アスペクト指向を利用してデバッグコードを挿入できるソフトウェア開発環境

A software development environment for enabling insertion of debug code by using Aspect-Orientation

薄井 義行 † 千葉 滋 †
Yoshiyuki USUI Shigeru CHIBA

† 東京工業大学情報理工学研究科数理計算科学専攻
Dept. of Mathematical and Computer Sciences, Tokyo Institute of Technology
usui@csg.is.titech.ac.jp chiba@is.titech.ac.jp

本稿では、アスペクト指向を利用してデバッグコードを挿入できる Java 用ソフトウェア開発環境 Bugdel を提案する。アスペクト指向の利用例としてデバッグコードをアスペクトとして記述する方法が有名である。しかし、AspectJ のような汎用的なアスペクト指向システムにはデバッグに不向きな点もある。具体的には、特定の位置にはデバッグコードを入れることができない、アスペクトとして記述するデバッグコード内からデバッグ対象プログラムのローカル変数にアクセスできないなどの問題がある。また、デバッグを行うために AspectJ の文法を覚えなければならない。それに対し Bugdel はデバッグ専用のアスペクト指向システムであり、ソースコードの行番号を pointcut で指定でき、advice コード内から pointcut 位置に存在する局所変数へのアクセスを許可する。また、GUI を利用して pointcut 指定を行える。

1 はじめに

近年、オブジェクト指向を補完する技術としてアスペクト指向が注目されている。アスペクト指向の利用例としてデバッグコードの記述がある。アスペクト指向を利用することで、デバッグコードとソースコードを別々に記述できる、デバッグコードの挿入位置を一度に複数指定できるという利点がある。

アスペクト指向システムとしてアスペクト指向言語 AspectJ が有名だが、汎用的なシステムであるためデバッグには不向きな点もある。例えば、特定の位置にはデバッグコードを挿入できない、advice コードから pointcut 位置にある局所変数へアクセスできないなどの問題がある。また、利用するには AspectJ の文法を覚える必要がある。

そこで本稿ではデバッグ専用のアスペクト指向システムである Bugdel を提案する。Bugdel ではデバッグのための機能として、任意の行番号を pointcut で指定できる line pointcut を提供し、advice コード内から pointcut 位置に存在する局所変数へのアクセスを許可する。また、GUI による pointcut の指定を可能にする。これらの機能を提供するには、統合開発環境による支援が必要であり、Bugdel は統合開発環境である Eclipse のプラグインとして実装されている。

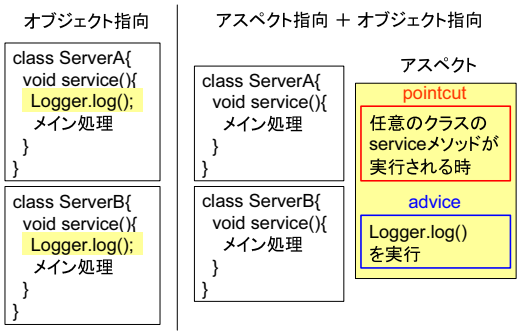

図 1: アスペクト指向を使った例

以下 2 章で既存のアスペクト指向システムを使ったデバッグの問題点を示す。3 章で本稿で提案するアスペクト指向システム Bugdel について述べる。4 章で関連研究について述べ、5 章でまとめる。

2 既存のアスペクト指向システムを利用したデバッグとその問題点

アスペクト指向 [4] とはロギングや同期処理など様々なモジュールにまたがる処理 (横断的関心事) を

アスペクトとしてモジュール化する方法である。アスペクトは主に次の要素から成る。

Pointcut どこでコードを実行するのかを指定

Advice 何のコードを実行するのかを指定

例えば、任意のクラスの service メソッドの直前にログ出力命令を実行させる場合には図 1 のようになる。

pointcut はプリミティブなものがシステムによりあらかじめ提供されている。Java を言語拡張した汎用的なアスペクト指向言語である AspectJ[3] で提供されている pointcut のいくつかを説明する。

- get pointcut: フィールドの参照位置を指定
- set pointcut: フィールドへの代入位置を指定
- execution pointcut: メソッド実行位置を指定
- handler pointcut: 例外ハンドラ実行位置を指定

advice を記述する際に AspectJ では実行のタイミングを (1)before: pointcut で指定した位置の直前、(2)after: pointcut で指定した位置の直後、(3)around: pointcut で指定した位置の処理の置き換えの 3 種類の中から選択し pointcut 位置で実行するコードを記述する。以下に図 1 のアスペクトを AspectJ で記述した例を示す。

```
1public aspect LogAspect{
2  pointcut p():
3       execution(void *.service());
4  before():p(){
5    Logger.log();
6  }
7}
```

このようにアスペクト指向プログラミングではプログラム中に散らばる処理をアスペクトとしてモジュール化できる。また、元のソースコードを変更せずに新たな実行コードを追加できる。なお、元のプログラムにアスペクトで指定したコードを埋め込むことを weave と言う。

2.1 アスペクト指向を利用したデバッグコードの記述

アスペクト指向の利用例として、デバッグコードをアスペクトとして記述する方法がある。アスペクト指向を利用することで次のような利点が得られる。

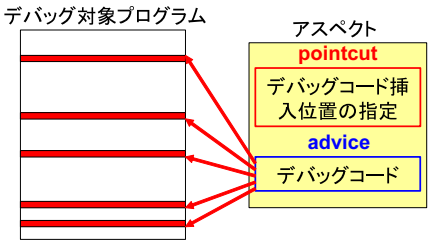

図 2: アスペクト指向を利用したデバッグコードの記述

まず、ソースコードとデバッグコードを別々に記述できる。デバッグコードは本来プログラムには関係ないものである。そのため、ソースコード中にデバッグコードを記述するとソースコードが読みにくくなってしまう。また、デバッグコードの挿入、削除を行う時に誤って、その周辺のプログラムを変更してしまう可能性がある。デバッグコードをアスペクトとして記述するとデバッグコードとソースコードを別々に記述することができ、ソースコードの可読性が低下することがなく、ソースコードを変更してしまう可能性もない (図 2)。

二つめの利点は、pointcut によりデバッグコードの挿入位置をまとめて指定できる点である。例えば「ある変数 x に値を代入している位置全て」という pointcut を指定すればシステムが対象となる位置を検索し自動的にデバッグコードを挿入する。

2.2 AspectJ/AJDT を利用したデバッグコード挿入の問題点

アスペクト指向を利用する環境として、アスペクト指向言語 AspectJ が有名である。また、AspectJ の利用を統合開発環境で支援するツールとして Eclipse のプラグイン AJDT(AspectJ Development Tools)[1] がある。前節でアスペクト指向を使ったデバッグの有用性を述べたが、AspectJ は汎用的なアスペクト指向言語であるためデバッグには不向きなも点ある。

一つめは、特定の位置を pointcut で指定できない点である。デバッグの際には、ある特定の位置、例えばソースコードの 10 行目にデバッグコードを挿入したいと考える場合がある。しかし、AspectJ では任意の位置を pointcut で指定することはできず、ある種の位置にはデバッグコードを挿入することができない。なぜなら、AspectJ が提供する pointcut はフィールドアクセスやメソッド呼び出しなどオブジェ

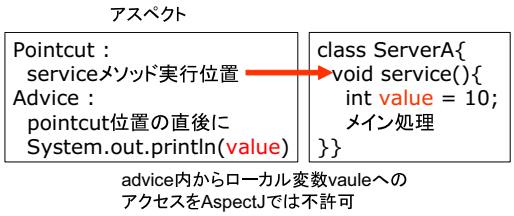

図 3: pointcut で指定した位置に存在するローカル変数へのアクセス

クト指向において主要なイベントを指定するものである。その際、フィールドアクセスならフィールド名、メソッド呼び出しならメソッド名など、クラスの実装に依存しない部分を指定する。それに対して特定の位置、例えばソースコードの 10 行目というのはクラス (メソッド) の実装に依存するものである。そのため、このような位置を pointcut で指定できると、メソッドの実装を安易に変更できなくなり、クラスのモジュール性を損ねてしまう。そのため AspectJ では任意の位置を pointcut で指定する機能を提供していない。

二つめは、advice コード内から pointcut 位置に存在するローカル変数にアクセスすることができない点である。例えば、図 3 の advice は記述できない。そのため、デバッグ対象プログラムに存在するローカル変数の値をログ出力させることができない。このような制約があるのは、ローカル変数もメソッドの実装に依存するものであり、advice コード内からのアクセスを許可すると、クラスのモジュール化を損ねてしまうからである。また、private 変数へのアクセスにも制約がある。AspectJ はクラスのカプセル化を阻害しないように設計されているため、advice コード内からクラスの private 変数へアクセスは基本的には行えず、アクセスを許可する場合には privileged 修飾子などの特別な記述を行う必要がある。

三つめは、アスペクトを記述するには AspectJ の文法に沿った記述を行う必要があり、デバッグをするために AspectJ の文法を覚えなければならない点である。また、提供されるプリミティブな pointcut の種類も覚える必要がある。

3 Bugdel

Bugdel とはアスペクト指向を利用してデバッグコードを挿入できる Java 用のソフトウェア開発環境

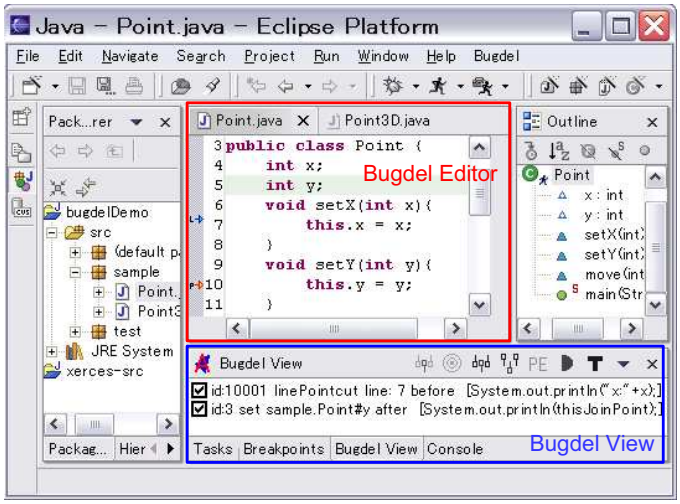

図 4: Bugdel Editor, Bugdel View

である。統合開発環境である Eclipse[2] のプラグインとして実装されており、ユーザーインターフェイス部分は主に Bugdel エディタと Bugdel ビューから構成されている (図 4)。

Bugdel では 2 章で示した AspectJ の問題点を解決するために次の特徴を持つ。(1) ソースコードの行番号を pointcut を指定する line pointcut 機能を提供。(2)advice コード内から pointcut で指定した位置に存在する局所変数へのアクセスを全て許可する。(3)GUI を利用して pointcut の指定を行える。これらの機能について以下の節で詳しく述べる。

3.1 提供する pointcut

Bugdel ではフィールドアクセス、メソッド呼び出しなどの pointcut に加えて line pointcut を提供する (表 1)。line pointcut によりソースコードの行番号を pointcut で指定できる。line pointcut は 2.2 節で述べたようにメソッドの実装に依存するためクラスのモジュール性を低下させてしまう。しかし、デバッグには必要であると考え提供する。

line pointcut を提供するには統合開発環境による支援が必要である。例えば、ソースコードの 10 行目に line pointcut を指定した後、10 行目よりも上の行に改行文字が記述された場合、line pointcut で指定した行番号を 11 行目に更新しなければならない。この問題に対して、Bugdel では統合開発環境 Eclipse のリソースマーカー機能を利用して行番号の監視を行うことで対応している。

pointcut	説明
fieldSet	フィールド代入位置
fieldGet	フィールド参照位置
methodCall	メソッド呼び出し位置
constructorCall	コンストラクタ呼び出し位置
methodExecution	メソッド実行位置
constructor Execution	コンストラクタ実行位置
handler	例外ハンドラ実行位置
cast	キャストの実行位置
instanceof	instanceof 演算子の実行位置
line pointcut	ソースコード中の行番号

表 1: Bugdel が提供する pointcut の一覧

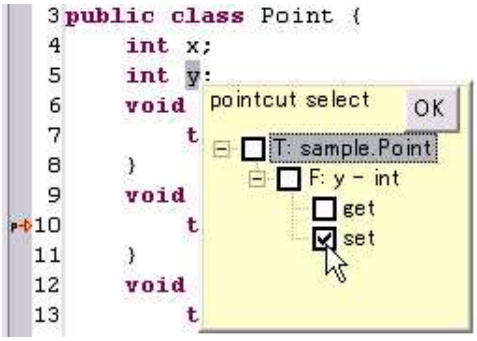

図 5: pointcut の候補の表示

3.2 GUI を利用した pointcut の指定

pointcut の指定は AspectJ のようにプログラミング言語で記述するのではなく Bugdel エディタやダイアログなど GUI を利用して指定する。GUI を利用して pointcut の指定を行えるので、Bugdel を利用するために覚えることが AspectJ よりも少ない。

クラスメンバに関する pointcut の指定を行うには、マウスを使ってソースコード中のフィールド名、メソッド名又はクラス名を選択する。次にポップアップメニューを表示させ「pointcut」を選択すると pointcut の候補が表示されるので目的のものを選ぶ。図 5 の例ではフィールド名 y を選択しているので y に対して set(fieldSet) と get(fieldGet) が候補として表示されている。

また、ダイアログを使って pointcut を指定することもできる。Eclipse のメニューバーの項目から「Bugdel」→「pointcut」を選択するとダイアログが表示されるので必要な項目を埋める (図 6)。その際、

図 6: pointcut ダイアログ

入力するクラス名やメソッド名などに”*”や”+”を利用することができる。”*”はワイルドカード、”+”は任意のサブクラスという意味である。例えば、pointcut として methodCall を選択し、クラス名に「Point+」、メソッド名に「get*()」を入力すると Point クラスのサブクラスのメソッド名が get で始るメソッドの呼び出し位置が対象となる。ただし、ダイアログを使って pointcut の指定する場合にはクラス名などをユーザが入力する必要がある。

line pointcut の指定は、指定する行のルーラー上 (エディタの左側) でポップアップメニューを表示させ「line pointcut」を選択する。またはルーラー上でマウスをダブルクリックする。

指定された pointcut の情報は Bugdel ビュー (図 4) に表示される。また、pointcut の対象になる位置にはマーカーが表示されデバッグコードが挿入される位置を把握することができる。

3.3 advice コードの記述

Bugdel ビュー (図 4) に表示されている pointcut を選択すると図 6 と同様のダイアログが表示される。その中に advice コードを記述する。

3.3.1 局所変数へのアクセス

Bugdel では advice コード内から pointcut で指定した位置に存在するローカル変数へのアクセスを許

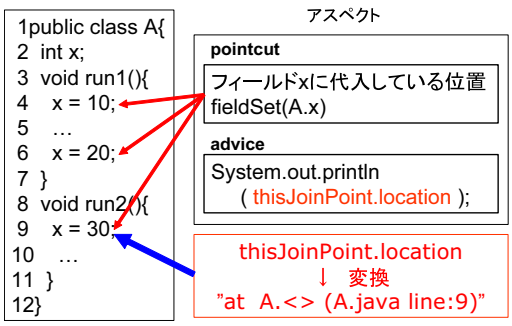

図 7: thisJoinPoint.location の利用例

可する。図 3 で示した advice が実行可能である。2.2 章で述べたように、ローカル変数はメソッドの実装に依存するものであり、クラスのモジュール性を低下させるため AspectJ ではアクセスを許可していない。しかし、デバッグの際には必要であると考え Bugdel では許可する。また、クラスの private 変数へのアクセスも無条件に許可する。このような機能を提供すると存在しない変数名を advice コードに記述する可能性がある。そこで、存在しない変数を記述した場合、警告のマーカーをエディタ上に表示させユーザに知らせる。

3.3.2 動的な情報を表す特殊変数

pointcut の対象となる位置のファイル名と行番号を表す特殊変数 thisJoinPoint.location が提供されている。この変数は pointcut で指定されている位置に応じて、それを表す文字列に変換される (図 7)。この他にも特殊変数が提供されており、その中のいくつかを説明する。

thisJoinPoint pointcut 位置のコンテキスト情報

thisJoinPoint.target pointcut が fieldSet、fieldGet、methodCall の場合ターゲットオブジェクト

$r, $w, $0, $1, $2, … Bugdel の処理系の中で使われているライブラリ Javassist が提供する特殊変数である。それぞれの意味は pointcut の種類によって異なる。例えば、pointcut として handler を指定した場合、$1 は catch 文で取得する例外オブジェクトを表す。この時、advice コードに「$1.printStackTrace();」と記述すると例外オブジェクトの printStackTrace() メソッドの呼び出しコードが指定される。これらの詳細は文献 [5] で述べられている。

3.4 デバッグコードの埋め込み (weave)

デバッグコードの挿入はクラスファイルを変換することで行なわれる。埋め込みの実行はメニューバーの項目から「Bugdel」→「weave this file」または「weave all」を選択した際に実行される。「weave this file」は現在開いているソースコードに対応するクラスファイルに、「weave all」はプロジェクト全体のクラスファイルにデバッグコードを埋め込む。クラスファイルにデバッグコードが埋め込まれるため、クラスファイルの実行に Eclipse や Bugdel は必要ない。

4 関連研究

AspectJ[3] は Java を言語拡張した汎用的なアスペクト指向言語である。また、AspectJ の利用を統合開発環境で支援する Eclipse のプラグイン AJDT[1] がある。しかし、2 章で述べたように AspectJ/AJDT は汎用的なアスペクト指向システムであるため、Bugdel が提供する line pointcut や pointcut 位置の局所変数へのアクセスなどの機能が無い。

5 まとめと今後の課題

本稿ではアスペクト指向を利用してデバッグコードを挿入するシステム Bugdel について述べた。Bugdel はデバッグ専用のシステムであり line pointcut、advice コード内から pointcut 位置の局所変数へのアクセス、GUI による pointcut の指定など汎用的なアスペクト指向システムには無い機能を提供する。

今後の課題として、Bugdel を使って指定した pointcut、advice から AspectJ のコードを自動生成する機能を考えている。

参考文献

[1] AJDT aspectj development tools eclipse subproject. http://www.eclipse.org/ajdt.

[2] Shavor, S., D'Anjou, J., Fairbrother, S., Kehn, D. and Kellerman, J. "The Java Developer's Guide to Eclipse". Addison-Wesley, March (2003).

[3] Kiczales, G., Hilsdale, E., Hugunin, J., Kersten, M., Palm, J., Griswold, W.G. "An Overview of AspectJ". In *Proc. ECOOP 2001*, pp.327-353 (2001).

[4] Kiczales, G., Lamping, J., Menhdhekar, A., Maeda, C., Lopes, C., Loingtier, J.M.,Irwin, J.: "Aspect-oriented programming". In *Proc. ECOOP 1997*, pp.220-242 (1997).

[5] Shigeru Chiba and Muga Nishizawa. "An easy-to-use toolkit for efficient Java bytecode translators". In *Proc. GPCE 2003*, pp.364-376 (2003).