AOP とメタデータアノテーションを用いた ネットワーク共有機能実装プロセスの自動化

Automated implementation of network-sharing capability by using AOP and metadata annotation

中口 孝雄 [†,††]　廣瀬 誠 [†]　山縣 敬一 [†]

Takao NAKAGUCHI　Makoto HIROSE　Keiichi YAMAGATA

[†] 京都情報大学院大学

The Kyoto College of Graduate Studies for Informatics

[††] 株式会社アントラッド

Untrod, Inc.

m04w0026@m1.kcg.edu　m_hirose@kcg.ac.jp　k_yamagata@kcg.ac.jp

アスペクト指向プログラミングでは，従来複数のメソッドに散在していたコードを単一のアスペクトにモジュール化し，ビルド時に挿入するという構成手法を用いる．本稿で述べる WhiteDog Studio では，その手法をネットワーク共有機能実装プロセスの省力化へ応用した．GUI 上にクラス/メソッドツリーを表示し，そこで選択されたメソッドの実行がネットワーク上で共有されるようなアスペクトコードを生成してビルドし，ネットワーク共有機能を挿入するというものである．しかし，挿入するネットワーク共有機能を正しく動作させるには，適切なメソッドを選択する必要があるため，その実装に関する知識が要求される．本稿では，各メソッドが選択に適しているかをソースコードの開発者から提示させる機構を導入することで，ネットワーク共有機能の実装プロセスを自動化する手法について論じる．

1 はじめに

アスペクト指向プログラミング (AOP)[1] は，近年注目を集め，盛んに研究や実践が行われているプログラミング技術である．AOP では，複数のモジュールに散在する横断的な関心事をまとめるためのモジュール化単位であるアスペクトを導入し，プログラムの保守性や再利用性を向上させる．このアスペクトは，コードを挿入する点 (ジョインポイント)，その集合 (ポイントカット)，実際に挿入する処理の定義 (アドバイス) で構成される．そしてアスペクトウィーバーを用いてビルドされ，対象となるポイントカットにアドバイスで指定された処理が挿入される．アスペクトウィーバーとは，アスペクトをプログラムに挿入する処理を行うツールであり，その代表的なものとしては AspectJ[2][3] が挙げられる．本稿においても，この AspectJ を使用する．また，AspectJ ではアスペクトの定義に AspectJ 言語を用いている．本稿では，この AspectJ 言語で記述されたコードをアスペクトコードと呼ぶ．

WhiteDog System[4] では，ネットワーク共有機能の実装プロセスを省力化するために AOP を用いた．これは，オブジェクトのネットワーク共有機能に必要なライブラリおよび基底アスペクトをあらかじめ実装し，AOP を用いてプログラムに挿入することで，ネットワーク共有機能の実装を行うというものである．さらに，WhiteDog Studio と呼ばれる GUI ツールを実装した．これは，GUI に選択可能なクラス/メソッド一覧を表示し，その選択情報からアスペクトコードを生成して AspectJ を起動しビルドを行うビルドツールである．この WhiteDog Studio には，ネットワーク共有機能の実装プロセスを省力化するにあたって，改良を要する点が残っている．本稿で提案する手法は，それを解決するものである．

本稿の構成は，以下の通りである．まず本稿で提案する手法の背景として，2 章で WhiteDog System の概要とそのオブジェクト共有手法および AOP について述べ，3 章で WhiteDog Studio の概要と改良を要する点を述べる．次に 4 章で AOP とメタデータアノテーションを組み合わせた手法について提案し，5 章でその手法により得られた効果について述べ，6 章にまとめを記述する．

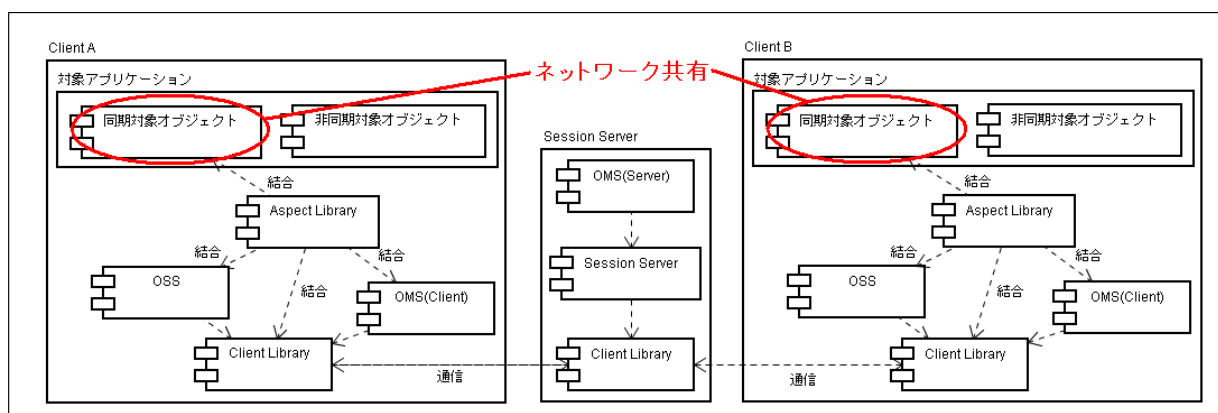

図 1: WhiteDog System のモジュール関係図

2 WhiteDog System

2.1 WhiteDog System の概要

WhiteDog System は，情報処理振興事業協会 (現独立行政法人情報処理推進機構) による平成 14 年度末踏ソフトウェア創造事業で採択されたプロジェクトにおいて開発され，プロジェクトの終了後も開発が継続されているシステムである．紀プロジェクトマネージャグループの，“ XML 通信を基盤としたオブジェクト共有・オーナー管理システム ”[4] がそれである．

このプロジェクトでは，主に以下のモジュールを実装した．

- 汎用ネットワークセッションサーバ (Session Server)
- 通信処理，XML 文書解析処理を実装したライブラリ (Client Library)
- オブジェクト共有システム (Object Sharing System)
- オーナー管理システム (Owner Management System)
- 上記の機能を結合する際の汎用処理を実装したアスペクトライブラリ (Aspect Library)

Session Server は，クライアント間のメッセージ配信を中継するセッションサーバとして実装されている．また通信データフォーマットには XML を採用している．

Client Library は，Session Server との TCP/IP を用いた通信機能を実装したライブラリである．また XML 文書の解析処理も実装されている．Session Server においても，その基盤部分に Client Library が使用されている．

Object Sharing System(OSS 以下) は，アプリケーション内のオブジェクトを，ネットワークを介して共有させる機能を実装している．Client Library を用いて実装されており，クライアントだけで完結している．

Owner Management System(以下 OMS) は，同一セッションに参加しているクライアントの中から一台だけにオーナー権限を与え，その状態を管理するシステムである．オーナー権限を持ったクライアントが切断した際の，次のオーナーを決定する機能や，オーナー権限の移動機能を実装している．またこのオーナー権限を利用した機能として，メッセージの排他制御機能を実装している．この OMS は，Session Server 上での拡張サービスおよびそのサービスをクライアント上で利用するためのライブラリとして実装されている．

以上のモジュールは，標準の Java 言語のみで記述されている．これらモジュールの機能を，AOP を用いて対象アプリケーションに結合するためのライブラリが，Aspect Library である．

各モジュールの関係を図 1 に示す．

2.2 WhiteDog System のオブジェクト共有手法

WhiteDog System における同期処理は，特定のメソッドの実行を共有することで，対象オブジェクトの状態を同期させるというものである．これにより，

同じセッションに参加しているクライアント間で，対象オブジェクトを共有している状態を作り出すことができる．

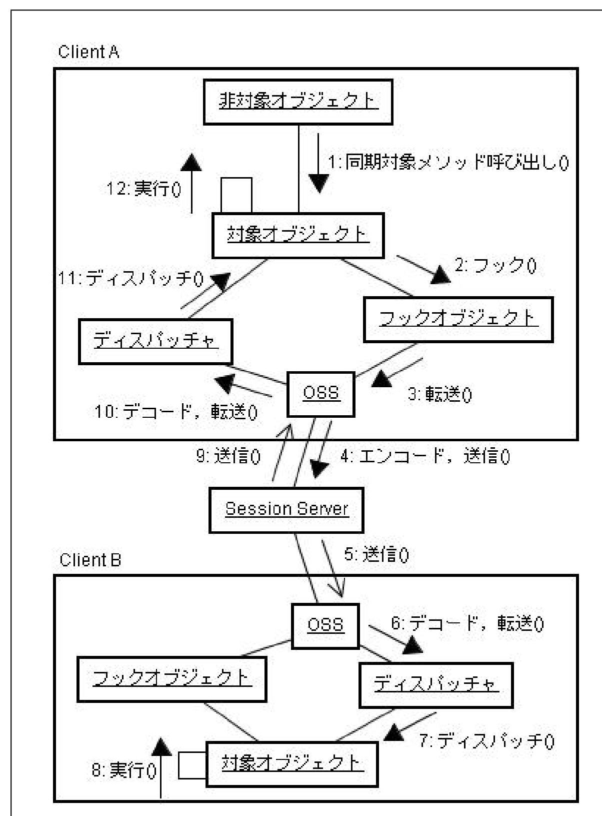

図 2: WhiteDog System におけるオブジェクト共有の仕組み

図 2 は，Client A で同期対象オブジェクトに対して行われたメソッド呼び出しの実行が，Client A，Client B 間で同期される手順を示している．

まず，同期対象オブジェクトの同期対象メソッドが呼ばれた際 (1) に，その呼び出しをフックオブジェクトがフックする (2)．フックオブジェクトはその実行を OSS へ転送 (3) し，OSS がエンコードを行い Session Server へ送信する (4)．

Session Server は受信したメッセージをそのままセッションへ参加しているクライアントへ送信する (5, 9)．

Session Server からメッセージを受け取った OSS は，メッセージをデコードしディスパッチャへ転送する (6, 10)．ディスパッチャは同期対象オブジェクトに対し，デコードされた情報を元にメソッドをディスパッチする (7, 11)．最終的にディスパッチされたメソッドが対象オブジェクトで実行される (8, 12)．

エンコード時には Java の持つシリアライズ機構を利用し，ディスパッチ時には，これも Java の持つリフレクション API を利用している．そのため，この同期処理は対象オブジェクトに依存しない．

2.3　WhiteDog System の AOP

Aspect Libary では，ネットワーク共有機能の実装に必要なアスペクトコードの汎用部分を実装している．以下のような機能が AbstractAspect アスペクトで提供されている．

- Session Server への接続の開始に関するメソッド
- 同期対象オブジェクトの登録に関するメソッド
- 同期対象メソッドへ挿入するフックコード

利用者は，この AbstractAspect を継承し，対象アプリケーションに依存する部分を記述する．実際に，簡単なペイントアプリケーションに対してネットワーク共有機能を挿入するするアスペクトを，図 3 に示す．

```
1  package jp.takao.whiteboard;
2
3  import jp.whitedog.aspect.AbstractAspect;
4
5  public aspect WhiteBoardAspect
6    extends AbstractAspect
7  {
8    private String sessionName
9      = "object." + getClass().getName();
10
11   after():                                        ①
12     execution(Canvas.new(..))
13   {
14     registerObject(sessionName
15       , thisJoinPoint.getTarget());
16   }
17
18   after(): execution(WhiteBoardFrame.new(..)){    ②
19     connect("localhost", 12539);
20   }
21
22   public pointcut concernedMethodsExecution():    ③
23     execution(void Canvas.drawLine(..))
24      || execution(void Canvas.fillOval(..))
25      || execution(void Canvas.fillRect(..))
26      || execution(void Canvas.drawString(..))
27     ;
28 }
```

図 3: ペイントアプリケーションに対するアスペクト

図 3 の①部分が同期対象オブジェクトの登録に関するコードである．ここでは，同期対象としている Canvas クラスのコンストラクタ実行後に，作成されたオブジェクトを同期対象として登録している．登録時の引数として，登録するオブジェクト自身と，セッションを識別する文字列を渡している．

②部分が Session Server への接続を開始するコードである．WhiteBoardFrame クラスが作成された後

に，接続を開始している．接続時のパラメータとして，ホスト名に”localhost”，ポートに”12539”を指定している．

③部分が，同期対象メソッドを指定するコードである．concernedMethodExecution ポイントカットの定義として，同期対象となるメソッドを列挙している．このポイントカットは抽象ポイントカットであり，AbstractAspect 内にこのポイントカットに対するアドバイスとしてメソッドの同期に必要な処理が実装されている．

3 WhiteDog System の拡張

3.1 WhiteDog Studio

WhiteDog System では，AOP を応用し，ネットワーク共有機能を対象アプリケーションへ挿入するというアプローチを提案した．これは，節 2.2 に示したオブジェクト共有手法に基づいてネットワーク共有機能を実装し，AOP を用いてそれを対象アプリケーションに結合するというものである．これにより，アプリケーション毎にネットワーク共有機能を個別に開発することなく，実装することが可能となった．

WhiteDog System を利用する際には，AspectJ 言語を習得し，アプリケーション毎にアスペクトコードを記述する必要がある．そのアスペクトコードを可能な限り削減するため，抽象ポイントカットと抽象アスペクトを利用し，アスペクトコードの汎用部分を基底アスペクトとして用意した．これにより，記述する必要のあるアスペクトコードは，図 3 に示したように，数十行にまで縮めることができた．

平成 14 年度未踏ソフトウェア創造事業後，WhiteDog System の発展として，WhiteDog Studio(以降, WhiteDog Studio 0.9.4) を開発した．これは，記述する必要のあるアスペクトコードを排除し，AspectJ 言語の学習コスト，アスペクトコードの記述コストを不要にするものである．その実現のために，WhiteDog Studio 0.9.4 には，GUI 操作によりアスペクトコードを生成する手法 [5] が実装されている．これにより，GUI 操作のみでネットワーク共有機能を対象アプリケーションに結合することが可能となった．

図 4 に，WhiteDog Studio 0.9.4 にペイントアプリケーションを読み込んだ直後の画面を示す．

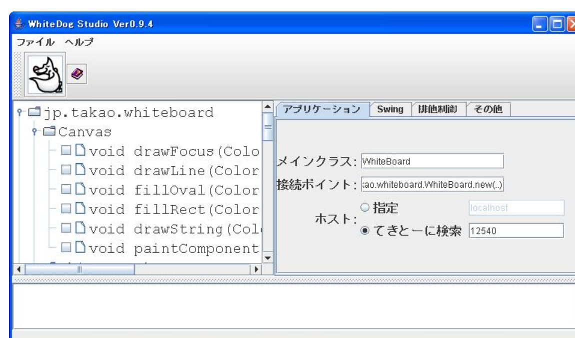

図 4: ペイントアプリケーションを読み込んだ直後の WhiteDog Studio 0.9.4

この画面上で同期させるメソッドをクリックし，実行ボタンを押すと，図 3 に相当するアスペクトコードが生成される．画面左上，犬のアイコンが描かれている部分が実行ボタンである．また，WhiteDog Studio 0.9.4 では，ビルド処理も実装した．アスペクトコードを生成した後，AspectJ コンパイラを起動してビルドを行い，出力されたオブジェクトと Client Library 等のライブラリを結合し，新たな jar ファイルを生成するというものである．これにより，従来手動で定義していたビルドファイルが不要になった．

WhiteDog Studio 0.9.4 を利用する際の，ネットワーク共有機能実装プロセスを図 5 に示す．

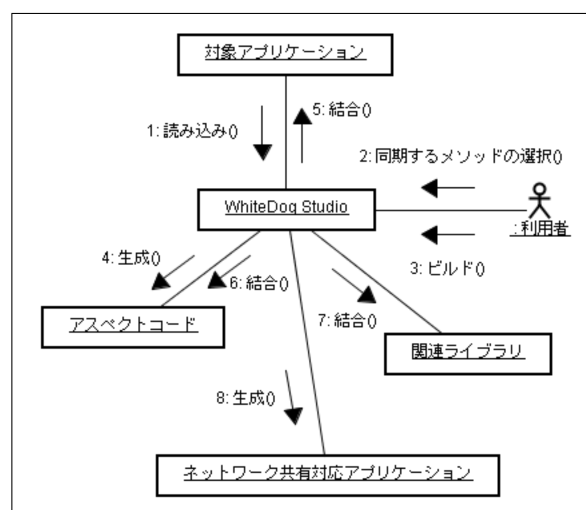

図 5: WhiteDog Studio 0.9.4 利用時の流れ

3.2 改良を要する点

WhiteDog Studio 0.9.4 を利用することにより，GUI による操作のみでネットワーク共有機能を実装することができる．しかし，メソッドを選択するか

どうかの判断を適切に行うためには，その実装に関する知識が必要となる．例えば図 4 で挙げた例では，drawLine，fillOval，fillRect，drawString の 4 つのメソッドを選択し，他の 2 つのメソッド (drawFocus, paintComponent) を選択しなければ，適切にキャンバスへの描画が同期する．その判断を行うためには，次の知識が必要である．

- drawFocus メソッドは一時的にフォーカス矩形を描画するだけで，実行されてもキャンバスの内容は変化しない
- drawLine メソッドは実際にキャンバスへ線を描画する
- fillOval メソッドは実際にキャンバスへ円を描画する
- fillRect メソッドは実際にキャンバスへ矩形を描画する
- drawString メソッドは実際にキャンバスへ文字列を描画する
- paintComponent はキャンバスのディスプレイへの描画時に呼び出されるメソッドで，実行されてもキャンバスの内容は変化しない

これらの知識はアプリケーションの開発者が持っているものであるため，そのアプリケーションを開発した本人でなければ，適切なメソッドを選択することは難しい．また，開発するプログラムが大規模な場合，開発者がすべてを覚えているわけではないので，メソッドが選択に適するかどうかの知識を得るために，開発者自身も元のソースコードを参照しなければならない可能性も考えられる．

そこでこの点を改良するため，Java5 で導入されたメタデータアノテーションを用いた問題解決方法を提案する．

4 メタデータアノテーションを使用した手法

手法を提案するに先立ち，メタデータアノテーションについて述べる．メタデータアノテーション自体は本稿で提案するものではなく，Java5 の仕様として提供されているものである．

4.1 メタデータアノテーション

メタデータアノテーション (以下，アノテーション) は，Java5 で追加された言語拡張機能である [6]. Java5 のドキュメント内では，注釈とも記述されるが，本稿ではアノテーションと表記する．アノテーションは，クラスやメソッドといったプログラムの要素に，メタデータを付加する手法である．同様の手法には，JDK に標準で添付されている javadoc コマンド用の javadoc タグや，それを拡張した XDoclet [7] といった手法がある．これらの手法がソースコードレベルでメタデータを付加するものであるのに対し，アノテーションでは，クラスファイルおよび実行時においてもメタデータを保持する．

保持レベルは，アノテーション定義時に指定する java.lang.annotation.RetentionPolicy 列挙型で表現され，ソースレベル (RetentionPolicy.SOURCE)，クラスレベル (RetentionPolicy.CLASS)，実行時 (RetentionPolicy.RUNTIME) の 3 つがある．それぞれ.java ファイル，.class ファイル，JavaVM によるロード後に対応しており，後になるほど保持期間は長くなる．保持レベルを指示するには，Retention アノテーションを用いる．

アノテーションでは，アノテーションの対象となる要素を指定することができる．それは java.lang.annotation.ElementType 列挙型で表現され，型 (ElementType.TYPE)，コンストラクタ (ElementType.CONSTRUCTOR)，メソッド (ElementType.METHOD) などがある．対象を指定する際には，Target アノテーションを用いる．

また，アノテーションでは，アノテーションの型付けやデフォルト値の導入，取得インターフェースのリフレクション API への統合も大きな特徴である．これらの特徴によって，アノテーションを利用した場合，リフレクション API により得られる型付けされた情報を付加することができる．

4.2 アノテーションを使用したアスペクトコード生成手法

節 3.2 で挙げた点を改良するため，WhiteDog Studio に，アノテーションを用いた手法を導入した．この手法は，アプリケーションのロード時に，クラスやメソッドに付加されたアノテーションを検索し，検出したアノテーションから得た情報を GUI に反映させるものである．これにより，WhiteDog Studio の利

用者は開発者から提示された情報を知ることができる．また，アノテーションが反映された GUI の状態はアスペクトコード生成時に参照されるため，開発者から提示されたアノテーションは直接生成されるアスペクトコードに影響を与える．以降，この手法を導入した WhiteDog Studio を，WhiteDog Studio 0.9.5 と呼ぶ．

まず，共有に適したメソッドを提示するために導入したアノテーションの定義を図 6 に示す．

```
1 @Retention(RetentionPolicy.RUNTIME)
2 @Target(ElementType.METHOD)
3 public @interface WhiteDogMethod{}
```

図 6: WhiteDogMethod アノテーション

この WhiteDogMethod アノテーションの保持レベルには，RetentionPolicy.RUNTIME を指定している．WhiteDog Studio 0.9.5 ではクラスローダを介して実際にクラスをロードし，リフレクション API 経由でこのアノテーション情報を取得する．また，WhiteDogMethod アノテーションは同期対象メソッドを指定するものなので，ターゲットは ElementType.METHOD を指定している．

図 7 に，同期対象メソッドを持つクラスの説明を提示するアノテーションの定義を示す．

```
1 @Retention(RetentionPolicy.RUNTIME)
2 @Target({ElementType.TYPE
3   , ElementType.CONSTRUCTOR
4   , ElementType.METHOD})
5 public @interface WhiteDogDescription{
6   String value();
7 }
```

図 7: WhiteDogDescription アノテーション

保持レベルは WhiteDogMethod アノテーションと同様，RetentionPolicy.RUNTIME を指定している．ターゲットは ElementType.TYPE，ElementType.CONSTRUCTOR，ElementType.METHOD を指定している．

このアノテーションは，クラスに対する説明を提示するものであり，value に渡された文字列は，WhiteDog Studio 0.9.5 の GUI 上で，クラス名の前に表示される．また，このアノテーションが提示されたクラス名の前にはチェックボックスが表示され，クリックしてチェック状態を変更すると，そのクラスが持つ同期対象メソッド (WhiteDogMethod が提示されているもの) 全てが同じ状態になる．つまり，そのクラスを同期対象とするかどうかを一括して変更できる．

WhiteDog Studio では，上記以外に，接続する Session Server のホスト名，Session Server への接続を開始するタイミングとなるメソッド，メインクラスなども指定できる．これら 3 つの情報を提示する手段として，WhiteDogConnection アノテーション，WhiteDogConnectionStartPoint アノテーション，WhiteDogMainClass アノテーションを導入した．

WhiteDogConnection アノテーションの定義を図 8 に示す．

```
1 @Retention(RetentionPolicy.RUNTIME)
2 public @interface WhiteDogConnection{
3   String host();
4 }
```

図 8: WhiteDogConnection アノテーション

このアノテーションでは，ホスト名を指定するプロパティとして，String host(); を宣言している．

WhiteDogConnectionStartPoint アノテーションの定義を図 9 に示す．

```
1 @Retention(RetentionPolicy.RUNTIME)
2 @Target({ElementType.CONSTRUCTOR
3   , ElementType.METHOD})
4 public @interface WhiteDogConnectionStartPoint
5 {}
```

図 9: WhiteDogConnectionStartPoint アノテーション

このアノテーションでは，対象として，コンストラクタとメソッドを指定している．このアノテーションが付加されたコンストラクタあるいはメソッドが実行されると，Session Server への接続が開始される．

WhiteDogMainClass アノテーションの定義を，図 10 に示す．

```
1 @Retention(RetentionPolicy.RUNTIME)
2 @Target(ElementType.TYPE)
3 public @interface WhiteDogMainClass{}
```

図 10: WhiteDogMainClass アノテーション

このアノテーションは，アプリケーションのメインクラスを指定するものである．そのため，対象には ElementType.TYPE を指定している．WhiteDog Studio 0.9.5 は，jar ファイルの生成時に，このアノ

テーションが指定されたクラスを MANIFEST ファイルの Main-Class に出力する．

5 効果

5.1 ビルドプロセスの自動化

ペイントアプリケーションの Canvas クラスに対して節 4.2 のアノテーションを使用したサンプルコードを，図 11 に示す．

```
01 @WhiteDogConnection(host = "127.0.0.1")
02 @WhiteDogDescription("キャンバス部分")
03 public class Canvas extends JComponent{
04   ...
05   public void paintComponent(Graphics g)
06   {...}
07
08   @WhiteDogMethod
09   public void fillOval(Color aColor
10     , int anX, int aY
11     , int aWidth, int aHeight)
12   {...}
13   @WhiteDogMethod
14   public void fillRect(Color aColor
15     , int anX, int aY
16     , int aWidth, int aHeight)
17   {...}
18   @WhiteDogMethod
19   public void drawLine(Color aColor
20     , int x1, int y1, int x2, int y2
21     , int aLineWidth)
22   {...}
23   @WhiteDogMethod
24   public void drawString(
25     Color aColor, Font aFont
26     , String aString, Rectangle aRect)
27   {...}
28   ...
29 }
```

図 11: 導入されたアノテーションの使用例

図 11 では，Canvas クラスに対してアノテーションを付加し，アドレス 127.0.0.1 で動作している Session Server へ接続すること (@WhiteDogConnection)，説明として“ キャンバス部分 ”と表示すること (@WhiteDogDescription)，fillOval メソッド，fillRect メソッド，drawLine メソッド，drawString メソッドを同期対象とすること (@WhiteDogMethod) を提示している．この Canvas クラスを含むペイントアプリケーションの jar ファイルを，WhiteDog Studio 0.9.5 に読み込んだ直後の状態を図 12 に示す．

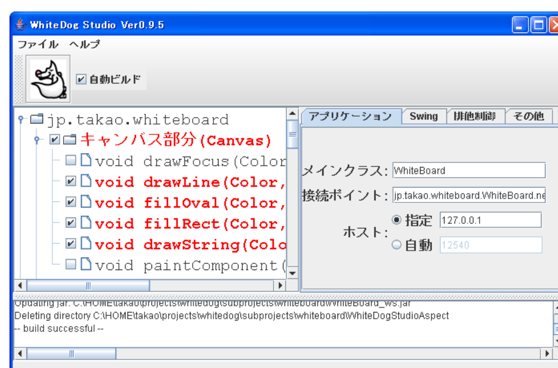

図 12: ペイントアプリケーションを読み込んだ直後の WhiteDog Studio 0.9.5

WhiteDog Studio 0.9.5 は，アノテーションを含むアプリケーションを読み込む際に，以下の処理を行う．

1. WhiteDogDescription が付加されたクラスにチェックボックスを表示する
2. WhiteDogDescription で提示された説明をクラス名の前に追加し，強調表示する
3. WhiteDogMethod が付加されたメソッドを強調表示する
4. WhiteDogMethod が付加されたメソッドをあらかじめ選択状態にする
5. WhiteDogConnection，WhiteDogConnection-StartPoint，WhiteDogMainClass により設定情報を取得する
6. 画面上部のチェックボックス“ 自動ビルド ”がチェックされ，かつ読み込まれたアプリケーションが，WhiteDogMethod アノテーションが付加されたメソッドを少なくとも 1 つ持つ場合，自動でビルドを行う

1〜4 により，WhiteDog Studio 0.9.5 の利用者は，同期対象としてどのメソッドが適切であるか，およびそれらのメソッドを同期対象とすることで何が同期されるかを知ることができる．

5 に示すアノテーションが無い場合，デフォルト値を検出して表示するが，その手法は WhiteDog Studio 0.9.4 のものと同一である．

4〜6 により，アプリケーションを読み込むだけで，そのアプリケーションにネットワーク共有機能を挿入することが可能となる．

生成されたネットワーク共有機能対応のペイントアプリケーションの実行画面を図 13 に示す．

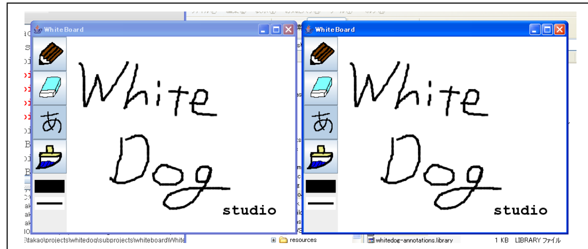

図 13: 生成されたペイントアプリケーション

5.2 同期対象のカスタマイズ

ビルドプロセスの自動化による省力化効果は高いが，アプリケーション内に同期対象オブジェクトが複数存在する場合，利用者が同期させるもの，させないものを選択できることも必要である．ペイントアプリケーションの例において同期できるものは，キャンバスに対する描画の他に，ツールボタンの選択状態，描画色，描画に使用するペンの太さなどがある．図 14 に，ツールボタンの選択状態を変更するメソッドに WhiteDogMethod アノテーションを，そのメソッドを持つ ToolButtonModel クラスに WhiteDogDescription アノテーションを付加したソースコードを示す．

```
01 public class WhiteBoardFrame extends JFrame{
02    public WhiteBoardFrame(){
03      ...
04      @WhiteDogDescription("ツール選択部分")
05      public class ToolButtonModel extends
06        JToggleButton.ToggleButtonModel{
07        ...
08        @WhiteDogMethod
09        public void setSelected(
10          boolean aSelected){
11          ...
12        }
13      }
14    }
15 }
```

図 14: ツールボタン選択状態の同期の提示

この図 14 に示したアノテーションと図 11 に示したアノテーションの両方が提示された場合，WhiteDog Studio 0.9.5 でアプリケーションを読み込むと，図 15 に示す画面が表示される．

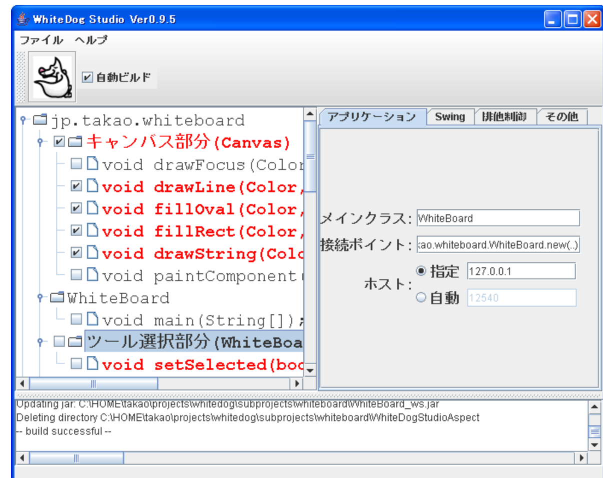

図 15: ツールボタンの選択状態変更メソッドも提示した例

利用者は，ツールボタンの選択状態を同期させたくない場合，“ ツール選択部分 ”のチェックを外すか，その setSelected メソッドのチェックを外してビルドを行えばよい．図 15 は，“ ツール選択部分 ”のチェックを外した状態である．この場合，生成されたアプリケーションでは，ツールボタンの選択状態は同期されなくなる．このような，同期方法をカスタマイズする場合においても，実装に関する知識は不要である．

setSelected メソッドがチェックされていない状態でビルドを行い，生成されたペイントアプリケーションの実行画面を図 16 に示す．

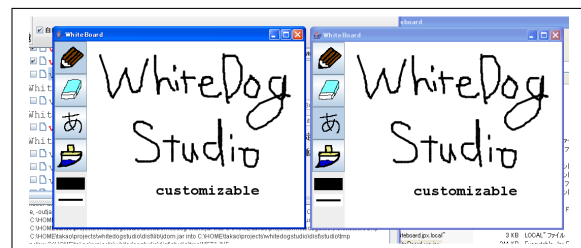

図 16: 同期対象をカスタマイズして生成されたペイントアプリケーション

図 16 では，図 13 と異なり，ツールボタンの選択状態は同期されていない．そのため，左側のペイントアプリケーションのキャンバスでドラッグを行うとペン描画ツールによる描画が行われ，右側だと文字描画ツールによる文字描画範囲の選択が行われる．

5.3 アノテーションを使用した効果

本稿で提案した手法を導入する前の WhiteDog Studio 0.9.4 では，図 17 に示すように，利用者に対して同期対象となるメソッドの実装に関する知識が要求されていた．

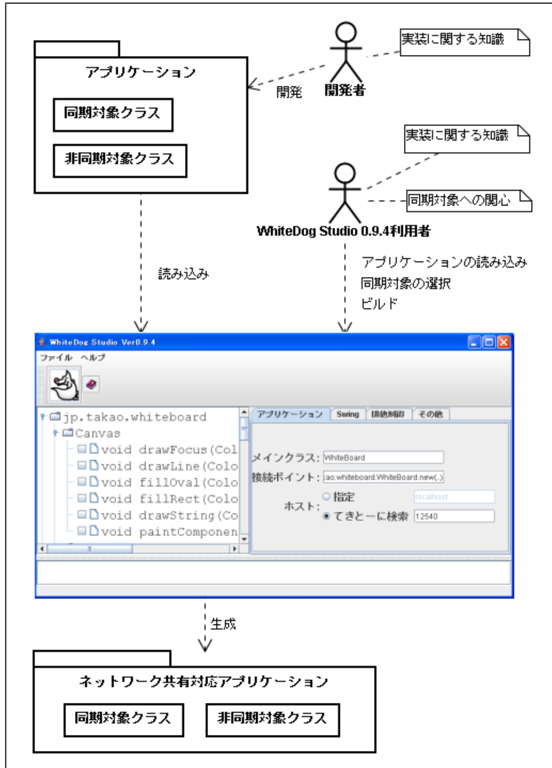

図 17: アノテーション導入前 (WhiteDog Studio 0.9.4)

本稿で提案した手法を導入することにより，図 17 で示されている，WhiteDog Studio 0.9.4 利用者に要求されていた実装に関する知識を不要にすることができた．加えて自動ビルドを行うことで，利用者が対象アプリケーションに対してネットワーク共有機能を実装する際のプロセスが省力化された．その様子を図 18 に示す．

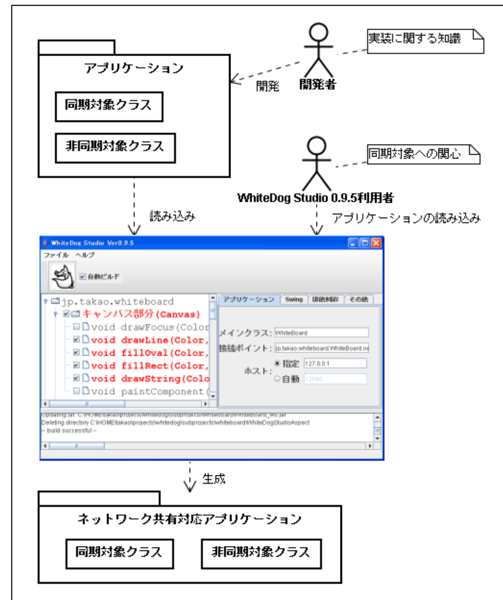

図 18: アノテーション導入後 (WhiteDog Studio 0.9.5)

さらに，利用者が同期対象をカスタマイズすることも可能とした．その場合でも，利用者には実装に関する知識は必要とされない．その様子を図 19 に示す．

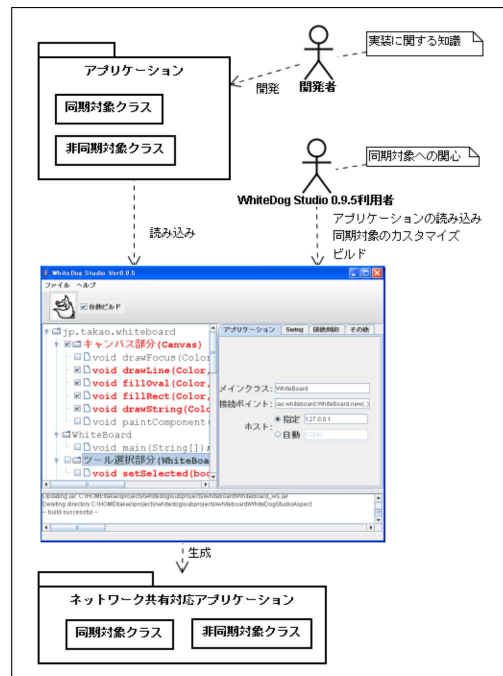

図 19: 利用者によるカスタマイズ (WhiteDog Studio 0.9.5)

6 おわりに

従来の WhiteDog Studio 0.9.4 では，利用する際に対象アプリケーションの開発者と同等の知識が必要とされていた．その理由は，WhiteDog Studio 0.9.4 においてメソッドを選択する際に，特定のメソッドが共有に適しているかどうか，その実装に関する知識が無いと判断することが困難なためである．この点を改良するために，本稿ではアノテーションを用いて開発者から必要な情報を提示する手法を導入した．これにより，WhiteDog Studio の利用者に求められていた対象アプリケーションの実装に関する知識が不要となり，ネットワーク共有機能の実装プロセスの自動化，および同期対象のカスタマイズが可能となった．

WhiteDog System では，その開発を通じて AOP を用いた機能実装プロセスの省力化を試みている．さらにその試みを進めるため，以下のことを課題として考えている．

- 対象機能の拡大
- 適用例の拡大
- 同期対象クラスおよびメソッドの選択支援
- ビルドプロセスへの組み込み

対象機能の拡大および適用例の拡大は，本稿と WhiteDog System，WhiteDog Studio で提案しているアプローチの有用性をさらに向上させるために必要である．

同期対象クラスおよびメソッドの選択支援は，省力化自体をさらに進めるために必要である．これは，WhiteDog Studio において，選択されていない，或いは WhiteDogMethod アノテーションが付加されていないメソッドの中から，同期対象となり得るメソッドの候補を検出することなどが考えられる．

ビルドプロセスへの組み込みとは，WhiteDog Studio で行っている処理を，対象アプリケーションのビルド処理に組み込むものである．具体的には，ant[8] タスクを実装し，それを使ったビルドファイルの例を作成することが挙げられる．

AOP では，横断的な関心事をアスペクトへモジュール化し，そのアスペクトを対象モジュールへ挿入するというソフトウェア構成手法を用いる．モジュール化されるコードの内容は従来のオブジェクト指向で記述されおり，アスペクト言語で記述されるのはポイントカットの指定部分およびアドバイスの定義部分においてである．WhiteDog Studio では，対象アプリケーションの構造や開発者からの提示，および利用者による GUI 操作によって得た情報からアスペクトコードの生成が可能であり，利用者がそれを記述するコストを省けることを示した．今後 GUI によるアスペクトコード生成手法およびアノテーションの導入によるその自動化をさらに発展させていくことは，アスペクト指向プログラミングの利用コストを削減することはもとより，ソフトウェア開発全体に対する生産性の向上においても重要な意義があると考える．

参考文献

[1] Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Lopes, Jean-Marc Loingtier and John Irwin, Aspect-Oriented programming. In *Proc. ECOOP '97*, 1997, pp. 220-242.

[2] G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm and G. Griswold, An Overview of AspectJ, In *Proc. ECOOP '2001*, 2001, pp. 327-352.

[3] The AspectJ project at Eclipse.org http://eclipse.org/aspectj/.

[4] 中口孝雄, 和田健之介, 熊坂妥仁, XML 通信を基盤としたオブジェクト共有・オーナー管理システム, 平成 14 年未踏ソフトウェア創造事業開発成果論文, 2002.

[5] 中口 孝雄, 廣瀬 誠, 山縣 敬一, ネットワーク共有機能を実装するための GUI を用いたアスペクトコード生成手法, 第 49 回システム制御情報学会研究発表講演会講演論文集, 2005, pp. 373-374.

[6] 注釈 http://java.sun.com/j2se/1.5.0/ja/docs/ja/guide /language/annotations.html.

[7] XDoclet: Attribute-Oriented Programming http://xdoclet.sourceforge.net/xdoclet/index.html.

[8] Apache Ant http://ant.apache.org/.