オンデマンドのアスペクト・ウィービングに基づくWebアプリケーション開発のすすめ

佐藤芳樹[†,†††] 立堀道昭[††] 千葉滋[†,†††]

Web アプリケーションは、クライアントの嗜好や動作環境にあわせて柔軟にカスタマイズされることが望ましい. 本稿では, 我々の開発した実行時にアスペクトと呼ばれるコードモジュールをプログラムに合成するツールである Wool を用いて, ユーザ毎に最適なアプリケーションプログラムを生成し, 必要に応じてその挙動を実効性能が上がるように変更する手法について述べる. プログラマは, 状況に応じた異なる処理を, Web アプリケーションプログラムにハードコードすることなく, 別にアスペクトとして柔軟に記述することができ, Wool を使って Web アプリケーションの稼働を止めずに合成することができる.

A Web Application based on On-demand Aspect Weaving

Yoshiki Sato,[†] Michiaki Tatsubori[††] and Shigeru Chiba[†]

It is desirable that a web application is customized for the preference of clients or the execution environment. This paper presents an implementation technique that makes web application change its behavior for efficiency on demand by generating programss specific to each user, using the tool named Wool we developed. Wool composes aspects with a running program. With Wool, programmers can describe the specific operations as the aspect separately from the program of a web application. Furthermore, they can compose this aspect with the web application with no downtime.

1. はじめに

近年, B2B, B2C の基盤として企業内外のアプリケーションを Web ブラウザと統合する Web アプリケーションの本格的な導入が進んでいる. 基幹アプリケーションと連携して業務処理を行うような Web アプリケーションには, セッション管理, トランザクション管理, 負荷分散, DB 処理, セキュリティなどの多くの機能が搭載されるため, その開発や保守のコストが非常に大きい.

Enterprise JavaBeans (EJB) はビジネスアプリケーションに必要なサーバ側の機能を EJB コンテナが肩代わりするが, 提供される機能は限定されていて, 新機能に対する拡張性が乏しい. EJB コンテナがサポートすべき機能は, J2EE (Java2 Enterprise Edition) によって, データベースアクセス, トランザクション管理などにあらかじめ規定されている. したがって, 例えば EJB コンテナに新たにレスポンスキャッシュやデータのプリフェッチ, アクセス制御機能を追加しようとした場合, コンテナを独自に拡張するか, アプリケーションプログラムにハードコードしなければならない.

キャッシュやプリフェッチのような機能は, 一般化できず状況によって最適なアルゴリズムが異なる. 例えば, 株取引に関するサイトの場合, 株式市場が開いている時間帯には情報が頻繁に更新されるためキャッシュが無駄になるし, 閉っている時間帯にはキャッシュが応答速度を向上させる. また, ネットオークションサイトでは, 膨大な出品物の中からユーザの嗜好に応じて, あらかじめ商品データベースからジャンルを絞って情報を取得した方が商品の検索や閲覧の応答が向上する.

しかしながら, クライアントの嗜好や動作環境に特化する Web アプリケーションの開発は難しく, 注意深くコーディングしなければオーバーヘッドが大きくなってしまう. キャッシュやプリフェッチのためのコー

† 東京工業大学大学院情報理工学研究科
Graduage School of Information Science and Engineering, Tokyo Institute of Technology
†† IBM 東京基礎研究所
IBM Tokyo Research Laboratory
††† 科学技術振興事業団, CREST
CREST, Japan Science and Technology Corp

ドは, 上手にモジュール化できないため, 状況に応じて切り替えるためには, アプリケーションプログラム中の様々な場所にその処理を記述しなければならなく, 見通しが悪く開発性・保守性が悪くなる. さらに, 実行中にそれらすべてが発火しているかどうかをチェックしなければならない.

本稿では, 我々の開発した, 必要に応じてアスペクトと呼ばれるコード群をプログラムに実行時合成するシステム Wool[7] を用いて, Web アプリケーションを状況にあわせて効率的にカスタマイズできるようにする方法について述べる. プログラマは Web アプリケーションの実効性能をあげるために, 状況に応じて特化するコード群をアスペクトに記述する. Wool は, 指定された条件が満たされるとアスペクトをプログラム中の適切な場所へ合成する. Wool によって特化されるコードは必要になるまで合成されないので, 状況に特化した処理の発火条件を無駄にチェックする必要がなく, そのオーバーヘッドを軽減できる. アスペクトによるプログラムの改変はアスペクト指向プログラミング [3] (AOP) ではウィーブ と呼ばれるため, Wool が実行時に必要に応じて適切な場所のコードを改変する事を特に オンデマンドのアスペクト・ウィービングと呼ぶ.

以下では, まず次章で, 既存の Web アプリケーションについて議論し, 続いて 3 章でオンデマンドのアスペクト・ウィービングを具体的な Wool 上のアスペクト例とともに説明する. 5 章で本論文をまとめる.

2. カスタマイズ可能な Web アプリケーション

状況に応じてカスタマイズ可能な Web アプリケーションは, 自身で状況を判断して動作を変更する. そのために必要な機能は, 大きく次の 4 つに分けて考えることができる.

状況の検査と記録 (モニタリング) 実行時間, メモリ使用量, 負荷, 外部環境などを取得して記録する.

動作状況の推定 (リーズニング) アプリケーションの動作状況を検査結果から推定する.

適切な動作方針の決定 (リゾルブ) 状況から動作をどう変更すべきかを判断し, 動作方針を決定する.

決定された方針の実践 (チューニング) 決定に従い, 実際にアプリケーションの動作を変更する.

本稿では, このうち, 最初と最後のモニタリングとチューニングの部分の開発に注目して議論する. 以降本章では, 自己をカスタマイズすることが可能な Web アプリケーションの実装を, Web のレスポンスキャッシュを例にとり, 従来の手法の問題点をまとめる.

2.1 レスポンスキャッシュ

的確なレスポンスキャッシュは Web アプリケーションの応答性をチューニングする. しかし, キャッシュした結果がよく再利用される場合は, 効果的であるが, そうでない場合, 逆にキャッシュの導入によるメモリ使用量の増加や応答速度の低下により, 実行効率が低下してしまうこともある. そこで, キャッシュの適用の有無や, キャッシュ・エントリの無効化のアルゴリズムなどを, 状況にあわせて調整できるように実装しなければならない. 例えば, 個人情報からキャッシュする場所を決めたり, 全体的にヒット率が高くないならキャッシュを控え目にしたり, 頻繁に無効化されるキャッシュは行なわない, さらにアクセスの集中する時間帯には積極的に行なう, などの戦略がある.

キャッシュ戦略を使い分ける場合, それに関わるコードはプログラム中に散らばり見通しが悪く保守性や再利用性が低下し, さらに, それはパフォーマンスを劣化させる. 例えば, Servlet で構築したショッピングサイトでショッピングカートの内容をキャッシュする場合を考える. カートに入れた商品を表示するページでは, Servlet 内で商品リストや合計金額を取得するメソッド getGoodsList() や getAmount() が呼び出され, そのメソッドにはカートの内容が変らない時に前に呼び出した結果を返すようなコードが記述される. また, カートの内容が変更された時に呼び出される updateCart() には, 関連するキャッシュを無効化するコードが記述される. このように, キャッシュのためのコードはアプリケーションロジックとは無関係にモジュール間に散らばってしまう. また, キャッシュするかどうかの判定に, 過去数時間の統計的なヒット率の算出や, 部分的にヒットする組み合わせの算出のような重い処理を伴う場合, ブラウザからのリクエストに対して常に無駄なオーバーヘッドを引き起こしてしまう.

3. オンデマンドのアスペクト・ウィービング

我々の開発した動的アスペクトウィーバである Wool を用いて, 必要に応じてアスペクトをウィーブすることで, 状況に応じて適切な場所へ適切な処理をアプリケーションプログラムに埋め込む手法 (図 1) を提案する. 我々のシステムでは, 従来, モジュール間に散らばり, 状況によって実行された方が効率的だが, 動的に切り替える事が困難であったキャッシュやプリフェッチのような処理をアスペクトとして記述する. 処理の内容は静的に決めなければならないが, 実行される場

所を実行時に決定できるためアプリケーションの動作を柔軟に変更することができる.

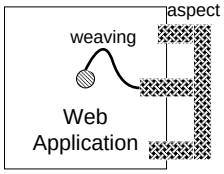

図 1 オンデマンドウィービングの模式図
Fig. 1 A diagram of on-demand weaving

3.1 アスペクト指向プログラミング

アスペクト指向プログラミング (AOP) では, キャッシュやプリフェッチのような非機能的な処理を単一モジュール化することができる. AOP は, モジュール間に散らばる処理を, その具体的な処理内容と実行される場所をそれぞれアドバイス, ポイントカット という形で表現することによって, データと共にアスペクトというモジュール単位に記述することができる.

```
 1: public abstract aspect ResponseCaching {
 2:   private HashMap cache = new HashMap();
 4:   private double hitRatio,threshold;
 5:   abstract pointcut op1();
 6:   abstract pointcut op2();
 7:   pointcut caching(String key) : op1() && args(key);
 8:   String around(String key): caching(key) {
 9:     if (hitRatio < threshold) return proceed();
10:     Key key = Key.createKey(key);
11:     String result = (String)cache.get(key);
12:     if (result == null) {
13:       String s = proceed(key);
14:       cache.put(key, s); return s;
15:     } else { return result; }}
16:   pointcut invalidating() : op2()
17:   before(): invalidating() {
18:     cache.clear(); }}
```

図 2 AspectJ を使ったキャッシュのためのアスペクト例 ☆
Fig. 2 An example of the caching aspect in AspectJ

図 2 に汎用的な Java 言語の AOP 拡張である AspectJ[2] で記述したレスポンスキャッシュのためのアスペクト例を示す. `ResponseCaching` アスペクトは, 抽象アスペクトでありキャッシュとその無効化のためのアドバイスを定義している. それぞれのアドバイスを実行する場所は, 具象アスペクトで抽象ポイントカット (`op1()`,`op2()`) をオーバーライドして決定される. 例えば, `ResponseCaching` アスペクトを継承した `StockCaching` アスペクトで `pointcut op1():execution(* StockMarket.get*())` や `pointcut op2():execution(* StockMarket.update*())` のようにポイントカットを記述する. また, 具象アスペクトでヒット率の閾値を設定し, ヒット率に応じてキャッシュするかどうかを決めることができる.

AspectJ はアスペクトとアプリケーションプログラムを静的に合成するため, `ResponseCaching` アスペクトの具象アスペクトを動的にアプリケーションプログラムにウィーブすることはできない. また, 具象アスペクトを動的に生成する事, すなわちキャッシュやその無効化を行う場所を実行時に設定することもできない. したがって, 静的にキャッシュやその無効化が行われる可能性のあるすべての場所を, `pointcut op1():execution(* *.*(..))` のように指定し, アドバイスの中で想定されるすべての条件の判定を行うようにアスペクトを記述しなければならない. もし, オーバーヘッドを避けるために, AOP を捨て, 各呼び出しに最適な条件文をハードコードした場合, 途端に保守性・再利用性が低下してしまう.

3.2 オンデマンドのアスペクト・ウィービング

我々は, ポイントカットを動的に決定しアスペクトを実行中のプログラムにウィーブすることで, アプリケーションがクライアントや動作環境に特化して振る舞うように変更する. アスペクトは, Wool が提供する API を利用して AspectJ と同じように記述される. プログラマは, 対象となるアプリケーションプログラム中で, ポイントカットを決定し, 自身にウィーブするようなコードを記述する.

接続クライアントからキャッシュする場所とキャッシュを行うかどうかの閾値を取得し, 適切なアスペクトを生成して自分自身にウィーブする Web アプリケーションプログラムのコード片を示す.

```
Client c = session.getAttribute("client")
double t = c.getThreshold();
WlAspect azpect = new ResponseCaching(t);
Pointcut cache = reasoning(azpect,c);
azpect.setPointcut(cache);
Wool wool = Wool.connect(...);
wool.weave(azpect);
```

このコードは, 例えば Servlet プログラムでは, doGet() や doPost() のようなアプリケーションの入り口で実行される. セッションオブジェクトから取得したクライアント情報から, キャッシュを有効にする閾値, キャッシュを行う場所を `Pointcut` オブジェクトとして取り出している. 次に, 閾値を渡してアスペクトを表す `WlAspect` オブジェクトを生成する. このアスペクトには静的にキャッシュやインバリデーションのためのアドバイスが記述されている. 最後に, 動的に決定されたポイントカットをアスペクトにセットし, `Wool` オブジェクトを使ってプログラムに合成して

いる.

3.3 Wool

Wool は, 動的にアスペクトをプログラムに合成する Java DAOP (dynamic AOP) システムである. Wool は実行時にプログラム変換を行ってアスペクトをプログラムに合成する. また, プログラマが Java 言語によりアスペクトを記述するための API を提供する. 合成したプログラムを動作中のアプリケーションに反映させるために, JDK1.4 の JPDA (Java Platform Debugger Architecture) の HotSwap 機能をを利用して変換後のプログラムをアプリケーションに反映させている. JDK1.4 の JPDA は, デバッグモードの性能劣化がたかだか 5%程度に抑えられている.

4. 関 連 研 究

実行時にアスペクトをプログラムに合成する DAOP システムは数多く提案されている. JAC[4] や Handiwrap[1] は, 考えられるプログラム中のすべての場所に検査コードを挿入する. それらの場所では, 合成したアスペクトが有効かどうかが常にチェックされる. 我々の予備実験では, この手法は無駄なコードが数多く埋め込まれるため, Wool により 26%程度アプリケーションの実行速度が向上されることが分かっている. したがって, Web アプリケーションのような長時間稼働のシステムには常にオーバーヘッドを生じさせる. JIT コンパイラで静的にフックを挿入する手法[5] は, オーバーヘッドを減少させると報告されているが, HotSpot VM のように JIT コンパイルが最初のメソッド呼び出しで起こるとは限らない VM には適応できない.

Wool でも利用している JPDA を使った手法として PROSE[6] が提案されている. PROSE は breakpoint をセットして, そこでデバッガがアドバイスを実行する. PROSE は Wool と同じようにオンデマンドにアスペクトを合成することが可能である. しかし, モニタリングやチューニングのコードを実行するためにコンテキストスイッチが頻繁に発生するため, そのオーバーヘッドは非常に大きく, Wool により 98%程度実行性能が向上される事が分かっている.

キャッシュサーバのキャッシュやプリフェッチ戦略をアスペクトに記述し, ダイナミックリンクするシステムも提案されている[8]. しかし, このシステムはアドバイスが実行されるポイントを静的に決定しているため, 散らばった処理を動的に切り替える事はできない.

5. ま と め

本稿では, オンデマンドなアスペクト・ウィービングによりクライアントの嗜好や動作環境に応じて, 実行効率のために自己を特化させる Web アプリケーションの実現手法について述べた. 我々の開発した動的アスペクトウィーバの Wool を用いれば, ポイントカットを動的に変更することができるため, モジュール間に散らばる非機能的な処理を効率的良く動的に切り替える事が可能になる. また, アスペクト指向によって, モジュール化された非機能的な処理の保守性・再利用性も向上する.

今後は, 本稿で例示したアスペクトを実装し, 稼働中の Web アプリケーションに生じるオーバーヘッドを実験によって確認する必要がある. 現在, Wool のプロトタイプ実装では, 様々なアスペクトを実アプリケーションに合成することができている.

参 考 文 献

1) Baker, J. and Hsieh, W.: Runtime Aspect Weaving Through Metaprogramming, *AOSD 2002*, pp. 86–95 (2002).
2) Kiczales, G., Hilsdale, E., Hugunin, J., Kersten, M., Palm, J. and Griswold, W. G.: An Overview of AspectJ, *ECOOP 2001*, LNCS 2072, Springer-Verlag, pp. 327–353 (2001).
3) Kiczales, G., Lamping, J., Menhdhekar, A., Maeda, C., Lopes, C., Loingtier, J.-M. and Irwin, J.: Aspect-Oriented Programming, *Proceedings European Conference on Object-Oriented Programming*, Vol. 1241, Springer-Verlag, Berlin, Heidelberg, and New York, pp. 220–242 (1997).
4) Pawlak, R., Seinturier, L., Duchien, L. and Florin, G.: JAC:A Flexible Framework for AOP in Java, *Reflection'01*, pp. 1–24 (2001).
5) Popovici, A., Alonso, G. and Gross, T.: Just in Time Aspects: Efficient Dynamic Weaving for Java, *2nd International Conference on Aspect-Oriented Software Development* (2003).
6) Popovici, A., Gross, T. and Alonso, G.: Dynamic Weaving for Aspect-Orinented Programming, *AOSD 2002*, pp. 141–147 (2002).
7) Sato, Y., Chiba, S. and Tatsubori, M.: A Selective, Just-In-Time Aspect Weaver, *Second International Conference on Generative Programming and Component Engineering* (2003).
8) Segura-Devillechaise, M., Jean-Marc Menaud, G. M. and Lawall, J. L.: Web Cache Prefetching as an Aspect: Towards a Dynamic-Weaving Based Solution, *2nd International Conference on Aspect-Oriented Software Development* (2003).