたのは、慣れていないクラスやメソッドを利用したときに、このシステムがどのような役割を果たすのかを調べるためである。アンケートの質問1の結果を表4.2に示す。

アンケート内容
システムを利用し、下記の項目についてお答えください。
1. 最上位に出力されるステートメントが妥当な確率
2. 上位3ステートメント内に妥当なステートメントが存在する確率
3. 上位3ステートメントがすべて妥当な確率
4. 妥当なステートメントが出力されない確率(出力結果0の場合も含む)

図4.1 信頼性に関してのアンケート内容

表4.1 アンケート結果

設問	0 - 20%	21 - 40%	41 - 60%	61 - 80%	81 - 100%
1	0	0	0	2	5
2	0	0	0	2	5
3	0	0	1	3	3
4	4	3	0	0	0

単位(人)

アンケート内容
質問1 システムを利用し、下記の項目についてお答えください。
1. メソッドの利用を効率よく学習することができた
2. 新しいメソッドを知ることができた
3. メソッド記述に対して有益な情報が得られた
4. メソッド記述時の負担が軽減された
5. 自分が何をすればよいのかという意思決定の支援になった

質問2 利点と欠点を挙げてください。

図4.2 有効性に関してのアンケート内容

表4.2 アンケート結果

設問	評価平均
1	4.33
2	3.83
3	4.33
4	4.00
5	3.67

図4.2の質問2からは以下の回答を得た。

利点

- メソッドを調べる煩わしさが改善された。
- あるメソッドに関して関連のあると思われるクラスやメソッドをすぐに参照できるのが良かった。
- 次に使用される可能性のあるステートメンが表示されるのは便利。
- 得たい情報の多くがブラウザのみで得ることができる。

欠点

- 登録されている情報の少なさ
- 記述支援情報より、ソースコードへのリンクがもっとも役に立っているのかもしれない。

4.2 アンケート結果に対する考察

4.2.1 出力結果の妥当性

「妥当なステートメントが出力されない確率」において21-40%域が3人と高いように思え

る。この数値ではシステムの信頼性は高いとはいえない。この問いに対しての意見を求めたところ、「出力0のものが多い」という解答を得た。今のところ、この問題に対処する方法は考えていない。しかし、頻繁に利用されない、利用場所が限定されているメソッドに関しては学習段階では必要にならないとも考えられる。

他にも、信頼性に関係していると思われる要因として、以下のことが考えられる。

・手当たりしだいに採取したサンプルソースコードの妥当性
・データの選別処理の欠如(ソースコードの良し悪しにかかわらず登録作業を行った。)

4.2.2 記述支援情報の有効性

3項目で4を超える好評価を得た。このアンケート結果よりメソッド記述時の負担は低減され、記述時の効率は改善されると思われる結果を得ることができた。しかし、実際に記述するときの意思決定の支援に対して効力を発揮していないことがわかった。このことに関して意見を求めたところ「システムの提供する情報により、確かにクラスやメソッドを調べる範囲が限定されるが、ドキュメントに記述されている内容がわかりづらいので、最終的に利用してよいのかは実際に試してみないとわからない。」という意見を得た。このシステムでは、出力されたクラスやメソッドを実際に利用していいのかという判断材料にドキュメントへのパスを提供して、容易にドキュメントを参照できるようにしているが、ドキュメントに記述されている内容だけでは出力されたステートメントを利用してよいのかという意思を決定するまでに達しない場合もあるようだ。しかし、java標準APIに関してはjava標準ドキュメントがもっとも優れた情報のライブラリであるのは間違いない。このドキュメントのより一層の充実をSun MicroSystem社に期待する。

5. おわりに

膨大なクラスライブラリから自分の望む処理を行うメソッドを探し当てるのは非常に負担が大きかった。さらに、プログラムに慣れていない学習者は自分が何をすればよいのかもわからない状態で検索を行わなければならない。

本研究ではシステム側からどのメソッド(ステートメント)を記述したらよいのかという情報をプログラム解析情報から生成し、記述者に対して発信するシステムを開発し、その有効性を評価した。実験の結果、プログラム解析情報から生成された情報を発信することにより、記述者のメソッド記述時の「検索の負担」が低減できたといえる。しかし、「意思決定の支援」に関しては思うような結果が得られなかった。しかし、これは出力結果の信頼性の向上とドキュメントの充実が進めば改善されるものと考えられる。

最後に今後の課題についてだが、システムの提供する情報の質と量を今後増やしていかなくてはならない。質の点では、開発したシステムの発信する記述支援情報は、まだ信頼できる段階に達していない。今後は、信頼性の向上と情報データベースの充実を目的とする。また、エディタに搭載するなどの発展も望みたい。

参考文献

[1]microsoft visual Studio ホームページ, http://www.microsoft.com/japan/msdn/vstudio/default.asp

[2]Borland JBuilder ホームページ, http://www.borland.co.jp/jbuilder/

[3]Badros,G.j. : JavaML : a markup language for Java source code,Computer Networks , Vol.33,No.1-6(2000),pp.159-177

[4]山中祐介・大畑文明・井上克郎 :プログラム解析情報のXMLデータベース化 -提案と実現-, コンピュータソフトウェア Vol.19,NO.1,january(2002) pp.39-43

[5]田中 哲・一杉 裕志 :ソースコード理解支援ツール --- JavaMarkup : http://spa.jssst.or.jp/2002/spa02-3-1.pdf

[6]JavaCC, http://www.webgain.com/products/java_cc/