2004－OS－96　(11)
2004／6／17

コンパイル時に織り込み位置を決定する 動的アスペクト指向プログラミング

豊田陽一[†]　服部隆志[††]　萩野達也[††]

実行コストの軽減と動的なアスペクトコードの織り込みの両立を実現するために，コンパイル時に織り込み位置を決定する手法を提案する．実行環境のサポートによって動的なアスペクトコードを織り込む手法では実行コストが高くなる問題がある．本手法はコンパイラのサポートによってアスペクトコードを動的に織り込むため，実行コストを抑えつつ，十分に高い柔軟性を持つ．アプリケーションとして，携帯端末上で位置情報に応じて織り込むモジュールを自動的に変更するシステムを実現する．

Determining the Weaving Point at Compile-Time for Dynamic Aspect-Oriented Programming

Yoichi Toyota,[†] Takashi Hattori[††] and Tatsuya Hagino[††]

This paper proposes a new method for an aspect weaving system which determines the weaving point at compile-time, and selects advices for weaving at run-time. The execution cost of existing dynamic aspect weaving systems is high because they are totally supported by run-time systems. We design the aspect weaving system which is partially supported by the compiler, that reduces the substantial part of the execution cost. We are implementing a mobile application which automatically changes woven modules according to location information on a portable device.

1. はじめに

近年，計算機の性能の飛躍的な発展により，計算機上で利用されるプログラムは巨大化，複雑化の一途をたどる．そのため，プログラムの開発には大人数が携わり，開発期間も長期になる．このような巨大で複雑なプログラムは，計算機の性能が貧弱だった頃に利用されていたプログラムよりも保守のコストが非常に高い．保守のコストを下げるために，異なる機能群を分離して再利用できるような様々な手法が提案されて来た．アスペクト指向プログラミング[1]は複数のモジュール群に横断して存在する機能をアスペクトとして分離して記述するために提案されたプログラミング手法である．

既存のアスペクト指向プログラミング処理系はアスペクトの織り込みをコンパイル時に行うものと実行時に行うものとに分類することが出来る．実行時にアスペクトの織り込みを行う手法では，実行時のコンテキストによって織り込むアスペクトを変えるなど，柔軟なプログラミングが可能であるが，アスペクト織り込みのためのコストは非常に高い．そこで，本研究ではJava[4]環境においてアスペクトの織り込み箇所をコンパイラによって決定し，アスペクト織り込みの実行は実行時に動的に行うシステムを提案する．このシステムによって，実行コストの軽減と動的なアスペクト織り込み実行の両立を実現する．

本稿ではこのシステムを利用して，携帯端末上で位置情報などに応じて織り込むモジュールを自動的に変更するアプリケーションの実装を行った．近年はGPSを始めとするセンサーデバイスが普及しつつあり，位置情報などのコンテキストに適応した処理を行うアプリケーションが要求されている．動的に織り込むモジュールをアスペクトとして定義し，モジュールを織り込む条件を事前にプログラミングすることで，アスペクト指向のプログラミング技法を利用して容易にコンテキストアウェアなソフトウェアを開発することが可能となる．

2. システムの概要

本節では，コンパイル時にアスペクトの織り込み箇所を決定するアスペクト指向プログラミングシステムの概要について述べる．システムの概要図を図1に示す．本システムは，動的なアスペクト織り込みの実行と実行コストの軽減のため，以下の方針によって設計

† 慶應義塾大学　政策・メディア研究科
Graduate School of Media and Governance, Keio University
†† 慶應義塾大学　環境情報学部
Faculty of Environmental Information, Keio University

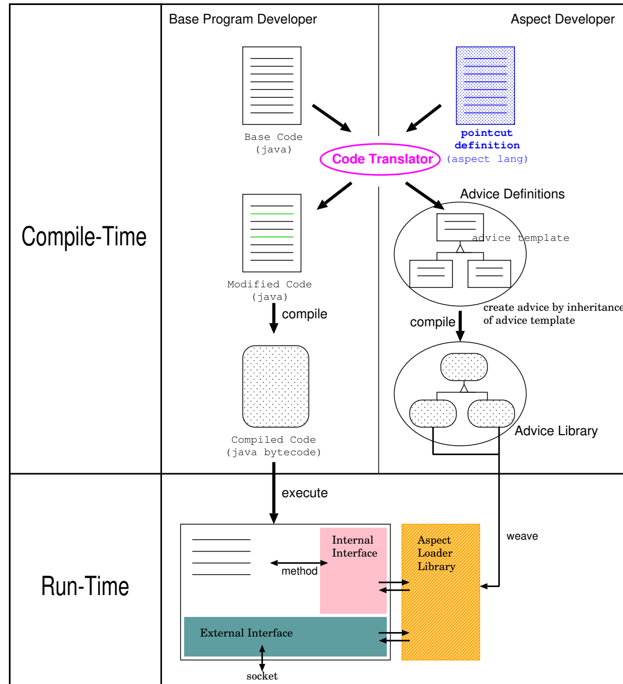

図 1 提案するシステムの概要図

された．

2.1 設 計 方 針

(1) pointcut 定義と advice 定義の分離

本システムにおいてアスペクトの定義は専用言語を用いて行われ，その際アスペクト適用箇所を決定する pointcut 定義と，具体的に実行する処理を決定する advice 定義は分離して記述する．これにより，アスペクト記述言語で書く必要があるのは pointcut 定義だけでよく，advice 定義は通常の Java 言語によって記述することが出来る．アスペクト記述言語で書かれたコードは全てコンパイル時に静的に処理するため，実行時に挿入，取り外し，選択が行われる advice の定義についてはアスペクト記述言語から切り離すことが出来る．

(2) ソースコードレベルの変換

アスペクト記述言語によって定義された pointcut は，アスペクト適用の対象となるベースプログラムに対してソースコード変換によって埋め込まれる．ベースプログラムはコードトランスレータによってアスペクト記述言語で定義された pointcut で指示された箇所にアスペクトの挿入や取り外しが可能になるようなコード片を挿入される．コード変換時にコードトランスレータは advice 定義のためのテンプレートクラスを生成する．

(3) Java クラスローダの利用

アスペクトの挿入，取り外し，選択は Java クラスローダを利用することで実現する．コードトランスレータによって変換されたベースプログラムは pointcut で指示された場所において，advice 定義のテンプレートクラスのメソッドを呼び出す．実際の advice 定義はこのテンプレートクラスを継承したクラスを定義することによって行う．このようにして定義されたクラスは実行時にクラスローダによってロードされ，ベースプログラムが呼び出すテンプレートクラスと置き換わることで実行される．同じテンプレートクラスを継承した advice 定義クラスを複数用意すれば，実行時にそれらのクラスをロードすることによって実行する処理を選択することができる．クラスローダの機構は標準の Java 実行環境が持っているものであるため，アスペクトの動的な挿入，取り外し，選択を行うために特別な実行環境を用意する必要がない．

2.2 アスペクト適用箇所の決定

アスペクト適用箇所は，本システムが提供するコードトランスレータによって静的に決定される．コードトランスレータはアスペクト定義文法にしたがって記述されたコードから，実行時にアスペクトを選択，挿入，取り外しができるような通常の Java 言語で記述されたコードに変換する．コードトランスレータに対する入力，出力は以下に示す．

2.2.1 コードトランスレータへの入力

コードトランスレータにはアスペクト適用の対象となるベースプログラムと，アスペクト記述言語によって書かれた pointcut 定義コードを入力として受け取

```
ASPECT := 'aspect' IDENTIFIER ASPECT_BODY
ASPECT_BODY := '{' POINTCUTS '}'
POINTCUTS := POINTCUT | POINTCUTS
POINTCUT := 'pointcut' POINTCUT_DEFS ':' CONDITIONS DYNAMIC_CONDITION
POINTCUT_DEFS := IDENTIFIER '(' ARG_LIST ')'
ARG_LIST := ARG | ARG ',' ARG_LIST
ARG := TYPE IDENTIFIER
CONDITION_LIST := CONDITION | CONDITION OP CONDITION_LIST
OP := '&&' | '||' | '!'
CONDITION := PRIMITIVE_CONDITION | POINTCUT_DEFS
PRIMITIVE_CONDITION := PRIMITIVE '(' ACCESS TYPE CLASS_NAME '.' METHOD ')'
PRIMITIVE := 'call'
DYNAMIC_CONDITION := '{' DYNAMIC_CONDITION_DEFS '}'
DYNAMIC_CONDITION_DEFS := [java_statement]
ACCESS := '*' | 'public' | 'protected' | 'private'
METHOD := '..' | ARG_LIST
```

図 2 アスペクト定義文法 (拡張 BNF)

る．pointcut 定義は図 2 の文法のアスペクト記述言語によって行われる．

`aspect` 宣言子によって一つのアスペクトに関する pointcut 群のモジュールを定義する．ここで定義された pointcut 群に対してアスペクトの動的な挿入，取り外し，選択が行われる．個々の pointcut は `pointcut` 宣言子によって定義され，一つの pointcut につき静的条件と動的条件を設定することができる．

静的条件はプリミティブである pointcut の選択と，それに関する条件の指定によって行われる．現時点で，本システムはメソッドの呼び出しのみを join point として扱っているため，プリミティブな pointcut はメソッド呼び出しを意味する `call` のみをサポートしている．`call` は引数として pointcut となるメソッドの条件を指定する．条件はメソッドのアクセス修飾子，戻り値の型，クラス名，メソッド名をとることが出来る．また，これらの条件はそれぞれワイルドカードをサポートしている．任意の戻り値を指定したい場合，TYPE を指定する箇所に*を記述する．

動的条件はベースプログラム中の変数の状態を記述することで，動的なコンテキストによる実行の選択を行うことができる．条件の記述は通常の Java 言語のメソッド定義と同様に行い，`true` を返した時にこの pointcut で指定されたアスペクトを実行する．また，実行時にこの動的条件を無視して強制的にアスペクトの読み込み，取り外しを行うことが出来る．以下に，Log クラスの変数 debug が `true` の場合という条件を表すアスペクトの定義例を図 3 に示す．

```
aspect LogAspect {
  pointcut logging() : call(public * *..(..)) {
    return Log.debug;
  }
}
```

図 3 アスペクト定義例

`pointcut` 宣言によって定義された pointcut は実際の処理を行う advice に渡す引数を指定することができる．引数は pointcut の条件として指定された引数群の中から個々の引数を選択する必要がある．これによりメソッド呼び出しの際に渡される引数はそのまま advice の処理に利用することが出来る．図 4 に引数の定義例を示す．この例では，全てのクラスの `setSize(int, int)` メソッドを `sizeChanged` という名前の pointcut として定義し，それぞれの引数をこの pointcut に挿入される advice に対して渡す処理を行う．

```
aspect LogAspect {
    ...
    pointcut sizeChange(int x, int y) :
      call(* * *.setSize(int x, int y)){
      ...
    }
    ...
}
```

図 4 引数の定義例

また，アスペクトを適用する対象となるベースプログラムのコードファイルは，pointcut 定義ファイルとともにコードトランスレータに渡されて処理される．通常の java プログラムはクラス毎に分割コンパイルが可能だが，本システムではアスペクト適用の対象となるコードはまとめて処理される必要がある．アスペクト適用の対象とならないクラスは分割してコンパイルすることができる．

2.2.2 コードトランスレータからの出力

コードトランスレータは pointcut 定義ファイルとベースコードファイルを処理し，その出力として動的なアスペクト織り込みに対応したプログラムコードと advice 開発用テンプレートを出力する．

ベースコードファイルはコードトランスレータによって，動的にアスペクトを織り込むことのできるコード

`loadAspectByName(String name)`	クラス名で指定されたアスペクトを挿入する
`loadAspectByID(String id)`	ID で指定されたアスペクトを挿入する
`unloadAspectByName(String name)`	クラス名で指定されたアスペクトを取り外す
`unloadAspectByID(String id)`	ID で指定されたアスペクトを取り外す
`addLoadableAspect(String id, InputStream in)`	アスペクトを読み込み，ID からロード出来るようにする
`getLoadableAspect(String id)`	ID に対応するアスペクトを取得する
`getLoadableAspectMap()`	アスペクトの一覧を取得する

表 1 内部インターフェイスがサポートする API

LOAD *[class_name]*	クラス名で指定されたアスペクトを挿入する
UNLOAD *[class_name]*	クラス名で指定されたアスペクトを取り外す
SETCLASSPATH *[path]*	アスペクトのサーチパスを設定する
GETCLASSPATH	アスペクトのサーチパスを取得する
POINTCUTLIST	pointcut の一覧を取得する
ADVICELIST	現在挿入されている advice の一覧を取得する

表 2 外部インターフェイスがサポートするプロトコル

に変換される．コードトランスレータは pointcut 定義ファイルで指示された pointcut をベースコードファイル内から検出し，その箇所にアスペクトを実行するためのコードを挿入することで，動的なアスペクト織り込みに対応したコードを生成する．生成されたコードは，アスペクトを動的に選択し，実行するためのインターフェイスを持つ．pointcut 定義ファイルによって指定されたメソッドはこのインターフェイスのメソッド呼び出しに置き換わることで，そのメソッドの前後に処理を挿入することができる．また，アスペクト読み込みの初期状態は pointcut 定義内の動的条件によって決定される．

一方，pointcut 定義ファイルで指定した箇所に埋め込まれる advice は，コードトランスレータが生成する advice 開発用テンプレートを利用して開発する．この pointcut 定義ファイルに関連する advice の定義は生成された advice 開発用テンプレートクラスを継承することで行う．advice 開発用テンプレートは通常の Java 言語で記述される．advice 開発用テンプレートでは pointcut 定義ファイルで定義された pointcut 毎に，その pointcut によって指定されたメソッドが呼び出される前，呼び出された後，呼び出しの前後で行われる処理を記述するためのメソッドが定義される．advice 開発者はこれらのメソッドをオーバーライドすることによって advice の定義を行う．

2.3 アスペクトの適用

実行時のアスペクト挿入，取り外し，選択はすべて Java 言語の機能の範囲で実現されている．そのため本システムでビルドされたプログラムは通常の Java 環境で動作可能である．ここではクラスローダを利用してアスペクトを選択し，読み込むためのライブラリであるアスペクトローダと，そのアスペクトローダを利用して実際のアスペクト処理の呼び出しを行うためのインターフェイスについて記述する．

2.3.1 アスペクトローダ

アスペクトローダは Java のクラスローダを利用して advice 定義を読み込むためのライブラリである．アスペクトローダは advice 定義を以下の方法で読み込む．

- クラス名を指定して読み込む
- 入力ストリーム (ファイル，ネットワーク等) からクラスファイルを直接読み込む

アスペクトローダに advice 定義のためのクラスの名前を指定することで，実行中の Java 仮想マシンに設定されたクラスパスから参照できる advice 定義クラスを読み込むことができる．この場合，読み込みの対象となるのは現在実行中の Java 仮想マシンが解決可能なクラスのみである．プログラムに対してアスペクトを適用する時に初めて仮想マシン上にクラスがロードされる．

また，advice 定義のためのクラスファイルを直接読み込むことで，任意の場所に存在する advice 定義クラスを読み込むことができる．ストリームからクラスファイルを読み込む際，そのアスペクトの識別子を指定することで，読み込まれた aspect 定義ファイルはプログラム中においてこの識別子によってアクセスすることができる．プログラムに対してアスペクトを適用する際，事前にストリームから仮想マシン上にクラスをロードし，識別子を登録しておく必要がある．

2.3.2 アスペクトローダへの内部インターフェイス

アスペクトローダはコードトランスレータによって変換されたプログラムから呼び出される．アスペクトローダを呼び出すクラスは advice 開発用テンプレートであり，このテンプレートは advice 開発のためのメソッド定義の他に，必要に応じてアスペクトローダのライブラリを利用してプログラムから呼び出される advice を切り替える処理を行っている．このクラスはプログラムに対して実行時にアスペクトを切り替えるためのメソッド群を定義している．このメソッド群を

表 1 に示す.

2.3.3 アスペクトローダへの外部インターフェイス

アスペクトの切り替えはプログラムの外部からソケットを介して行うことができる．プログラムのコード変換時にコードトランスレータに対して外部からアスペクトの切り替えが行えるように指示した場合, advice 開発用テンプレートクラスにソケットインターフェイスを持たせることができる．表 2 はアスペクトローダの外部インターフェイスがサポートするプロトコル一覧を示す.

3. 携帯端末への応用

本節では，このシステムを利用した携帯端末向けアプリケーションについて述べる．まず，ターゲットとなる携帯端末の必須スペック等について述べ，その後, 実装したアプリケーションの詳細について記述する.

3.1 実装環境

本システムを利用する上で，ターゲットマシンに必須となるのがクラスローダを利用することができる Java 仮想マシンが搭載されていることである．そのため，Personal Java もしくは J2ME[5] CDC をサポートする環境である必要がある．本稿では，実装環境に Sharp 社の Zaurus SL-C700 を利用した．ターゲットマシンのスペックを表 3 に示す.

CPU	Intel Xscale-PXA250 400MHz
OS	Linux 2.4.18
Java version	J2ME Personal Profile 1.0

表 3 ターゲットマシンのスペック

ターゲットマシンの環境として，Sun Microsystem 社が提供する J2ME Personal Profile for Zaurus を利用した．従来の SL-C700 では Personal Java ベースの Java 環境がインストールされているが，これは既に古い規格で JDK1.1 相当の API までしかサポートしない．J2ME Personal Profile は Java2 に対応しており，JDK1.3 相当の API をサポートする.

3.2 位置情報に応じた経路情報アプリケーション

携帯端末上で非常によく使われるアプリケーションの一つとして Web ブラウザが挙げられる．出掛けた先でのレストラン情報の検索や，天気予報，終電の時刻表など様々な状況において利用される．ここでは, 出掛けた先から帰宅するための支援アプリケーションを作成する．例えば，現在地点から最寄りの駅まで遠いときは地図を表示し，駅の近くに居る場合は路線案内や時刻表のページを表示する．具体的には，目的の情報がある Web ページの URL を位置情報や登録情報から生成し，そこから必要な情報を取得し，表示を行うアプリケーションを作成する．Web ブラウザの起動はアプリケーションの GUI 内に存在するボタンを押すことで明示的に行う.

アプリケーションの構成要素は，目的地を登録するためのボタン，現在地を更新するためのボタン，現在地から目的地までのルートを計算させるボタン，計算結果を出力するためのフィールドがある．ユーザは目的地を明示的に登録し，現在地を更新することでルートを計算する準備をすることが出来る．現在地の取得方法は GPS デバイスが利用可能かどうかによって, GPS から取得するコードと手動で入力するコードを選択することが出来る．アスペクト指向プログラミングを利用して開発をおこなうと，このようなデバッグコードをアプリケーションコード本体と独立して記述出来るメリットが得られる.

現在地から目的地までのルートの計算は，現在地が駅から近いか遠いかによって行う処理が変わる．駅から近い場合，最寄りの駅を調べ，その駅から目的地までの路線情報を出力する．駅から遠い場合は，最寄りの駅と現在地が表示される地図を出力する.

共通の処理

このアプリケーションは以下のプロセスで実行される.

- GUI の構築
- 目的地を登録
- 現在地を取得
- 目的地と現在地からルートを取得
- 結果を `java.awt.component` として出力

また，このアプリケーションにおいてアスペクトが織り込まれる pointcut の箇所の一覧を表 4 に示した.

`String getHere()`	現在地を取得
`String getDestination()`	目的地を取得
`void getRoute(String dept, String dest)`	経路を検索
`Component getResultComponent()`	結果を出力

表 4 pointcut となるメソッド一覧

`getHere()` によって現在地を取得する際，GPS が有効である場合は通常の処理を横取りし，GPS から位置情報を取得するアスペクトを挿入し，実行する．それ以外の pointcut は駅から近い場合と遠い場合によって，それぞれ異なるアスペクトが挿入される．以下に詳細を記述する.

駅から近い場合

駅から近い場合，現在地から目的地までの路線経路を出力する．駅が近い時にこのアプリケーションを実行し，ルートを表示した例を図 5 に示す．駅が近い場合，出力結果である路線経路はテキスト情報として出力される.

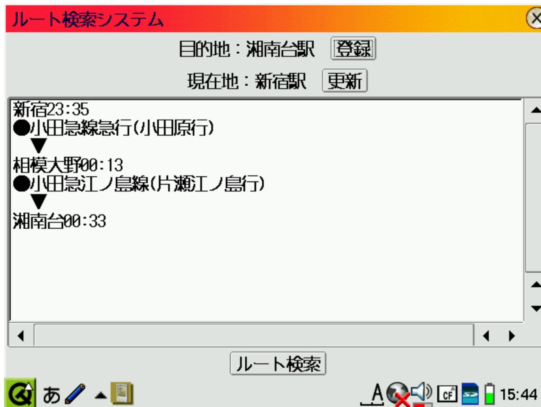

図 5　ルート表示の実行例 (1)

getRoute() が呼ばれると，時刻表サイトから経路を取得するアスペクトが挿入される．具体的には，現在地と目的地から時刻表サイトへの URL を生成し，接続を行う．取得した情報の整形は getResultComponent() が呼び出された時に行う．このメソッドは路線の経路検索の結果を GUI オブジェクトとして整形するためのメソッドである．このメソッドが呼び出された時に，時刻表サイトの情報を整形するためのアスペクトを挿入することで，路線の経路検索の結果を java.awt.Component のサブクラスである java.awt.TextArea として取得することが出来る．

駅から遠い場合

駅から遠い場合，現在地と最寄りの駅を含む地図を出力する．駅が遠い時にこのアプリケーションを実行し，ルートを表示した例を図 6 に示す．

図 6　ルート表示の実行例 (2)

getDestination() はユーザが明示的に入力した目的地を返すメソッドだが，駅から遠い場合はこの処理を横取りし，最寄りの駅を返すというアスペクトが挿入される．これにより，getRoute() に渡す引数が現在地と最寄りの駅になる．その後，getRoute() が呼ばれると，地図サイトから経路を取得するアスペクトが挿入される．具体的には，現在地と最寄りの駅から地図サイトへの URL を生成し，接続を行う．getResultComponent() が呼び出された時に，地図サイトの情報を整形するためのアスペクトを挿入することで，路線の経路検索の結果を java.awt.Component のサブクラスである java.awt.Panel として取得することが出来る．結果の地図は java.awt.Panel に描画されて出力される．

4. 評　　価

4.1 他のアスペクト指向プログラミング手法との比較

表 5 にアスペクト指向プログラミングの定性的評価の結果として，実現可能な事例の一覧を示す．

静的アスペクト指向プログラミングではコンパイル時にアスペクトを適用するため，実行時に既に適用されたアスペクトの処理やパラメータを変更することができず，Web サーバなど停止することできないアプリケーションではアスペクトの処理やパラメータを変更することが困難である．

動的アスペクト指向プログラミングは実行時にプログラム中の任意の箇所に割り込んでアスペクトを実行する必要がある．プログラム中の任意の箇所に割り込む手法としては，アスペクトが適用される可能性のある箇所すべてにアスペクトを実行するかどうかの検査を行うコードを埋め込む手法，プログラムの実行環境を拡張して割り込む手法などがある．任意の箇所にアスペクトを埋め込めるようにするため，動的アスペクト指向システムは非常に複雑なシステムになる．また，実行環境の拡張によって実現するものがほとんどであるため，そのようなシステム向けに作成されたプログラムは非常に限られた実行環境でしか実行できない問題がある．

本システムでは，アスペクトの実行時適用の実現とシステムの単純さを両立する．アスペクトの実行時適用において実行効率の低下やコードサイズの増加，実行環境の制限などの面でコストの高かったアスペクト適用箇所の動的決定を排除することで，アスペクトの実行時適用を単純なシステムで実現することができた．また，アスペクトの実行時適用に関するライブラリは全て通常の Java プログラムによって記述されているため，本システムで作成されたプログラムは通常の Java

	Our System	Static AOP	Dynamic AOP
実行時アスペクト選択	○	×	○
システムの単純さ	○	○	×
多様な実行環境	○	○	×
アスペクト定義の容易さ	○	○	×

表 5　アスペクト指向システムによって実現可能な事例

言語で記述されたプログラムと同様に扱うことができる．また，この本システムのモデルではシステムの実装が Java に依存しない．Java のクラスローダ，ネイティブコードのダイナミックリンクライブラリなどが存在する環境全てにおいて本システムのモデルは実装可能である．

また，動的アスペクト指向プログラミングでは，アスペクトの定義はアスペクト適用の対象となる言語によって定義される．動的アスペクト指向プログラミングシステムの一つである PROSE[3] では，アスペクトの定義は Java の文法によって行われる．pointcut の考え方は従来のオブジェクト指向プログラミングには無く，オブジェクト指向言語である Java で pointcut 定義の記述を行うことは比較的困難である．動的なアスペクト適用によって実現可能なアプリケーションが数多くあるにもかかわらず，現在最も広く利用されているアスペクト指向プログラミングシステムは静的にアスペクトを織り込む AspectJ[2] であることからも，動的に適用されるアスペクト定義の記述が困難であることは明らかである．一方，本システムは pointcut の定義を独自の文法で行う．この文法は pointcut を言語レベルでサポートするため，pointcut 定義の記述が容易に行える．

4.2 オーバーヘッド計測

本システムの性能評価として，アスペクト適用時のオーバーヘッドを計測する．他のシステムと比較し，動的にアスペクトを織り込むコストが抑えられていることを示すため，ワークステーション上で性能評価を行った．表 6 に測定環境を示す．

プロセッサ	Pentium 4 2.6GHz
メモリ	DDR-SDRAM 512MB
OS	Linux (kernel 2.6.4)

表 6 実験環境

アスペクト適用のオーバーヘッドは，pointcut によって指定されたメソッドの前後に何もしない空のアスペクトを適用し，そのメソッド呼び出しにかかった時間を測定する．本システムの比較対象として以下のシステムについても同様の測定を行った．

Java

メソッド呼び出しの実行コストの基準として，アスペクト適用を行わない時のメソッド呼び出しオーバーヘッドについて計測した．

AspectJ

AspectJ は，現在最も広く利用されているアスペクト指向プログラミングシステムで，アスペクトの織り込みをコンパイル時に行うシステムである．本システムと AspectJ のメソッド呼び出しのオーバーヘッドを比較することで，動的なアスペクト織り込みの実行コストが抑えられていることを示す．

PROSE

PROSE は，動的にアスペクトの適用を行うことの出来るシステムである．PROSE はアスペクトの織り込み箇所の決定を実行時に行っているため，本システムと比較することでアスペクト織り込み箇所の動的決定を排除することによるパフォーマンスの向上が行えていることを示す．

測定結果はアスペクトを適用しない場合の実行時間を 1 とし，各処理系毎にアスペクトを適用しない場合との実行時間比を求めた．表 7 に測定結果を示す．

処理系	実行時間比
Our AOP System	18
Java (no aspects)	1
AspectJ (static AOP)	23
PROSE (dynamic AOP)	503

表 7 アスペクト適用時のメソッド呼び出し時間

本システムと AspectJ のオーバーヘッドの差は，AspectJ と比較して本システムの方が機能が少ないために発生する物で，本システムは静的アスペクト指向システムとほぼ同等の速度が得られたと言える．本システムではアスペクト適用箇所の決定はコンパイル時に行っているため，アスペクト適用箇所の動的決定に伴うオーバーヘッドが無い．そのため，静的アスペクト指向システムである AspectJ とほぼ同等の実行性能が得られる．

一方，本システムと PROSE を比較した場合，本システムは PROSE より圧倒的に少ないオーバーヘッドでアスペクト適用を実現することができた．PROSE はデバッガを利用してアスペクト適用箇所の動的決定を行っている．これによって PROSE の実行効率は本システムや AspectJ と比較してアスペクト適用時のメソッド呼び出しのオーバーヘッドが非常に大きい．

5. おわりに

動的アスペクト織り込みと実行コストの抑制の両立のため，コンパイル時に織り込み位置を決定する動的アスペクト指向プログラミングシステムを提案し，実装を行った．動的にアスペクトを織り込む際の最も大きいオーバーヘッドとなるアスペクト織り込み箇所の決定を静的に行うことで，実行コストを大幅に抑えることができた．また，システムの有用性に関しては，携帯端末上において位置情報に応じて処理を変更するアプリケーションを記述できるということを示した．

今後の課題としては，コンパイルされたバイトコードに対するアスペクト織り込みの実現がある．現在のアスペクト適用はソースコードに対して行っているため，すでにコンパイルされたコードに対してアスペクトを織り込むことができない．AspectJ ではすでに直接バイトコードを編集することでコンパイル済みのコー

ドに対するアスペクトの織り込みを実現している．本システムがバイトコードに対するアスペクト織り込みを実現することで，既存のアプリケーションから容易に携帯端末向けのアプリケーションを開発することが出来るようになる．

また，本システムの動作にはクラスローダを必要とするため，携帯電話などで動作する J2ME CLDC では利用することができない．そのため，アスペクトをクラス以外のもので実現する必要がある．例えば，アスペクトをスクリプトとして表現し，携帯電話上で簡単なスクリプトエンジンを記述することで本システムと同等のことが実現出来ると考えている．

参 考 文 献

1) Gregor Kiczales, John Irwin, John Lamping, Jean-Marc Loingtier, Cristina Videira Lopes, Chris Maeda and Anurag Mendhekar: *“Aspect-Oriented Programming”*, Proceedings of European Conference on Object-Oriented Programming(ECOOP1997), 1997.

2) Gregor Kiczales, Erik Hilsdale, Jim Hugunin, Mik Kersten, Jeffrey Palm and William G. Griswold: *“An Overview of AspectJ”*, Proceedings of European Conference on Object-Oriented Programming(ECOOP2001), 2001.

3) Andrei Popovici, Thomas Gross and Gustavo Alonso: *“Dynamic Weaving for Aspect-Oriented Programiming”*, Proceedings of 1st International Conference on Aspect-Oriented Software Development(AOSD2002), 2002.

4) Tim Lindholm, Frank Yellin, 村上 雅章 (訳): *“Java* 仮想マシン仕様”, ピアソンエデュケーション, 2001.

5) Sun Microsystems: *“Java2 Micro Edition”*, http://java.sun.com/j2me/index.jsp