ソースコード上での情報タグ伝播によるコードリーディング支援

石尾 隆[†], 田中 昌弘[†], 井上 克郎[†]

本研究では，オブジェクト指向プログラムのコードリーディングにおいて，開発者が調査したい変数の値やオブジェクトの型などの条件をソースコード上に記入し，その情報を用いて動的束縛等の読解を支援する手法を提案する．本稿では，提案手法の概要と，その実現に必要なデータフロー解析，部分評価など，種々の要素技術について考察する．

Code Reading Environment Propagating Information Tags

Takashi Ishio[†], Masahiro Tanaka[†], Katsuro Inoue[†]

We propose an approach to supporting code reading for object-oriented programs by resolving dynamic binding and simplifying conditional statements with information tags that represent assumptions of developers. This position paper describes background technologies such as data-flow analysis and partial evaluation to implement our code reading environment.

1. はじめに

オブジェクト指向プログラミングでは，インタフェースと実装を分離することで実装の詳細を隠し，外部モジュールとの結合を弱めている．疎結合のモジュールは，保守が容易であり，また再利用に適している．

一方で，情報隠蔽は，プログラムの詳細を理解するためのコストを上昇させる[6]．オブジェクト指向プログラムの1つの機能は，複数のオブジェクトの協調動作として実装される[5]が，情報隠蔽の結果，このような協調動作はインタフェース経由でのメソッド呼び出しとしてソースコードに記述される．そのため，デバッグやインスペクション作業で詳細な実行命令列を追跡する場合，開発者が分析したい実装クラスに関する動的束縛の解決などを開発者が逐一手作業で行う必要がある[2]．また，開発者が選択した動的束縛の解決結果等の情報は保存されないため，たとえば同じコードを読み返すたび，同じ動的束縛の解決を強いられる非効率的な作業環境となっている．

そこで，本研究では，効果的なコードリーディングのためのソースコードへのタグ付け手法を提案する．具体的な利用シナリオは次のようなものである：(1)開発者が，ソースコード上の変数を選択し，その変数が格納するオブジェクト型や値域を指定する条件タグ，あるいは自由記入のコメントタグを設定する．(2) 開発環境は，与えられたタグを，データフローに従って伝播させる．(3) メソッド呼び出しにおける動的束縛や，条件分岐がタグの情報で解決される場合，それをエディタ上に注釈として表示する．

このような開発環境があれば，紙にソースコードを印刷して読解を行う机上デバッグと類似の作業を，統合開発環境上で行うことが可能になる．情報タグの読み書きは，印刷したソースコード上にメモを記入することに相当するが，タグの伝播や保存によって，過去に調べた内容を開発者が管理可能となり，また，開発者間での情報共有も容易になることが期待される．

以降，本稿では，開発環境が提供する機能と，それを実現するための要素技術について述べる．

2. 情報タグを用いたコードリーディング

2.1. 対象とするコードリーディング作業

本手法が対象とするコードリーディングは，特定のクラス群が実装する機能（feature）[3]など，限定された量のコードの詳細な実行手順を分析する作業である．

2.2. ソースコードに記入する情報タグ

開発者は，ソースコードの読解において，しばしば，引数などの数値や，オブジェクトの型情報を仮定し，特定の条件下でのプログラムの振る舞いを演繹的に推論していく[7]．仮定された値は，他のメソッド呼び出しへの引数や戻り値として他のメソッドへと波及していくため，開発者の記憶の許す限り，各メソッドでの影響を追跡可能である．しかし，その過程や成果がドキュメントとして記述されることは少なく，過去の調査結果を後から確

†大阪大学大学院情報科学研究科

認あるいは再利用することは困難となっている.

本研究では, 開発者の持つ仮定や知識を, 条件式で記述した条件タグ, あるいは自由記述したコメントタグとして作成し, ソースコードに登場する変数名(構文木の特定のノード)に結び付けることを提案する.

条件タグとしては, 数値型変数に対する値域の指定, オブジェクト型の変数に対する以下の2種類の式, また, これらの式に対する AND, OR, NOT 演算を使用可能とする.

- `v instanceof C` (型検査)
- `v1 == v2` (比較)

図 1 に, 型情報に関するタグを付与したソースコードを示す. このコード例では, getSelectedShape 呼び出しの戻り値が IShape 型と宣言されているが, その実際の型が Circle であると仮定してタグを付与している.

このような条件タグとは別に, 自然言語による自由記述を行うコメントタグも用意しておき, 変数の値の意味などを記述可能とする. 情報タグとして変数宣言から独立させると, 同じ変数であっても中間状態と最終的な計算結果など複数の値を格納しうる場合, 異なるタグを付与して区別することが可能である.

2.3. 情報タグの伝播

ある変数の内容について記述された情報タグは, その変数の値が書き換えられるまで, たとえばメソッド呼び出しの引数として使われた場合は他のメソッドの中へも到達する. そこで, データフロー解析を用いて, タグの確実な到達および到達可能性のある変数を計算し, タグの伝播を行う. このデータフロー解析には, Hajnal らのトークン伝播に基づく解析手法を用いて, ソースコードの必要な部分に対してのみ計算する[4].

2.4. 情報タグを用いた動的束縛の解決

オブジェクトの型情報を記述した条件タグがメソッド呼び出し文に到達した場合, その型情報を使って動的束縛を解決する. これにより, 複数のメソッド呼び出しで, 一貫した動的束縛の解決を自動的に行う.

指定されたタグ情報のみで解決できない場合は, 開発者による追加の情報を付与されるまでは未解決とする.

2.5. 情報タグを用いた条件文の簡略化

条件タグが if 文や for 文などの条件式に到達した場合, それらの式に部分評価(partial evaluation)を適用することができる. これによって, 条件タグで指定された条件下で常に実行される(あるいは常に実行されない)分岐を検出し, 条件に応じたプログラムの振る舞いを効果的に理解することができる[1].

```
public void setRectangle() {
            Circle      return type: Circle
    IShape shape = getSelectedShape();
                Rectangle  Circle    return type: Rectangle
    Rectangle bounds = shape.getBounds();
  Rectangle
    bounds.setSelectedRate();
    Circle            Rectangle
    shape.setBounds(bounds);

}
```

図 1 タグ付きソースコード

3. 考察

動的束縛などによるプログラムの振る舞いは, プログラム全体に対する解析によっても部分的に解決することが可能である. しかし, そのような全体解析は, 結果の正確性や計算時間の面で, 大規模システムなどの分析には適用できない. 本研究で提案する環境は, 動的束縛などに対する全体解析を行うかわりに, 開発者が調べたいケースをあらかじめ限定してもらうことで, 解析を省略したまま, 調査を行うことを可能とする. 開発者が指定する条件タグの個数を少なくとも有用な結果が得られるかどうかが, 本研究の課題である.

謝辞

本研究は, 科学研究費補助金若手研究(スタートアップ)(課題番号:19800021)の助成を得ている.

参考文献

[1] Blazy, S. and Facon, P.: Interprocedural Analysis for Program Comprehension by Specialization. IWPC 1996, pp.133-141.

[2] Desmond, M., Storey, M.-A. and Exton, C.: Fluid Source Code Views for Just In-Time Comprehension. SPLAT 2006.

[3] Eisenbarth, T., Koschke, R. and Simon, D.: Locating Features in Source Code. IEEE Trans. Softw. Eng., Vol. 29, No. 3, pp.210-224, 2003.

[4] Hajnal, A. and Forgacs, I.: A Precise Demand-Driven Def-Use Chaining Algorithm. CSMR 2002, pp.77-86.

[5] Richner, T. and Ducasse, S.: Using Dynamic Information for the Iterative Recovery of Collaborations and Roles. ICSM 2002, pp.34-43.

[6] Spinellis, D.: Abstraction and Variation. IEEE Software, Vol.24, No.5, pp.24-25, 2007.

[7] Zeller, A.: Why Programs Fail. Morgan Kaufmann, 2006.