日本ソフトウェア科学会第 28 回大会 (2011 年度) 講演論文集

横断的関心事に対応したオブジェクト指向言語 GluonJ とその織り込み関係の可視化ツール

大谷 晃司　千葉 滋

本研究では，横断的関心事に対応したオブジェクト指向言語である GluonJ の可視化ツールを提案する．GluonJ は Java に最小限の拡張を加えてアスペクト指向を実現した言語である．アスペクトに該当する新しい機構，リバイザを用いて横断的関心事の分離を図る．リバイザはクラスの一種であり，アドバイスもメソッドであるため，織り込みは上書きと似た概念と見なすことが出来る．このような織り込み方法では，織り込み関係を統一的に見せられるアウトラインビューが可能になる事を述べる．

1 はじめに

今日，ソフトウェア開発を行う上で，オブジェクト指向言語は必要不可欠である．しかし，オブジェクト指向言語でも，分離できずに各ソースコードに散在してしまう関心事が存在する．そのような例として，ロギング処理や図形エディタの再描画処理などが挙げられる．このような関心事を横断的関心事と呼ぶ．横断的関心事を別のモジュールにひとまとめにして扱うことの出来る技術として，アスペクト指向がある．アスペクト指向を用いると，横断的関心事をモジュールに分離出来るため，一箇所にまとめて編集を行う事が出来る．

アスペクト指向言語の例として AspectJ が挙げられる．AspectJ はオブジェクト指向言語である Java 言語を拡張する事でアスペクト指向を実現した言語である．AspectJ では，クラス内に散在している横断的関心事を，アスペクトという新たな機構を用いて別のモジュールへの分離を達成している．しかし，アスペクト指向を用いて横断的関心事を別のモジュールへと分離させた場合，分離させる前のクラスからは横断的関心事に関するコードが完全に消えてしまうため，分離させた横断的関心事がどこに適用されるかを，元のクラスの側から知る方法が存在しない．このような問題に対し，AspectJ での開発を支援するツールとして AJDT(AspectJ Development Tools) が存在する．しかし，AspectJ はクラスとアスペクトを使い分けてアスペクト指向を実現しているため，織り込み関係を調べる時にアスペクトに関する情報，クラスに関する情報と言ったように，様々なツールを並用して用いる必要がある．

そこで，本研究では横断的関心事に対応したオブジェクト指向言語である GluonJ を用いた場合の可視化ツールについて述べる．GluonJ ではアスペクトに該当する新しい機構，リバイザを用いて横断的関心事を分離する．リバイザはクラスによく似ているので，アドバイスもメソッドとして定義出来るため，織り込みを上書きと似た概念と見なすことが出来る．これにより，織り込み関係を統一的に見せられるアウトラインビューが実現可能である事を示し，その拡張アウトラインビューの詳細を述べる．

以下，2 章は既存のアスペクト指向言語の織り込み関係の表示方法，3 章で横断的関心事に対応したオブジェクト指向言語，GluonJ の説明をし，4 章で織り込み関係を統一的に見せる拡張アウトラインビューを述べる．5 章では関連研究を挙げ，6 章でまとめを述

An object-oriented language GluonJ for crosscutting concerns and its visualization tool.

Koji Otani, Shigeru Chiba, 東京工業大学大学院数理・計算科学専攻, Dept. of Math-ematical and Computing Sciences, Tokyo Institute of Technology.

べる．

2 既存のアスペクト指向言語の織り込み関係の表示

オブジェクト指向言語により，ソフトウェアをモジュール化し，関心事を分離する技術は一定程度成功したと言える．しかし，オブジェクト指向言語でも分離出来ずに各ソースコードに散在する関心事が，ソースコードの質を劣化させてしまっている．そのような例として，ロギング処理や図形エディタの再描画処理などが挙げられる．このような関心事を横断的関心事と呼ぶ．アスペクト指向言語とは，横断的関心事を別のモジュールにひとまとめにして扱うことの出来る言語である．アスペクト指向言語は，オブジェクト指向言語では分離できなかった関心事を別のモジュールにひとまとめにして扱うことを可能にしている．アスペクト指向言語の機構については以下のようになっている．

- **アスペクト**
  横断的関心事を一つにまとめた新しいモジュール単位．ポイントカットとアドバイスから構成される．
- **ポイントカット**
  ジョインポイントの集合で，いつアドバイスを実行するかの指定を行う．条件を定め，プログラム実行中に存在するジョインポイントを限定し，集合を作る．
- **ジョインポイント**
  プログラム実行中で，アドバイスを織り込む事が可能な時を表す．ポイントカットにより選択される．
- **アドバイス**
  クラスでいうメソッドに相当する．ポイントカットで指定された時に実行される処理．
- **織り込み**
  クラスやアスペクトをジョインポイントで結びつける処理の事．
- **織り込み関係**
  どのアドバイスが，どのジョインポイントでメソッドに織り込まれるかという情報を指す．

アスペクト指向言語の例として AspectJ [4] がある．AspectJ ではアスペクトと呼ばれる新しいモジュールを作成し，横断的関心事をひとまとめにする事が可能である．しかし，横断的関心事をアスペクトに分離した事により，織り込み関係を知るにはプロジェクト内の関係のあるソースコードを自分で読んで把握する必要がある．これはプログラマにとって大変労力のいる作業であり，作業効率を低下させる原因となる．そのため，このような開発を支援するツールが必要不可欠であると言うのが一般的な見解となっている．

AJDT(AspectJ Development Tools) [5] は AspectJ 開発を支援するための Eclipse [6] のプラグインである．Eclipse は統合開発環境の一つでオープンソースのソフトウェアである．AJDT には AspectJ 開発を支援するためのツールが多数用意されている．しかし，AspectJ 開発支援ツールである AJDT では，織り込み関係を統一的に同じツールで見せる事が出来ない．以下に，AJDT が提供する支援ツールを列挙する．

エディタ

エディタでは，ソースコード内のどこに織り込みが行われるかを表示している．しかし，どのアドバイスが織り込まれるかという内容は表示されないため，全てのソースコードを確認しなければならない．

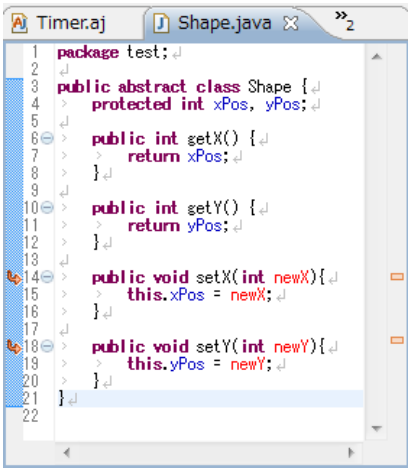

図 1 エディタ

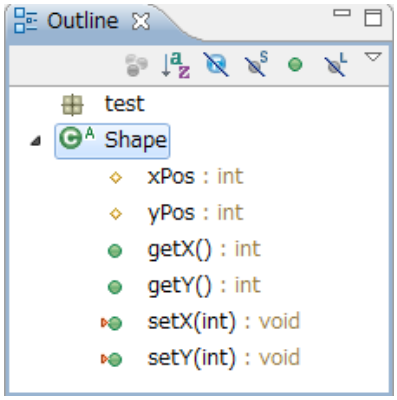

図 2　アウトラインビュー (クラス側)

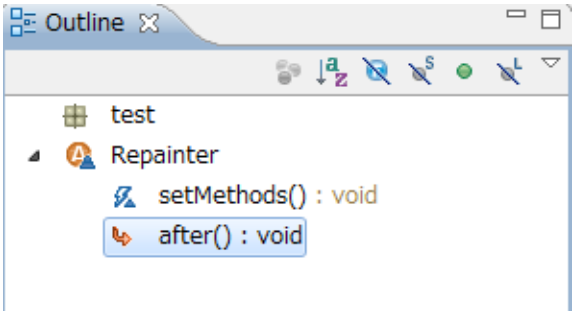

図 3　アウトラインビュー (アスペクト側)

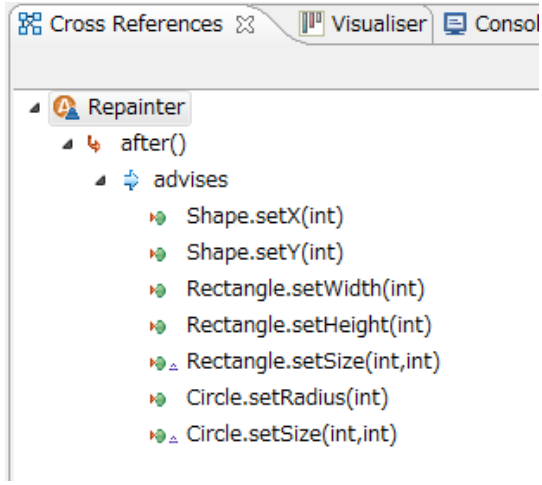

図 4　Cross References ビュー

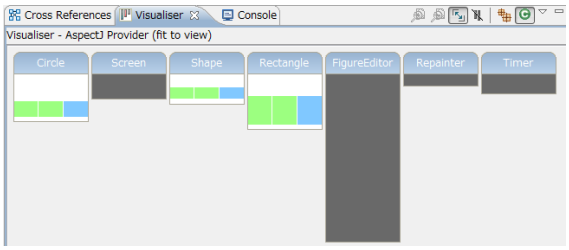

図 5　Visualizer ビュー

アウトラインビュー

アウトラインビューでは現在アクティブになっているファイルの内容を表示する．AJDT が提供するアウトラインビューは，クラス側とアスペクト側で表示が異なる．クラス側のアウトラインビューは図 2 である．織り込みが行われるメソッドに対して，メソッド名の前にあるマークに赤い矢印が付けられている．これにより，このメソッドに織り込みが行われるという事を知る事が出来る．しかし，どのアドバイスが織り込まれるかという情報を知る事は出来ない．

また，アスペクト側のアウトラインビューは図 3 のようになっている．こちらはアスペクト内で定義している内容，ポイントカット名，アドバイスの種類を列挙している．しかし，どこにアドバイスを織り込むかという情報は提供していないため，その情報を知る事は出来ない．

Cross-References ビュー

Cross References ビューはアウトラインビューと同様に，現在アクティブになっているファイルの内容を表示する．Cross-References ビューでは，アスペクト側と織り込みが行われるクラス側の双方から，織り込み関係を知る事が出来る．しかし，織り込み関係以外のフィールドや他に定義されているメソッドの情報が表示されないため，そのような情報を得るためには，他のビューを参照する必要がある．

Visualiser ビュー

Visualiser ビューは，パッケージ内にあるファイルに対してのアスペクトの影響が表示される．クラスごとに縦の棒グラフが表示され，内部に横方向に色のついた線が入っている．縦方向はソースコードの相対的な長さを表し，色のついた横線が入っている場所

はソースコードでアドバイスが実行される箇所を表している．異なるアスペクトは色の違いで区別され，どのアスペクトのアドバイスも実行されない場合はクラス内の要素は黒色表示となる．クラス内の横線をクリックする事で，ソースコード内の該当する行へとジャンプする事が可能である．

このビューにより，全体としてどの箇所にどのアスペクトが織り込まれるかという全体像は把握出来る．しかし，メソッド名が表示されないため，織り込み関係を得るためにはエディタにジャンプし，該当する箇所を読まなければならない．

3 横断的関心事に対応したオブジェクト指向言語: GluonJ

第二章では，代表的なアスペクト指向言語である AspectJ における開発支援ツール，AJDT について述べてきた．しかし，AJDT では，織り込み関係を統一的に同じツールで見せられないという問題が存在した．

第二章の問題は，横断的関心事に対応したオブジェクト指向言語を使用する事で解決する事が出来る．GluonJ [1] [2] という言語は，オブジェクト指向言語に最小限の拡張を加えてアスペクト指向言語を実現している．この章では，横断的関心事に対応したオブジェクト指向言語，GluonJ の機構について説明する．

Reviser

GluonJ では，リバイザと呼ばれる機構を用いて横断的関心事のモジュール化を図る．リバイザは，AspectJ のアスペクトに該当する物になるが，リバイザはクラスの一種である．ここでは，例として図形エディタの再描画処理を例に考える．リバイザでは，どのクラスに織り込みを行うかを指定するために，以下のように記述する．

Java で継承を行う際に extends と表記する部分を revises と表記する事で，リバイザを定義出来る．コードの織り込みは，メソッドの上書きとして記述する．図 6 のように，元々Shape クラスに定義されている set メソッドをオーバーライドするように記述すると，Shape クラスの set メソッドがリバイザで書かれた内容に上書きされる．また，super メソッドを呼ぶと上書きする前のメソッドの内容が呼び出される．GluonJ では，以上のような記述方法で，横断的関心事を分割する事が出来る．

```
1 protected class
2       ShapeRepainter revises Shape{
3    public void setX(int newX){
4       super.setX(newX);
5       Screen.repaint();
6    }
7    public void setY(int newY){
8       super.setY(newY);
9       Screen.repaint();
10   }
11 }
```

図 6 再描画処理を行う ShapeRepainter リバイザ

Require 節

GluonJ では，同じメソッドに対して複数の織り込みが行われる場合を想定して，Require という機構を備えている．Require はリバイザ同士の優先順位を決定するための機構である．

例えば，先ほどの例で set メソッドの実行時間を測

```
1 protected class
2    ShapeTimer requires ShapeRepinter
3       revises Shape{
4    public void setX(int newX){
5       double beforeTime
6             = System.nanoTime();
7       super.setX(newX);
8       double afterTime
9             = System.nanoTime();
10      // 実行時間の取得
11      double execTime
12            = afterTime - beforeTime;
13      System.out.println("setX: "
14                         + execTime);
15   }
16   public void setY(int newY){
17      ...
18   }
19 }
```

図 7 実行時間を計測する ShapeTimer リバイザ

りたいとする．この時，新たに ShapeTimer リバイザを図 7 のように定義する．set メソッドの時間計測は，repaint メソッドも含めて計測を行いたい．そのため，織り込みの順番は ShapeRepainter リバイザを適用してから ShapeTimer リバイザを適用するという順番が重要になってくる．この織り込みの重なりがあった時の順番を決めるために，二行目の requires が存在する．Require は自分より前に適用するリバイザを指定する事が出来る．逆に，二つ以上のリバイザを織り込む際，Require を用いて，適用する順番が定義されていない場合にはエラーが発生する．

Within メソッド

特定のメソッドから呼び出された時にのみ，メソッドの織り込みを行いたい時には within という機構を用いる．例えば，先ほどの例で，FigureEditer クラスの mouseDragged(MouseEvent) メソッドから Shape クラスの set メソッドが呼ばれた時のみ，織り込みを行い，他の場所から呼び出された場合には織り込みは行わないとしたい時には，次のように記述する．within の後にクラス (とメソッド) を定義する事で within で指定されたクラス (のメソッド) 内から呼び出された時のみ，メソッドの織り込みを行う事が出来る．

```
1 protected class
2       ShapeRepainter revises Shape{
3   public void setX(int newX)
4       within FigureEditer
5           .mouseDragged(MouseEvent){
6       super.setX(newX);
7       Screen.repaint();
8   }
9   public void setY(int newY)
10      within FigureEditer
11          .mouseDragged(MouseEvent){
12      ...
13  }
14 }
```

図 8　Within を用いた ShapeRepainter リバイザ

```
1 @Reviser
2 @Require(ShapeRepinter.class)
3 protected class
4       ShapeTimer extends Shape{
5   @Within(FigureEditor.class)
6   @Code("mouseDragged(MouseEvent)")
7   public void setX(int newX){
8       double beforeTime
9             = System.nanoTime();
10      super.setX(newX);
11      double afterTime
12            = System.nanoTime();
13      // 実行時間の取得
14      double execTime
15            = afterTime - beforeTime;
16      System.out.println("setX: "
17                         + execTime);
18  }
19  ...
20 }
```

図 9　アノテーションを用いた ShapeTimer リバイザ

アノテーションを用いた記述法

GluonJ では，Java のアノテーションを用いた記述法も存在する．図 6 で書かれた revises の部分を extends に変え，クラス定義の前に@Reviser と記述する事で，リバイザを定義出来る．Require はクラス定義の前に@Require と記述し，引数に前に適用させたいリバイザを記述する事で定義できる．また，within メソッドは，メソッドの前に@Within を挿入して引数にクラスを指定する．さらにメソッドも指定したい場合は，@Code を挿入し，引数にメソッド名を記述する事で指定出来る．

4　織り込み関係を統一的に見せるアウトラインビュー

3 章で述べた，GluonJ のような横断的関心事に対応したオブジェクト指向言語では，織り込み関係の表示も一つのツール，アウトラインビューだけで統一的に見せる事が出来る．その拡張アウトラインビューは，各メソッドが，どのメソッドから上書きされているか，どのメソッドを上書きされているかという表示を既存のアウトラインビューに加える事で表現出

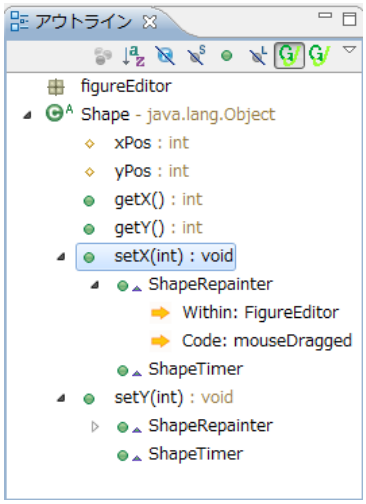

図 10　拡張アウトラインビュー (クラス側)

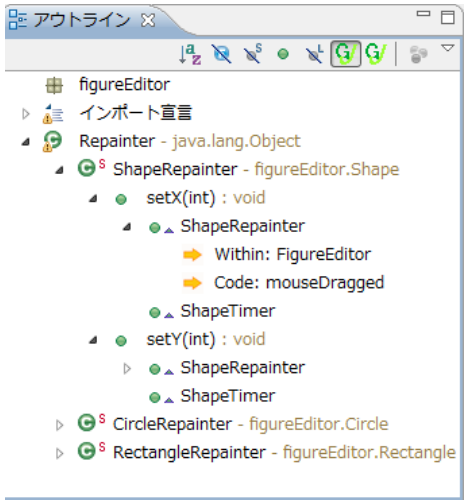

図 11　拡張アウトラインビュー (アスペクト側)

来る．

なぜなら，GluonJ では，アドバイスもメソッド単位で定義され，横断的関心事をモジュール化するアスペクトとなるリバイザもクラスの一種であるからである．これにより，繰り込む方も繰り込まれる方も，どちらもメソッドでありクラスとなるため，繰り込みは上書きと似た概念と見なす事が出来る．ゆえに，先ほど提案した拡張アウトラインビューで，クラスもアスペクトであるリバイザも表示する事が出来る．これにより，繰り込み関係を視点に関わらず統一的に見せるアウトラインビューが可能となる．

拡張アウトラインビュー

実際の拡張アウトラインビューが図 10 である．図 10 は繰り込まれる方であるクラス側のアウトラインビューとなっている．クラス名の葉に，フィールド，メソッドなど，既存のアウトラインビューと同様にクラスの要素を列挙している．そして，繰り込みが行われるメソッドには，さらに葉として繰り込みが行われる順番で，アスペクト (リバイザ) の名前を列挙している．さらに，アスペクト (リバイザ) が within メソッドを設定していた場合，その情報をさらに葉として表示を行う．これにより，クラスの視点から繰り込み関係をアウトラインビューで知る事が出来る．

また，図 11 が繰り込む方であるアスペクト (リバイザ) 側のアウトラインビューとなっている．アスペクト側でも同様に，まず要素をすべて列挙し，繰り込みを行うメソッドには，さらに葉として繰り込みが行われる順番で，アスペクト (リバイザ) の名前を列挙している．また，アスペクト側では，繰り込みを行う対象をクラス名の隣に表示している．図 11 の場合，ShapeRepainter の行にある figureEditor.Shape の事である．これにより，アスペクトの視点からも，繰り込み関係をアウトラインビューで知る事が出来る．

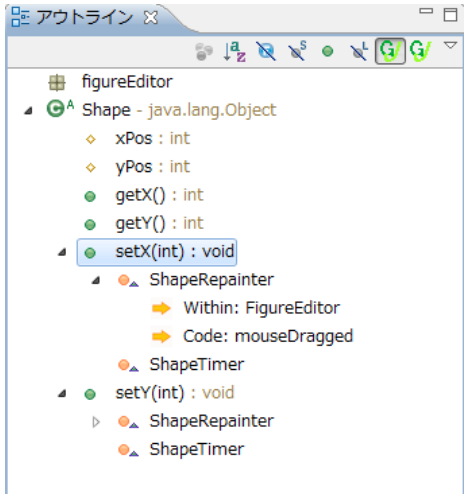

図 12　エラー表示

以上から，繰り込み関係の表現が，繰り込む方であ

るアスペクトからの視点と，織り込まれる方であるクラスからの視点のどちらかの視点からも，統一的にアウトラインビューで織り込み関係を見せる事が出来る．

また，この拡張アウトラインビューは，アスペクトの織り込む順番が定まらない場合にも対処している．それが図 12 となっている．ShapeRepainter と ShapeTimer の織り込みを行う順番が Require 等の情報から一意に定まらない場合，実行時にエラーが起きるため，その情報をメソッドの前に付いているアイコンの色を変える事で提供している．

既存ツールとの比較

2 章で挙げた AJDT で提供している各ツールとの比較をする．AJDT のエディタでは，ソースコード内のどこに織り込みが行われるかを，行の左のアイコンで可視化している．今回拡張を行ったのはアウトラインビューのためエディタでの可視化は出来ないが，拡張アウトラインビューでは表示されている要素をクリックする事で，エディタの該当する行へとジャンプする事が可能である．これにより，どのメソッドに織り込みが行われるかという情報を可視化している．

既存のアウトラインビューで可視化している情報は，拡張アウトラインビューでも同じように可視化している．既存のアウトラインビューでは，現在アクティブになっているファイルの要素の列挙に加えて，織り込みが行われるメソッドに対してマークが付けられていた．拡張アウトラインビューでは，ファイルの要素の列挙に加えて，織り込みが行われるメソッドに対してはマークではなく階層上で表示する事で可視化している．また，AJDT ではクラス側のアウトラインビューとアスペクト側で表示が異なっていたが，拡張アウトラインビューでは，クラス側，アスペクト側で同じ表示方法で提供を行っている．

Cross-References ビューでは，アスペクト側では，どこのクラスのどのメソッドに織り込みを行うかという織り込み関係を，クラス側では，どこのアスペクトから織り込みが行われるかという織り込み関係を可視化している．拡張アウトラインビューでは，アスペクト側でもクラス側でも，階層上で上から順番に織り込みが行われる順番を列挙する事で可視化している．

Visualizer ビューでは，パッケージ全体でのアスペクトの影響を棒グラフの形で可視化している．拡張アウトラインビューはクラス単位での表示であるため，Visualizer ビューで可視化している内容に対しては可視化できていない．しかし，コードの編集を行う際に全体の影響を可視化する必要は少ないのではないかと考えている．

5 関連研究

関連研究として，AspectMaps [3] を挙げる．これはアスペクト指向言語である AspectJ の開発支援ツールである．AspectMaps は次の三つの点に重点を置いて，織り込み関係の可視化を図っている．

- アスペクトがシステム内のどこに適用するように指定されているか，ジョインポイントを基準に可視化する．
- アスペクトが，それぞれのジョインポイントでどのような相互作用を及ぼすかを可視化する．
- 拡張性を考慮して，複数レベルのズーム機能を備えている．

図 13 が実際の AspectMaps の見方である．メソッド名の下が三段に分かれ，上から順番に before execution，around execution，after exceution のジョ

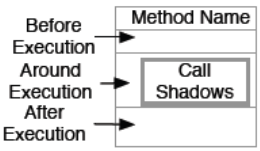

図 13 AspectMaps

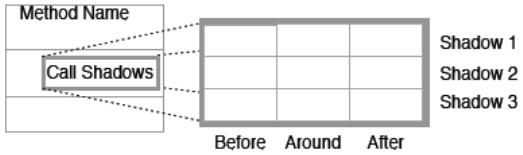

図 14 AspectMaps

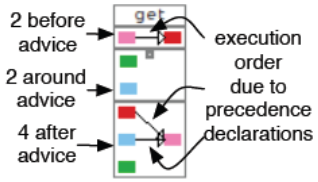

図 15　AspectMaps

インポイントに対応している．また，Call Shadows は，図 14 の様になっていて，こちらは call のジョインポイントに対応している．

実際使用した例が図 15 である．アスペクトごとに色が付けられ，それぞれアドバイスがどこで定義されるかを表示する．AspectJ には，declare precedence で織り込みの順番を決める事が出来る．これにより，織り込みの順番が決められたアスペクト同士には，図 15 のように，矢印で織り込まれる順番を提供している．

6　まとめ

織り込む方であるアスペクトからの視点と，織り込まれる方であるクラスからの視点の二つの視点に対して，織り込み関係を統一的に見せられるアウトラインビューを実現するために，横断的関心事に対応したオブジェクト指向言語である GluonJ について述べた．さらに，実際に折り込み関係を統一的に見せられるアウトラインビューの実装を行った．このアウトラインビューを用いることで，織り込み関係をアウトラインビューのみで統一的に見る事が出来る．

今後の課題としては，within メソッドで指定されたクラスにも織り込み関係を提供する事が挙げられる．現在，アウトラインビューで織り込み関係を提供しているのは，織り込む方のアスペクトと，織り込まれる方のクラスのみである．アスペクトでは，特定のクラスからメソッドが呼び出された時にのみ織り込みを行う，within メソッドという機構が存在する．これにより，within メソッドで指定されたクラスにも，織り込み関係を提供する必要がある．そのため，within メソッドで指定されたクラスにも，織り込み関係を提供する事を考えている．また，実際にプログラマにアウトラインビューを使用してもらい，評価も行いたいと考えている．実際に使用する事で生産効率を向上させる事が出来るのかを検証したい．

参 考 文 献

[1] Chiba, S., Igarashi, A. and Zakirov, S.: Mostly modular composition of crosscutting structures by contextual predicate dispatch, *Proceedings of the ACM international conference on Object oriented programming systems languages and applications*, pp. 539–554 (2010).

[2] Chiba, S., Nishizawa, M., Ishikawa, R. and Kumahara, N.: GluonJ home page, `http://www.csg.is.titech.ac.jp/projects/gluonj/`.

[3] Fabry, J., Kellens, A. and Ducasse, S.: AspectMaps: A Scalable Visualization of Join Point Shadows, *Proceedings of 19th IEEE International Conference on Program Comprehension*, pp. 121–130 (2011).

[4] the Eclipse Foundation: AspectJ, `http://www.eclipse.org/aspectj/`.

[5] the Eclipse Foundation: AspectJ development tools(AJDT), `http://www.eclipse.org/ajdt/`.

[6] the Eclipse Foundation: Eclipse.org home, `http://www.eclipse.org/`.