アスペクト指向プログラミングと Dependency Injection の融合

石川 零[†] 千葉 滋[†]

本稿はプログラムを構成するコンポーネントの内部を書き換えずにコンポーネントの結合を変更できるプログラムの設計手法として，glue コードを用いた設計を提案する．glue コードを用いた設計はアスペクト指向プログラミングに基づいているが，アスペクトをコンポーネントの結合の記述のみに用いるという特徴がある．この結合を記述するアスペクトが glue コードである．さらに従来のアスペクト指向システムを用いて glue コードを記述することは可能だが，我々はより glue コードに適したアスペクトを提供するアスペクト指向システムとして GluonJ を提案する．GluonJ が提供する glue コードは従来のアスペクトの機能に Dependency Injection の機能を取り込んだものである．

Aspect-Oriented Programming Meets Dependency Injection

Rei Ishikawa[†] and Shigeru Chiba[†]

This paper proposes programming design using a glue code. If a program is designed with a glue code, we can change the connection among components in the program without any changes of their implementations. A glue code is a new kind of aspects which is specialized for connecting components. This paper also proposes GluonJ, which is a new aspect-oriented system for glue codes. It provides integrated language constructs for dependency injection and aspect-oriented programming. GluonJ is more suitable for describing glue codes than existing systems.

1. はじめに

プログラムは一度完成した後に機能の追加や変更が必要になることが多い．そのためプログラムを後から簡単に変更できるように設計することは重要である．コンポーネントを用いた開発はプログラムを後から簡単に変更できるよう開発するための手法の 1 つである．コンポーネントとは関連する機能を 1 つにまとめた部品のことで，コンポーネントを用いた開発とはコンポーネントを組み合わせてプログラムを作るという開発手法である．

コンポーネントを組み合わせてプログラムを開発するときには，コンポーネントの内部を変えずにコンポーネントの組み合わせを変更できるようプログラムを設計するべきである．コンポーネントの組み合わせの変更は，プログラムを後から変更するときに必要となる．もしコンポーネントの組み合わせを変えるたびにコンポーネントの内部を変更しなければならないとすると，そのコンポーネントの再利用性は高いとは言えない．

我々はコンポーネントの内部を変更せずにコンポーネントの組み合わせを変えることができるプログラムの設計手法として，glue コードを用いた設計を提案する．glue コードを用いた設計はアスペクト指向プログラミングに基づいているが，アスペクトをコンポーネントの結合の記述のみに用いるという特徴がある．このコンポーネントの結合を記述するアスペクトが glue コードである．従来のアスペクト指向システムを用いて glue コードを記述することは可能だが，我々は glue コードにより適したアスペクトを提供するアスペクト指向システムとして GluonJ を提案する．GluonJ が提供する glue コードは従来のアスペクトの機能に Dependency Injection の機能を取り込んだものである．

以下 2 章では glue コードを用いた設計を提案し，3 章で GluonJ について示す．4 章では GluonJ と既存技術との比較を述べ，5 章では関連研究を述べる．最後に 6 章で本稿をまとめる．

† 東京工業大学 情報理工学研究科 数理・計算科学専攻
Department of Mathematical and Computing Sciences, Tokyo Institute of Technology

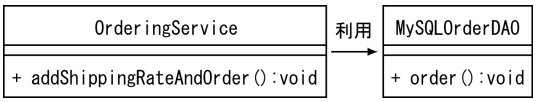

図 1 注文発注プログラムのクラス図
Fig. 1 A class diagram for the book store application

2. glue コードを用いたプログラムの設計

プログラムを構成するコンポーネントの組み合わせの変更は，コンポーネントの内部を変えずにできるようにするべきである．一般にプログラムは完成後に機能の変更，追加などが必要になることが多い．プログラムを構成するコンポーネントの組み合わせの変更は，このような変更をおこなうときに必要となる．もしコンポーネントの組み合わせを変えるたびにコンポーネントの内部を変更しなければならないとすると，そのコンポーネントの再利用性は高いとは言えない．

コンポーネントを変更せずにコンポーネントの組み合わせを変えられるようにし，再利用性を高める設計手法として，glue コードを用いた設計を提案する．glue コードを用いて設計したプログラムを構成するコンポーネントの再利用性は高い．

2.1 プログラムの変更例

複数のコンポーネントから構成されるプログラムと，コンポーネントの組み合わせの変更が必要となるプログラムの変更例を，以下の 2 つのコンポーネントから構成されるプログラムを用いて示す．このプログラムはオンライン書店プログラムの一部で，利用者の注文をデータベースへ送信するプログラムである．

このプログラムは OrderingService コンポーネントと MySQLOrderDAO コンポーネントから構成される．OrderingService コンポーネントの addShippingRateAndOrder メソッドは利用者が購入しようとしている本の総額を調べ，総額が一定以下ならば総額に送料を足して発注管理データベースに注文を送信する．注文の送信には MySQLOrderDAO コンポーネントの order メソッドを用いる．

このプログラムに対する変更として，以下の 2 つの変更を考える．

- 変更 1：データベースシステムの変更と新たな機能の利用
  この変更は注文発注プログラムが利用するデータベースシステムを別のデータベースシステムに置き換える場合を想定している．また新たに利用するデータベースシステムがデータベースアクセスの最適化アルゴリズムにヒントを与えられる機能を提供しており，その機能を利用することを想定する．新たに利用するデータベースシステムにアクセスするためのコンポーネントは OracleOrderDAO である．最適化アルゴリズムにヒントを与えるには OracleOrderDAO コンポーネントの sendHint メソッドを呼ぶ．
- 変更 2：安全性の強化
  この変更はプログラムの安全性を強化するために，プログラムを実行するユーザにデータベースへのアクセス権があるかを調べる機構を追加する場合を想定している．アクセス権の検査はデータベースをアクセスする処理の直前で，SecurityManager コンポーネントの check メソッドを用いて行う．SecurityManager コンポーネントを生成するには SecurityManager クラスのファクトリメソッドを用いる．

これら 2 つの変更はコンポーネントの組み合わせを変えることで実現できる．変更 1 を実現するには，OrderingService コンポーネントと結びついた MySQLOrderDAO コンポーネントを OracleOrderDAO に置き換えればよい．また変更 2 を実現するには，OrderingService コンポーネントに SecurityManager コンポーネントを新たに結びつければよい．

2.2 オブジェクト指向による設計とそれによる問題点

オブジェクト指向を用いたプログラムの設計では，コンポーネント同士を結合するためのコードをコンポーネントの内部に記述する．その結果，結合を変更するには結合するためのコードが記述されたコンポーネントの変更が必要になり，その再利用性は低下する．このことを以下のオブジェクト指向を用いて設計した注文発注プログラムの例を用いて具体的に示す．

```
public class OrderingService {
  private MySQLOrderDAO dao;
  public OrderingService() {
    dao = new MySQLOrderDAO();
  }
  public void addShippingRateAndOrder (
      User user, Book[] books) {
    int total = 0;
    for (int i = 0; i < books.length; i++)
      total += books[i].getPrice();
    if (total < 1500)
      total += getShippingRate(user);
    dao.order(user, books, total);
  }
}
public class MySQLOrderDAO {
  public void order (User user,
```

```
        Book[] books, int total) { ... }
}
```

オブジェクト指向を用いて設計した OrderingService コンポーネントの内部には OrderingService コンポーネントと MySQLOrderDAO コンポーネントとを結びつけるコードが存在する．結びつけるコードは 3 箇所あり，1 つは MySQLOrderDAO コンポーネントを指す MySQLOrderDAO 型の dao フィールドであり，1 つは MySQLOrderDAO のコンストラクタを呼び出すコードである．もう 1 つは addShippingRateAndOrder メソッドの実行時に dao フィールドの指すコンポーネントの order メソッドを呼び出すコードである．

このオブジェクト指向を用いて設計された OrderingService コンポーネントの再利用性は低い．なぜならプログラムを構成するコンポーネントの組み合わせを変える場合，OrderingService コンポーネントを変更せず再利用することができないからである．OrderingService コンポーネントの内部には，OrderingService コンポーネントと MySQLOrderDAO コンポーネントを結びつけるコードが含まれており，組み合わせを変えるにはこれらのコードを変更しなければならない．例えば 2.1 節で示した変更 1 をこのプログラムに対して行うには， OrderingService コンポーネントの内部に対して以下の 2 つの変更を加える必要がある．

- dao フィールドの型を OracleOrderDAO に変更し，MySQLOrderDAO のコンストラクタ呼び出しを OracleOrderDAO のコンストラクタ呼び出しに変更
- addShippingRateAndOrder メソッドに含まれる，order メソッドの呼び出しの直前に dao フィールドの指す OracleOrderDAO コンポーネントの sendHint メソッドの呼び出しを追加

また変更 2 を行うには OrderingService コンポーネントに対して SecurityManager コンポーネントを新たに組み合わせればよいが，それには OrderingService コンポーネントの内部に対して以下の 3 つの変更を加える必要がある．

- OrderingService コンポーネントに SecurityManager コンポーネントを指すフィールドを追加
- OrderingService コンポーネントのコンストラクタに SecurityManager コンポーネントのコンストラクタ呼び出しを追加
- addShippingRateAndOrder メソッドの冒頭に，SecurityManager コンポーネントのメソッド呼び出しを追加

コンポーネントの組み合わせ方を変えるときにコンポーネントを変更せず利用できないことは，プログラムを構成するコンポーネントが多い場合に特に問題となる．例えば OrderingService コンポーネントの他に MySQLOrderDAO コンポーネントを利用するコンポーネントが複数種類存在したとき，変更 1 を行うには MySQLOrderDAO コンポーネントのコンストラクタ呼び出しの変更と sendHint メソッド呼び出しの追加を，MySQLOrderDAO コンポーネントを利用する全てのコンポーネントに対して行わなければならない．変更 2 でも同様に，上に挙げた OrderingService コンポーネントに対する変更と同じ変更を，MySQLOrderDAO コンポーネントを利用する全てのコンポーネントに対して行わなければならない．

2.3 提案：glue コードを用いた設計

我々はコンポーネントの内部を変更せずにコンポーネントの組み合わせ方を変えることができるプログラムの設計手法として，glue コードを用いた設計を提案する．glue コードを用いた設計ではオブジェクト指向を用いた設計とは異なり，コンポーネントを結びつけるコードをコンポーネントの外側に分離する．この分離したコードを記述するためのモジュールが glue コードである．一般にコンポーネントを結びつけるためには以下の 3 通りのコードが用いられるが，glue コードはこれらのコード全てをコンポーネントから分離して記述する機能を持つ．

- フィールド宣言
  関連する別のコンポーネントを参照するフィールド．オブジェクト指向を用いた注文発注プログラムの例で OrderingService コンポーネントの dao フィールドは MySQLOrderDAO コンポーネントを指す．
- サブコンポーネントの生成と代入
  関連するコンポーネントを生成し，それをフィールドに代入するコード．コンポーネントの生成方法にはコンストラクタを用いた方法やファクトリメソッドを用いた方法がある．注文発注プログラムの例で OrderingService コンポーネントは，コンストラクタ内で MySQLOrderDAO 型のコンポーネントを生成し，dao フィールドへ代入している．
- サブコンポーネントのメソッド呼び出し
  別のコンポーネントのメソッドを呼び出すコード．注文発注プログラムの例で OrderingService コンポーネントは addShippingRateAndOrder メソッドの最後で MySQLOrderDAO コンポーネントの order メソッドを呼び出している．

glue コードはアスペクト指向プログラミングにおけ

るアスペクトの一種である．アスペクトとは横断的関心事を一つにまとめたモジュールのことである．横断的関心事とは，オブジェクト指向プログラミングなど従来のプログラミング手法でモジュール化することが難しい処理のことである．一般にアスペクトは横断的関心事を実装し，それと実行するコンポーネントを結びつける働きを持つ．例えば代表的なアスペクト指向言語である AspectJ[1),7)] のアスペクトは，アドバイスという言語機構を用いて横断的関心事を実装し，ポイントカットという言語機構を用いて横断的関心事とコンポーネントを結びつける．しかしこの glue コードを用いた設計では，アスペクトの利用を横断的関心事とそれを実行するコンポーネントとの結びつけのみに限定する．横断的関心事の実装にはアスペクトではなく通常のオブジェクトを用いる．従来のアスペクトを結びつけの機能に特化させたものが glue コードである．

3. GluonJ：glue コードを用いた設計を支援するアスペクト指向システム

我々は glue コードを用いて再利用性の高いコンポーネントを設計することを支援する Java 言語用のアスペクト指向システムとして GluonJ を開発した．AspectJ のアスペクトを glue コードの代用に使うことも可能だが，GluonJ を用いることでコンポーネントの結びつきをより直感的に記述することが可能になる．

3.1 GluonJ を用いたプログラムの例

2 章で例として用いた注文発注プログラムを GluonJ を使って記述し，それを用いて GluonJ が提供する glue コードの機能とその使い方を示す．以下は GluonJ を用いて設計した注文発注プログラムである．GluonJ を用いた注文発注プログラムは， MySQLOrderDAO コンポーネントと OrderingService コンポーネント，それらの組み合わせを記述した glue コードの 3 つから構成される．以下は MySQLOrderDAO コンポーネントと OrderingService コンポーネントである．

```
public class OrderingService {
  private User userdata;
  private Book[] booksdata;
  private int total;
  public void addShippingRateAndOrder (
      User user, Book[] books) {
    total = 0;
    for (int i = 0; i < books.length; i++)
      total += books[i].getPrice();
    if (total < 1500)
      total += getShippingRate(user);
    userdata = user;
    booksdata = books;
  }
}
public class MySQLOrderDAO
    extends OrderDAO {
  public void order (User user,
      Book[] books, int total) { ... }
}
```

オブジェクト指向による設計との違いは，コンポーネント同士を結びつけるコードがコンポーネント内に含まれていないことである．OrderingService コンポーネントと MySQLOrderDAO コンポーネントと結びつけるコードは以下の glue コードに記述する．GluonJ の glue コードは XML を用いて記述する．

```
<glue>
  <reference>
    MySQLOrderDAO OrderingService.
        aspect = new MySQLOrderDAO();
  </reference>
  <behavior>
    <pointcut>
      execution(void OrderingService.
          addShippingRateAndOrder(..))
    </pointcut>
    <after>
      MySQLOrderDAO.aspectOf(this).order(
          userdata,
          booksdata, total);
    </after>
  </behavior>
</glue>
```

3.1.1 <reference> タグ

<reference> タグは，コンポーネントを結合するために用いられる 3 通りのコードのうち，コンポーネントを指すフィールドと，コンポーネントを生成し代入するコードを分離して記述するためのタグである．このタグは既存のコンポーネントにフィールドを新たに追加する機能と，コンポーネントの生成時に別のコンポーネントを生成してそれをフィールドに代入する機能を持つ．コンポーネントの生成は Java コードを用いて明示的におこなう．上の <reference> タグの記述は，OrderingService コンポーネントに新しくフィールドを追加することと， OrderingService コンポーネントの生成時にコンストラクタを呼んで MySQLOrderDAO コンポーネントを生成し，それを追加したフィールドに代入することを意味する．

上の glue コードに含まれる <reference> タグの記述は MySQLOrderDAO コンポーネントを指すフィールドとして OrderingService クラスの aspect というフィールドを指定している．これは OrderingService ク

ラスの既存のフィールドではなく，匿名フィールドという特別なフィールドである．コンポーネントを代入するフィールドとしてあるコンポーネントの匿名フィールドを指定すると，そのコンポーネントには新たにフィールドが追加される．そして追加されたフィールドにコンポーネントが代入される．追加されるフィールドの名前はシステムによって自動的に決められる．匿名フィールドの値を取得するには aspectOf メソッドを用いる．aspectOf メソッドについては 3.1.2 節で詳しく説明する．

匿名フィールドではなく特定の名前を持つフィールドを追加して，そのフィールドにコンポーネントを代入することも可能である．この機能は AspectJ のインタータイプ宣言と似た機能である．これらの機能の違いについては 4 章で詳しく述べる．次の記述は匿名フィールドの代わりに link フィールドを追加して，そのフィールドにサブコンポーネントを代入する例である．

```
<reference>
  OrderDAO OrderingService.link =
      new MySQLOrderDAO();
</reference>
```

匿名フィールドの追加と特定の名前を持つフィールドの追加との違いは，既存のフィールドとの名前の衝突を意識する必要があるかどうかである．特定の名前を持つフィールドを追加する場合，そのフィールドには既存のフィールドと異なる名前をつけなければならない．なぜなら追加したフィールドと既存のフィールドとの区別ができなくなるからである．一方，匿名フィールドを追加する場合は名前の衝突を意識する必要はない．匿名フィールドには既存のフィールドの名前と衝突しない名前が，システムにより自動的に与えられるからである．この衝突を回避する機能は AspectJ における private フィールドのインタータイプ宣言の持つ機能と同様である．

一方，<reference> タグでは，生成したコンポーネントを既存のフィールドに代入することも可能である．生成したコンポーネントのメソッドをコンポーネントから直接呼びたいときには，この既存のフィールドを用いた方法が有効である．OrderingService クラスに OrderDAO 型の dao フィールドが用意されていた場合には，次の記述により MySQLOrderDAO コンポーネントを dao フィールドに代入できる．なお dao フィールドの型は定義してもしなくてもよい．

```
<reference>
  OrderingService.dao =
      new MySQLOrderDAO();
</reference>
```

3.1.2 <behavior> タグ

<behavior> タグはメソッド呼び出しをコンポーネントから分離して記述するためのタグである．この機能は AspectJ のアドバイスに似ている．これらの違いは 4 章で詳しく示す．<pointcut> タグ内の記述は，コンポーネント内のどこでメソッドを呼び出すかを指定する．<pointcut> タグ内の記述に用いる言語は AspectJ の提供するポイントカット言語とほぼ同じである．

<after> タグは <pointcut> タグが指定した箇所の直後に呼び出すブコンポーネントのメソッドを指定する．実行する処理は Java コードで指定する．このタグの中では任意の Java コードに加えて，aspectOf メソッドを利用できる．これは匿名フィールドが指すコンポーネントを取得するためのメソッドで，以下のような仕様をしている．

static $Aspect$ $Aspect$.aspectOf($Target$ target);

aspectOf メソッドの戻り値は，引数の $Target$ コンポーネントの匿名フィールドが指す $Aspect$ コンポーネントである．この aspectOf メソッドは，$Aspect$ 型の匿名フィールドの追加にともなって $Aspect$ クラスへ追加される．

上の <behavior> タグの記述は OrderingService コンポーネントの addShippingRateAndOrder メソッドの実行をポイントカットを用いて指定し，その処理の直後に，OrderingService コンポーネントの匿名フィールドが指す MySQLOrderDAO コンポーネントの order メソッドを呼び出すことを意味している．

また <behavior> タグの中では <after> タグの代わりに <before> タグ，<around> タグも利用できる．<before> タグは <pointcut> タグが指定した箇所の直前に呼び出すメソッドを指定し，<around> タグは <pointcut> タグが指定した箇所で実行される処理の代わりにおこなう処理を指定する．これらのタグの中で記述できるコードは <after> タグの中で記述できるコードと同じである．

3.2 GluonJ による結合の分離

2.3 節で，glue コードはコンポーネントを結合するために用いられる 3 通りのコードを分離する機能を持つものだと定義した．ここでは GluonJ の提供する glue コードがこの条件を満たしていることを示す．

- フィールド宣言
  <reference> タグを用いることで，フィールドをコンポーネントから分離して記述することができる．<reference> タグは別のコンポーネントを指

すためのフィールドをコンポーネントへ新たに追加できる．追加できるフィールドには特定の名前を持つフィールドと匿名フィールドがある．匿名フィールドを用いれば，既存フィールドとの名前の衝突を意識する必要はなくなる．

- サブコンポーネントの生成と代入
  <reference> タグを用いることで，コンポーネントを生成し代入するコードをコンポーネント内部から分離して記述することができる．コンポーネントの生成方法は Java コードを用いて明示的に記述できる．Java コードを用いるため複雑な生成も可能である．生成に複雑な処理を必要とする場合には，glue コードとは別に Java でファクトリメソッドを定義し，それを <reference> タグ内で利用することが可能である．
  また <reference> タグでは，生成したコンポーネントを代入するフィールドとして，既存のフィールドと新たに追加されたフィールドのどちらも指定することができる．
- サブコンポーネントのメソッド呼び出し
  <behavior> タグを用いることで，コンポーネントの処理の呼び出しをコンポーネント内部から分離して記述することができる．<behavior> タグは，処理を呼び出す箇所の指定にはポイントカット記述を，呼び出す処理の指定には Java コードを用いる．匿名フィールドが指すコンポーネントの処理を呼ぶときには，aspectOf メソッドを用いてコンポーネントを取得する．
  なおポイントカット記述では指定ができない箇所で別のコンポーネントの処理を呼び出したい場合には，行アノテーションというコーディング技術が利用できる．これについては 3.4.1 節で詳しく説明する．

3.3 glue コードを用いて再利用性が向上したコンポーネントの例

glue コードを用いてプログラムを設計することでコンポーネントの再利用性が向上することを，GluonJ を用いて設計した注文発注プログラムの例を用いて具体的に示す．2.1 節では 2 つの変更例を挙げたが，どちらの変更も GluonJ を用いて設計したプログラムに対しては glue コードを変更するだけで実現できる．まず変更 1 を実現するには，glue コードを次のように書き換えればよい．

```
<glue>
  <reference>
    OracleOrderDAO OrderingService.
        aspect = new OracleOrderDAO();
  </reference>
  <!-- フィールドと生成方法の変更 -->
  <behavior>
    <pointcut>
      execution(void OrderingService.
          addShippingRateAndOrder(..))
    </pointcut>
    <after>
      OracleOrderDAO.aspectOf(this).
          sendHint(this);
      <!-- sendHint 呼び出しの追加 -->
      OracleOrderDAO.aspectOf(this).order(
          this.userdata,
          this.booksdata, this.total);
    </after>
  </behavior>
</glue>
```

変更した glue コードの箇所は 2 つある．1 つは <reference> タグにある MySQLOrderDAO のフィールドの追加とコンストラクタ呼び出しを OracleOrderDAO に書き換えたことである．もう 1 つは <behavior> タグに sendHint メソッドの呼び出しを追加したことである．

また変更 2 を実現するには glue コードを次のように書き換えればよい．

```
<glue>
  <reference>
    MySQLOrderDAO OrderingService.
        aspect = new MySQLOrderDAO();
    SecurityService OrderingService.
        aspect = SecurityService.
                    factory(this.id);
  </reference>
  <!-- フィールドと生成のコードを追加 -->
  <behavior>
    <pointcut>
      execution(void OrderingService.
          addShippingRateAndOrder(..))
    </pointcut>
    <after>
      MySQLOrderDAO.aspectOf(this).order(
          this.userdata,
          this.booksdata, this.total);
    </after>
  </behavior>
  <behavior>
    <pointcut>
      execution(void OrderingService.
          addShippingRateAndOrder(..))
    </pointcut>
    <before>
      SecurityService.aspectOf(this).
          check(this.id);
    </before>
```

```
  </behavior>
  <!-- check メソッドの呼び出しを追加 -->
</glue>
```

変更した glue コードの箇所は 2 つある．まず <reference> タグを用いて SecurityManager コンポーネントのフィールドを追加し，ファクトリメソッドを呼び出して SecurityManager コンポーネントを生成，代入するコードを追加する．次に <behavior> タグを用いて check メソッドの呼び出しを記述する．

3.4 高度な結合の記述

3.4.1 行アノテーション

GluonJ は別のコンポーネントのメソッドを呼び出す箇所を指定するために <pointcut> タグを用いるが，<pointcut> タグでは指定できないプログラム中の箇所も存在する．例えば GluonJ のポイントカット記述言語を用いて，特定の for 文の直前を指定することはできない．

このような場合は行アノテーションと我々が呼ぶコーディング技術を用いて，コンポーネント内にある，サブコンポーネントの処理を呼び出したい箇所に目印をつければよい．行アノテーションとはコード内に記述する注釈のことである．例えば先に挙げた注文発注プログラムで，行アノテーションを用いて MySQLOrderDAO コンポーネントの order メソッドの呼び出し箇所に目印をつけるには，次のようにして OrderingService コンポーネントの addShippingRateAndOrder メソッドに行アノテーションを記述する．

```
public void addShippingRateAndOrder (
    User user, Book[] books) {
  ...
  DAO.send();
}
```

DAO.send() が行アノテーションである．行アノテーションは空の static メソッドの呼び出しとして記述する．行アノテーションに用いる static メソッドとクラスは開発者が用意する．このクラスやメソッドの名前は開発者が自由に決められる．

行アノテーションによって目印のついた箇所は <pointcut> タグを用いて指定できる．これには static メソッドの呼び出しを指定するポイントカット記述を用いる．次の <pointcut> タグの記述は上の行アノテーションを指定する．

```
<pointcut>
  call(static DAO.send())
</pointcut>
```

なおこの行アノテーションは GluonJ にのみ有効な手法ではなく，他のアスペクト指向言語を用いたときにも有効な手法である．

3.4.2 特定の実行フロー内で有効な匿名フィールド

<reference> タグは，特定の実行フローの中でのみ有効な匿名フィールドを追加する機能も持つ．この機能は AspectJ が提供する percflow 指定子と同等の機能である．特定の実行フローの中でのみ有効な匿名フィールドを追加するには，次のように匿名フィールドの追加先のクラスとして Cflow クラスを指定する．Cflow クラスは実行フローを表す特別なクラスである．

```
<reference>
  OrderDAO Cflow(
    call(void OrderingService.
      addShippingRateAndOrder(..))).
        aspect = new MySQLOrderDAO();
</reference>
```

Cflow クラスの後ろに書かれた括弧の中の記述は，どの実行フローの中で匿名フィールドを有効にするかを指定する．この括弧の中の記述には <pointcut> タグの内容を記述するときに用いるポイントカットの記述言語を用いる．この <reference> タグの記述は，addShippingRateAndOrder メソッドの呼び出しの間だけ有効になる OrderDAO 型の匿名フィールドを追加する．

このように追加された匿名フィールドの値を取得するには，aspectOf を次のように利用する．

```
<after>
  OrderDAO.aspectOf(thisCflow).order();
</after>
```

aspectOf メソッドの引数に渡されている thisCflow は，現在の実行フローを表す Cflow 型のオブジェクトである．aspectOf の引数に thisCflow オブジェクトを渡すことで，現在の実行フローで有効な匿名フィールドの値を取得できる．thisCflow は <after> タグ内で利用できる変数である．

3.4.3 クラスのグループへの匿名フィールドの追加

<reference> タグは，クラスのグループに対して匿名フィールドを追加することもできる．クラスのグループに対して追加された匿名フィールドは，グループ内のクラスをそれぞれインスタンス化したオブジェクトのグループごとに異なる値を持つ．この匿名フィールドの構造はオブジェクトのグループをキーとする多重ハッシュテーブルのようなものである．この機能は Association Aspects[11] が提供する perobjects 指定子と同等の機能である．次の <reference> タグの記述は OrderingService クラスと Session クラスのペアに MySQLOrderDAO 型の匿名フィールドを追加する．

```
<reference>
  OrderDAO OrderingService.
      aspect(Session) =
```

```
        new MySQLOrderDAO(this, args);
</reference>
```

この <reference> タグの記述により OrderingService クラスに追加された匿名フィールドは，複数の OrderDAO コンポーネントを指すことができる．この匿名フィールドが指すコンポーネントは，Session コンポーネントをキーとして識別される．キーの型は aspect の後ろの括弧に記述された型により決まる．この匿名フィールドが指すコンポーネントを取得するには，aspectOf を次のように利用する．

```
<after>
  OrderDAO.aspectOf(service, session)
          .order();
</after>
```

この aspectOf メソッドは，service 変数の匿名フィールドが指す複数の OrderDAO コンポーネントのうち，session 変数をキーとする OrderDAO コンポーネントを返す．service 変数の型は OrderingService クラス，session 変数の型は Session クラスである．もし service 変数の匿名フィールドに session 変数がキーとなる OrderDAO コンポーネントが存在しないときには，aspectOf メソッド内でこれをキーとする OrderDAO コンポーネントが新たに生成される．この生成には <reference> タグ内に書かれた MySQLOrderDAO コンストラクタが用いられる．

MySQLOrderDAO コンストラクタへの引数になっている this と args は特別な変数である．this 変数は aspectOf メソッドの 1 番目の引数を表す変数である．args 変数は aspectOf の残りの引数を表す Object 型の配列である．

4. 既存技術によるコンポーネントの組み合わせと問題点

コンポーネントの再利用性を高めることが可能なプログラムの設計手法として，これまではデザインパターンや Dependency Injection，既存のアスペクト指向言語が用いられてきた．しかしこれらの手法にはコンポーネントの結びつけを十分に分離できない，また結びつけの記述が間接的になるといった問題点がある．

4.1 デザインパターン

デザインパターンを用いてプログラムを設計する場合，プログラムの変更を予測し，それに備えて設計しない限りコンポーネントを変更せず再利用することはできない．以下はデザインパターンを用いて，データベースシステムの変更に備えて設計した注文発注プログラムの例である．このプログラムの設計にはデザインパターンの 1 つであるファクトリパターンを用いている．

```
public class OrderingService {
  private OrderDAO dao;
  public OrderingService() {
    dao = OrderDAOFactory.factory();
  }
  public void addShippingRateAndOrder (
      User user, Book[] books) {
    int total = 0;
    for (int i = 0; i < books.length; i++)
      total += books[i].getPrice();
    if (total < 1500)
      total += getShippingRate(user);
    dao.order(user, books, total);
  }
}
public class OrderDAOFactory {
  public OrderDAO factory() {
      return new MySQLOrderDAO();
  }
}
```

2.2 節で示したオブジェクト指向を単純に用いたプログラムとの違いは 2 つある．1 つは MySQLOrderDAO コンポーネントの生成を OrderingService コンポーネントから OrderDAOFactory に分離したことであり，もう 1 つは dao フィールドの型を OrderDAO インタフェース型にしたことである．このプログラムで用いるデータベースシステムを変更しても OrderingService コンポーネントの内部を変更する必要はない．変更が必要な箇所がファクトリメソッドの内部のコンストラクタ呼び出しに限られるためである．

しかしこのプログラムに対して設計時に準備しなかった変更を行う場合，コンポーネントを変更せず再利用することはできない．例えば 2.1 節で示した変更 1 を行うには，上記の変更に加えてさらに sendHint メソッドの呼び出しを追加しなければならないが，この変更を行うには OrderingService コンポーネント内部の変更が必要になる．変更が必要になる原因は，このプログラムが変更 1 が起こることを想定して設計されていないからである．もし起こりうる全ての変更をあらかじめ想定できるならば，デザインパターンを用いてどんな変更が起こってもコンポーネントを変更せず再利用できるプログラムを設計することができるが，全ての変更を設計時に想定することは困難である．

4.2 Dependency Injection

Dependency Injection はコンポーネントの再利用性を高める言語機構だが，コンポーネントの結びつけに用いられる 3 通りのコードのうち，コンポーネントを生成してフィールドに代入するコードしか分離する

ことはできない．生成と代入を行うコードは DI コンテナの設定ファイルを用いて記述する．DI コンテナとは Dependency Injection を提供するシステムである．以下は DI コンテナの 1 つである Spring Framework[6] を用いて記述した設定ファイルの例である．

```
<beans>
  <bean name="service"
        class="OrderingService">
    <property name="dao">
      <ref bean="dao"/>
    </property>
  </bean>
  <bean name="dao"
        class="MySQLOrderDAO" />
</beans>
```

開発者は DI コンテナのファクトリメソッドを呼ぶことで，この設定ファイルに従って MySQLOrderDAO コンポーネントが dao フィールドに代入された OrderingService コンポーネントを生成することができる．

GluonJ の機能のうち，<reference> タグの生成と代入を分離する機能は Dependency Injection の持つ機能と同等である．Dependency Injection との違いは，<reference> タグ内のスコープが，フィールドがあるコンポーネントのスコープと同じであることである．<reference> タグ内のスコープはコンポーネントの代入先フィールドのあるスコープと同じであるため，private フィールドへ直接代入することが可能である．

4.3 アスペクト指向言語

コンポーネントの再利用性を高めるために既存のアスペクト指向言語を用いる方法は 2 通りあるが，それぞれに問題点がある．

4.3.1 アスペクトをコンポーネントに用いる設計

まず結びつけるコンポーネントのペアのうち，利用される側のコンポーネントをアスペクトとして設計する手法がある．この手法を用いて注文発注プログラムを設計すると，MySQLOrderDAO コンポーネントがアスペクトになる．以下は AspectJ を用いて記述した MySQLOrderDAO アスペクトである．なお OrderingService コンポーネントは GluonJ を用いて設計したものと同様に Java クラスを用いて実装する．

```
aspect MySQLOrderDAO {
  after(OrderingService s) :
      execution(void OrderingService.
      addShippingRateAndOrder(..))
      && this(s) {
    User user = s.userdata;
    Book[] books = s.booksdata;
    int total = s.total;
    // 注文処理の実行
  }
}
```

アスペクトはポイントカットやアドバイスといった要素から構成される．ポイントカットとは実行中のプログラムの特定の時点を指定するための記述で，メソッドを実行する時点やフィールドをアクセスする時点を指定することができる．ポイントカットにより指定される時点をジョインポイントと呼ぶ．プログラムの実行がポイントカットが指定する時点に差し掛かったら，そのポイントカットと結びついたアドバイスが実行される．アドバイスとはポイントカットの指定する時点で呼ばれる処理のことで，ポイントカットと組み合わせて用いる．

この AspectJ を用いたプログラムでは，MySQLOrderDAO アスペクトのポイントカットはオブジェクト指向を用いたプログラムで OrderingService コンポーネント内にあった MySQLOrderDAO コンポーネントの order メソッドの呼び出しに対応する．また MySQLOrderDAO アスペクトの after アドバイスは MySQLOrderDAO コンポーネントにあった order メソッドに対応する．このポイントカットとアドバイスにより、order メソッドの呼び出しは OrderingService コンポーネントから分離される．また MySQLOrderDAO アスペクトはプログラム中で暗黙のうちにコンストラクタが呼ばれて生成される．

AspectJ を用いたプログラムを構成するコンポーネントの組み合わせを変更しても，OrderingService コンポーネントは変更せず再利用することができる．なぜなら order メソッドの呼び出しや MySQLOrderDAO コンポーネントを指すフィールド宣言など，結合のためのコードが OrderingService コンポーネントから取り除かれているためである．例えば 2.1 節で示した変更 1 は MySQLOrderDAO アスペクトの代わりに OracleOrderDAO アスペクトを定義して用いることで実現できる．また変更 2 は新たに SecurityManager アスペクトを定義することで実現できる．

しかしアスペクトをコンポーネントに用いる手法では，コンポーネントを通常のクラスでなくアスペクトとして設計しなければならない．アスペクトとして設計すると様々なプログラミング上の制約を受ける．この制約はアスペクト指向言語により異なる．例えば AspectJ のアスペクトは，アスペクトを独自の文法を用いて記述しなければならない，アスペクトをコンストラクタを直接呼んで生成することはできない，抽象アスペクトを定義するときに別な抽象アスペクトを親アスペクトに利用できないといった制約を伴う．また通常のクラスとして設計された既存のコンポーネント

をアスペクトとして用いることもできない．一方 glue コードを用いた設計では，コンポーネントの開発に制約は伴わない．glue コードを用いた設計では全てのコンポーネントを通常のクラスとして設計するからである．

また AspectJ が開発者に与える制約の 1 つに，アスペクトを用いて設計したコンポーネントと別のコンポーネントの結びつけを自由に指定できないという制約がある．これはアスペクトとコンポーネントとの結びつけ方が指定子によって限定されているからである．AspectJ は issingleton，perthis といった指定子を提供している．AspectJ が提供する指定子がサポートしていない結合をおこなうのは困難である．例えば，あるフィールドに同じ値を持った複数の OrderingService コンポーネントに 1 つの MySQLOrderDAO アスペクトを結びつけることは難しい．

4.3.2 アスペクトを glue コードの代用に使う設計

また AspectJ のアスペクトを glue コードの代わりに使う設計手法もある．この手法を用いて設計した注文発注プログラムは，GluonJ の glue コードを用いた設計と同様に OrderingService コンポーネントと MySQLOrderDAO コンポーネント，それらを組み合わせる glue コードの 3 つから構成される．OrderingService コンポーネントと MySQLOrderDAO コンポーネントは GluonJ を用いて設計した場合の Java クラスと同じものを用いる．glue コードの代用となる Glue アスペクトは以下に示す．

```
public privireged aspect Glue {
  private MySQLOrderDAO
      OrderingService.dao =
      new MySQLOrderDAO();

  after(OrderingService s) :
      execution(void OrderingService.
      addShippingRateAndOrder(..)) &&
      this(s) {
    s.dao.order(s.userdata,
        s.booksdata, s.total);
  }
}
```

この Glue アスペクトの機能は，GluonJ を用いて設計した注文発注プログラムの glue コードの機能と同等である．Glue アスペクトは 1 つのインタータイプ宣言と 1 つのアドバイスから構成される．インタータイプ宣言とは既存のクラスに対してフィールドやメソッドを追加する機能である．Glue アスペクトのインタータイプ宣言は，GluonJ を用いて記述した glue コードの <reference> タグに対応する．またアドバイスは GluonJ を用いて記述した glue コードの <behavior> タグに対応する．

AspectJ のアスペクトの持つ機能は glue コードの代用に使うには不十分である．なぜなら AspectJ のアスペクトは，コンポーネントを既存のフィールドに代入するコードを直接分離するための機能を備えていないからである．AspectJ のアスペクトを使ってコンポーネントを既存のフィールドへ代入するには，アドバイスを用いて代入するコードを記述しなければならない．このとき代入のコードは長く煩雑になってしまう．例えば注文発注プログラムの例で，dao フィールドが既に OrderingService コンポーネントに存在しているとする．このとき MySQLOrderDAO コンポーネントを dao フィールドに代入するコードは以下のようにアドバイスを用いたものになる．

```
  after(OrderingService s) :
      execution(OrderingService.
      new(..)) && this(s) {
    s.dao = new MySQLOrderDAO();
  }
```

一方 GluonJ の glue コードでは，<reference> タグを用いて既存フィールドへ代入するコードを短く簡潔に記述することができる．上のアドバイスと同等の機能を持つ <reference> タグの記述は以下のようになる．

```
<reference>
  OrderingServoce.dao =
      new MySQLOrderDAO();
<reference>
```

<reference> タグはサブコンポーネントの生成と代入の分離を目的として設計された言語機構であるため，追加されたフィールドへも既存のフィールドへも同じようにコンポーネントを代入することができる．

また AspectJ のアスペクトを glue コードの代わりに使った場合，ジョインポイントの文脈情報を取得するための記述が煩雑になってしまう．ジョインポイントの文脈情報は，glue コードから他のコンポーネントのコンストラクタやメソッドを呼び出すときに必要となるため，文脈情報の取得が煩雑なことは問題である．本節の例では， Glue アスペクトのアドバイスは OrderingService コンポーネントの userdata フィールドや booksdata フィールドの指すコンポーネントを取得し，それを引数に渡して MySQLOrderDAO コンポーネントの order メソッドを呼び出している．これらの文脈情報を取得するためには OrderingService コンポーネントの参照を取得する必要があるが，そのためにはまずポイントカット変数 s を宣言する必要があ

る．そして this ポイントカットを用いて OrderingService コンポーネントを s に束縛するという手順を踏まなければならない．

一方 3.1 節で示した GluonJ の glue コードでは，Java コードの中で this 変数を使うことで OrderingService コンポーネントへの参照を取得できる．ポイントカット変数や this ポイントカットを使う必要はない．

結びつけの記述が間接的になる原因は，AspectJ のアスペクトが独立したコンポーネントであるためである．Glue アスペクトから見て OrderingService コンポーネントは異なるコンポーネントである．そのため OrderingService コンポーネントやそのメンバにアクセスするには，まず this ポイントカットなどを用いてコンポーネントへの参照を取得し，そしてポイントカット変数を通じて間接的にアクセスしなければならない．一方 GluonJ の glue コードは独立したコンポーネントではなく，他のコンポーネントの一部である．そのため <behavior> タグの Java コードは OrderingService コンポーネント内部のコードと同じスコープを持つので，メンバに直接アクセスすることができる．

また同様の問題として，ジョインポイントの文脈情報が private フィールドに保存されているとき，AspectJ のアスペクトはこの文脈情報へアクセスできないという問題がある．注文発注プログラムの例では OrderingService コンポーネントの private フィールドがこれにあたる．アクセスできない原因は，上で述べた問題の原因と同じく，AspectJ のアスペクトが独立したコンポーネントであることである．一方 GluonJ の glue コードは，ジョインポイントを含むコンポーネントの private フィールドにアクセスできる．これは GluonJ の glue コードがジョインポイントを含むコンポーネントの一部だからである．

なお glue コードに privileged アスペクトを用いれば，glue コードが横断的関心事と結びつくコンポーネントの private フィールドへアクセスできるようになるが，privileged アスペクトの利用はプログラムの情報隠蔽を無効化するという問題を起こす．AspectJ の提供する privileged アスペクトは全てのクラスの private メンバをアクセスすることが可能である．Glue アスペクトは privileged アスペクトとして宣言されているため OrderingService コンポーネントの private メンバへアクセスできる．しかし privileged アスペクトとして実装した glue コードは横断的関心事を実行するコンポーネント以外の private メンバにアクセスすることも可能になってしまう．例えば privileged アスペクトとして実装した Glue アスペクトは MySQLOrderDAO コンポーネントの private メンバへのアクセスも可能になる．これはプログラムの情報隠蔽を無効化してしまう．GluonJ の glue コードからアクセスできる private メンバは上の例では OrderingService の private メンバのみであり，プログラムの情報隠蔽を無効にすることはない．

5. 関 連 研 究

Spring Framework, Seasar2[12)] , NanoContainer[4)] などの DI コンテナはアスペクト指向プログラミングのための言語機構を持つ．しかしこれらのシステムは glue コードを用いた設計に利用するには機能が不十分である．例えばこれらのシステムはフィールド宣言を既存のコンポーネントに追加する機能を持たないため，フィールド宣言によるコンポーネントの結合を分離して記述することができない．

glue コードを用いた設計では，開発者は横断的関心事を実装するコンポーネントを通常のクラスとして設計できる．一方で AspectJ のアスペクトをコンポーネントに用いた設計では，開発者は様々な制約に従って横断的関心事を実装するコンポーネントを設計しなければならない．開発者に与える制約が AspectJ に比べて少ないアスペクト指向システムは存在するが，全く制約を与えないシステムはない．

例えば AspectWerkz[2)] や JBoss-AOP[5)] はアスペクトをクラスとして，アドバイスをメソッドとして記述させ，ポイントカットなどを XML ファイルやアノテーションを用いて記述させる．しかしアドバイスとして用いるメソッドの引数の型や個数，また投げる例外はシステムが定めたものに限定される．

CaesarJ[8)] はアスペクトに用いるコンポーネントと通常のコンポーネントを区別しない．また CaesarJ は豊富な機能を持つため，これを用いることで GluonJ を用いた場合にほぼ相当するプログラミングが可能になる．しかし CaesarJ のアスペクトは直接横断的関心事を実装するためのものなので，これを用いて記述されたコンポーネントの結びつけは AspectJ のアスペクトを用いたときと同様に煩雑になってしまう．

また AspectJ が与える制約の 1 つに，コンポーネントの結びつけ方が限定されるという制約がある．これを解決しようとするアスペクト指向システムは存在するが，結びつけの記述の自由度は glue コードを用いて設計したときほど高くない．Association Aspects[11)] や Eos[9)] , Eos-U[10)] というアスペクト指向言語は複数のコンポーネントに対してアスペクトを結びつける

指定子を用意する．特に Eos-U はアスペクトとして用いるコンポーネントと通常のコンポーネントを区別しないため GluonJ と近い．しかしこれらのシステムが提供する生成と結びつけの機能には制限がある．例えばある 1 つの OrderingService コンポーネントと結びつけた MySQLOrderDAO アスペクトを，その結びつきを保ったまま別の OrderingService コンポーネントと結びつけることはできない．

GluonJ を用いることで横断的関心事を実装するコンポーネントを通常のクラスとして実装でき，またコンポーネントの結びつけを Java コードで細かく指定することができるが，一方で GluonJ は，単純な生成や結びつけをおこなうときにもこのような指定を必要とする．単純な生成や結びつけで十分な場合には，AspectJ のように用意された指定子から選ばせる機構を持つアスペクト指向システムのほうが便利である．しかし，アスペクト指向の研究が進み，新たな利用例が見つかるにしたがって GluonJ の提供する細かな指定が必要になる機会は増えると我々は考えている．

6. ま と め

本稿はコンポーネントの内部を変えずにコンポーネントの組み合わせを変更できるプログラムの設計手法として glue コードを用いた設計を提案した．glue コードを用いた設計ではアスペクトをコンポーネントの結合の記述のみに用いる．従来のアスペクト指向言語を用いても glue コードの記述は可能だが，本稿はより glue コードに適したアスペクトを提供するアスペクト指向システムとして GluonJ を提案した．

我々は既に GluonJ を提案した論文[3] を発表したが，本稿はその論文の内容に対してより議論を深めたものである．具体的には glue コードの役割やコンポーネントの結びつけに利用されるコードの種類をより明確に定義している．

参 考 文 献

1) AspectJ Project. Aspectj. `http://eclipse.org/aspectj/`.
2) Jonas Boner and Alexandre Vasseur. Aspectwerkz 1.0. `http://aspectwerkz.codehaus.org/`.
3) Shigeru Chiba and Rei Ishikawa. Aspect-oriented programming beyond dependency injection. In *ECOOP' 05*, pages 121–143, 2005.
4) Codehaus. Nano Container. `http://www.nanocontainer.org/`.
5) JBoss Inc. Jboss aop 1.0.0 final. `http://www.jboss.org/`.
6) Rod Johnson and Juergen Hoeller. *Expert One-on-One J2EE Development without EJB*. Wrox, 2004.
7) Gregor Kiczales, Erik Hilsdale, Jim Hugunin, Mik Kersten, Jeffrey Palm, and William G. Griswold. An overview of aspectj. In *ECOOP' 01*, pages 327–353, 2001.
8) Mira Mezini and Klaus Ostermann. Conquering aspects with caesar. In *AOSD '03*, pages 90–99. ACM Press, 2003.
9) Hridesh Rajan and Kevin Sullivan. Eos: instance-level aspects for integrated system design. In *ESEC/FSE 2003*, pages 297–306. ACM Press, 2003.
10) Hridesh Rajan and KevinJ. Sullivan. Classpects: unifying aspect- and object-oriented language design. In *ICSE '05*, pages 59–68. ACM Press, 2005.
11) Kouhei Sakurai, Hidehiko Masuhara, Naoyasu Ubayashi, Saeko Matsuura, and Seiichi Kimoya. Association aspects. In *AOSD '04*, pages 16–25. ACM Press, Mar. 2004.
12) The Seasar Project. Seasar. `http://homepage3.nifty.com/seasar/`.