アスペクト指向を利用した 永続オブジェクト・アクセスの高速化

青木 康博[†] 千葉 滋[†] 佐藤 芳樹[†*]
Yasuhiro AOKI Shigeru CHIBA Satou YOSHIKI

[†] 東京工業大学大学院情報理工学研究科
{aoki, chiba, yoshiki}@csg.is.titech.ac.jp

本稿では、オブジェクト指向システムとリレーショナルデータベース間で、細やかなデータの変換の指示が可能な永続システムである AspectualStore を提案する。AspectualStore では、アスペクト指向を利用してデータベースからの取得データの指示を出すことによって、アプリケーションを変更することなく取得データの効率化をはかることが出来る。そのため、AspectualStore を利用することで開発者はアスペクト内に記述されたデータ取得の指示のみを変更するよって、容易にシステムの高速化のためのチューニングを行うことが可能となる。

1 はじめに

現在 Web アプリケーションの普及に伴いリレーショナルデータベース (RDB) を用いたシステムの開発を容易に実現するための研究が盛んになっている。しかし、RDB には一般に膨大なデータが格納されている。そのため、RDB から安易に大量のデータを取得してしまうと、RDB との通信にかかるオーバーヘッドや大量なメモリ消費によって、システム全体のパフォーマンスを低下させてしまう危険性がある。

RDB を用いたシステムの開発には、永続システムがよく利用されてきた。永続システムとは、オブジェクト指向のクラスと RDB のレコードを対応づけるルールを記述することによって、O/R 間のインピーダンス・ミスマッチを自動的に解消してくれるフレームワークである。永続システムを利用した場合、RDB からのデータの取得は永続化されたオブジェクト (永続オブジェクト) のリファレンスをたどる際に透過的に行われるため、開発者は RDB を意識することなくシステムを構築することが出来る。

しかし、既存の永続システムでは、DB からのデータ取得において、柔軟な指定が困難である。そこで、システムのパフォーマンスを向上させるため、近年 O/R マッピングフレームワークがよく利用されている。O/R マッピングフレームワークでは、リファレンスを辿ることによる永続オブジェクトの取得の他に、SQL などのクエリ言語を用いて直接的に必要な永続オブジェクトを取得する方法が用意されている。

しかし、SQL を用いて最適化を行うためには、永続オブジェクトのフィールドアクセスによって構成されていたアプリケーションコードを SQL によって直接データを取得するコードに書き換える必要がある。O/R マッピングフレームワークでは、アプリケーションの保守性・可読性を高めるために Data Access Object パターン (DAO) を用いることを推奨しているが、 DAO を用いたとしても最適化の際には DAO、アプリケーションのビジネスロジックの双方を変更しなければならず、最適化のためのチューニングには、大きなコストがかかってしまう。

本研究では、アスペクト指向プログラミングに基づいた細やかな O/R 間のデータ変換が可能な Java 向けの永続システムである AspectualStore の提案・開発を行う。AspectualStore を利用することで、開発者は、永続オブジェクトのメソッド呼び出しによって DB から取得されるデータを、アプリケーションを変更することなく、かつ柔軟にアスペクトとして指定することが出来る。アプリケーションを変更することなく取得データを指定出来ることによって、最適化のためのチューニングにかかるコストを大幅に削減することが可能となる。

本フレームワークの実現にあたってはまず、既存の O/R マッピングフレームワークである Cayenne を改造してテーブルレコードのプロパティごとの取得を可能にした。さらに、永続オブジェクトに対してアプリケーションで必要となるデータを指示するための API を用意した。アスペクト内でこの API を記述することによって、開発者はアプリケーション

*現 三菱総研

から分離されたモジュールから、アプリケーションの文脈に応じた最適なデータを取得するよう指示を出せ、パフォーマンスのチューニングを容易に実現することが可能となった。

最後に、予備的な実験によって、永続システム、O/R マッピングフレームワーク、AspectualStore の性能比較を行い、AspectualStore を用いることによって 既存の O/R マッピングフレームワークに近い、アプリケーションの高速化が可能であることを確認した。

以下では、まず 2 章で、本研究の動機について示し、3 章で AspectualStore を用いることで、RDB を用いたアプリケーションの高速化を容易に実現することが出来ることを示す。4 章では AspectualStore で用意した API およびその利用例について説明し、5 章で AspectualStore の有用性の確認のために行った実験について述べる。7 章で関連する研究について述べ、最後に 6 章で本稿をまとめる。

2 コストのかかる最適化

本章では、現在のリレーショナルデータベース (RDB) を利用した Java アプリケーションの開発における、アプリケーションの最適化の必要性とその手間について述べる。

2.1 永続システム

RDB を利用して大量のデータを扱う Java アプリケーションの開発において、CMP Enity Bean[9] などの永続システムがよく利用されてきた。永続システムとは、RDB のレコードとオブジェクト指向言語のオブジェクトを対応づけるルールを記述することによってオブジェクトへのデータ読み込みや RDB へのオブジェクト永続化がアプリケーションから透過的に行なえるようにするフレームワークである。

永続システムを利用することによって開発者は RDB のテーブルとクラス間の煩雑なマッピング作業の手間から開放される。また、RDB からのデータの取得は、永続化されたオブジェクト (永続オブジェクト) のリファレンスをたどる際に透過的に行われることによって、開発者は RDB に特有な SQL などの機能を意識することなくシステムを開発することが可能となる。

しかし、既存の永続システムによる自動的な DB のデータ取得では RDB 特有の機能である where 句や join 句を利用して RDB のパフォーマンスを生かした適切な最適化を行うことが難しい。例えば、RDB に格納されている商品情報のうち、1 万円以上の商品の情報が欲しい場合を考える。このとき、DB から全ての商品情報を取得して一件ずつ価格を調べるより、where 句を利用して RDB に価格が 1 万円以上の商品を取得するような SQL を投げる方がはるかに効率がよい。システムの規模が大きくなれば、それに比例して DB のサイズも大きくなるため、通信のオーバーヘッド、メモリ使用量も増加する。そのような場合、特に適切なデータを効率よく取得することはシステムの実行性能を向上させるために重要となってくる。

2.2 O/R マッピングフレームワーク

RDB のパフォーマンスを生かした最適化を行いたいがために、近年永続システムに代わって Cayenne[4], Hibernate[2] などの O/R マッピングフレームワークがよく利用されるようになってきている。O/R マッピングフレームワークでは、フィールド・アクセスによる透過的な永続オブジェクトの取得の他に、SQL などのクエリ言語を用いて直接的に必要な永続オブジェクトを取得する方法が用意されている。システムの高速化を図りたい場合には、永続オブジェクトのリファレンスを辿ることによってオブジェクトを取得するのではなく、クエリ言語によって直接的に必要なデータを指定することによって、効率よく必要なデータを取得することが出来る。

システムの高速化を行うためには、オーバーヘッドとなっている箇所を特定してその箇所を最適化するといった作業が必要となる。このような作業はシステムの用件にあったパフォーマンスが得られるまで繰り返される。また、RDB との通信にかかるオーバーヘッドやシステム内で利用可能なメモリ量によっても最適化の方法は変わってくると思われるため、幾度となくシステムをチューニングする必要が生じる。このため、O/R マッピングフレームワークのように最適化のたびにアプリケーションコードを変更して SQL を埋め込むといった方法では、チューニングに膨大なコストがかかってしまう。また、チューニングによって SQL コードがアプリケーション内に散在してしまい、システムの可読性・保守性を阻害してしまう危険性がある。

O/R マッピングフレームワークでは、Data Access Object(DAO) を利用することによって SQL コード

がアプリケーション内に散在することを防ぐことは可能であるが、それでもチューニングを行うたびごとに、DAO に新たな最適化のためのメソッドを用意し、そのメソッドを利用するようにアプリケーションを書き換える必要が生じてしまう。このように、既存のフレームワークでは、RDB を利用したシステムの高速化には、多大なコストが生じてしまう。

3 細やかな O/R 間のデータ変換が可能な永続システム

我々は、アスペクト指向を利用して、効率よくシステムの最適化を実現するために AspectualStore を開発した。AspectualStore は、アスペクト内で永続オブジェクトに DB から取得するデータの指示を与えることで、システムの最適化を実現出来る永続システムである。アスペクトというアプリケーションから分離されたモジュールから永続オブジェクトが取得するデータの指示を行うことによって、アプリケーションのコードを変更することなくアプリケーションの文脈に応じた細やかなチューニングが可能となる。また、アスペクト内に最適化の指示が局所化されることによって、チューニングを容易に行える。

AspectualStore は、永続オブジェクトの getter、setter が呼び出された際に透過的に DB にアクセスし、データの永続化や DB からのデータの取得や永続オブジェクトの生成を行う。AspectualStore では、このような永続オブジェクトの getter の呼び出しによって新たに永続オブジェクトが生成される際に DB から取得するデータに対してプロパティ単位の細かな指定が可能となっている。プロパティ単位の細かな指示を出すことによって不必要なデータの取得を避けることが可能となる。ここで、プロパティとはオブジェクト指向でいうフィールドを指す。一般に、RDB を用いたオブジェクト指向のアプリケーションにおいては、オブジェクト指向言語のオブジェクトと RDB のレコードが対応づけられ、さらに各オブジェクトのフィールドとテーブル・レコードのプロパティが対応づけられる。AspectualStore では、多くの永続システムと同様にこれらの対応づけのルールは XML ファイルに記述する。また、取得データの指示を出すタイミングや指示を出すオブジェクトの指定はアスペクト指向を利用して行う。

3.1 最適化の例

永続システムにおいて、もっともパフォーマンスのチューニングが必要になる場面は、永続オブジェクトの getter の戻り値が永続オブジェクトのコレクションを返す場合である。返されるコレクションのサイズが膨大である場合、各オブジェクトに対して不必要なデータまで取得していたのでは、RDB との通信のオーバーヘッドが増加するばかりでなく、システムのメモリ容量を圧迫してしまい、その結果パフォーマンスを低下させることになってしまう。AspectualStore では、システムのパフォーマンスを向上させたい場合、永続オブジェクトのコレクションを取得する際に必要なデータを指定することで不必要なデータの取得を出来るだけ押さえ、システムの高速化を図ることが出来る。

例として、文献検索システムを考えてみる。例えば、ある学会で発表された 論文のリストを表示させたい場合、学会の内容が格納されている永続オブジェクトを Proceeding、論文の内容が格納されている永続オブジェクトを Paper とすると、アプリケーションのコードは以下のようになる。

```
// Proceeding オブジェクトを取得
Proceeding proceeding = ...;
// DB から Paper オブジェクトの List を取得
List papers = proceeding.getPapers();
for(int i = 0; i < papers.size(); i++) {
  Paper paper = (Paper)papers.get(i);
  String title = paper.getTitle();
  /* title を表示 */
}
```

このとき、アプリケーションでは Paper オブジェクトの title フィールドのみを利用するため、必要のない他のフィールドを DB から取得することは、パフォーマンスを低下させる原因となる。そのような場合、アスペクト内で、proceeding オブジェクトの getPapers() が呼び出される際に DB から取得する Paper オブジェクトのプロパティを title のみと指定してやることによって proceeding.getPapers() で取得されるコレクション内のすべての Paper オブジェクトは、primary key (PK) と title フィールドのみが取得されるように変更される。この様子を図 1 に示す。

なお、このような指示によって、元のプログラムの挙動が変わることはない。例えば、このような指示の元に取得された Paper オブジェクトに対して、title 以外の他のフィールドが参照された場合を考える。AspectualStore では、DB からデータを取得し

ていないフィールドにアクセスがあった場合、その時点であらためて DB アクセスする。そのため、不適切なデータ取得の指示によってシステムの速度が低下する可能性はあるが、システムの挙動までが変わる危険性はない。

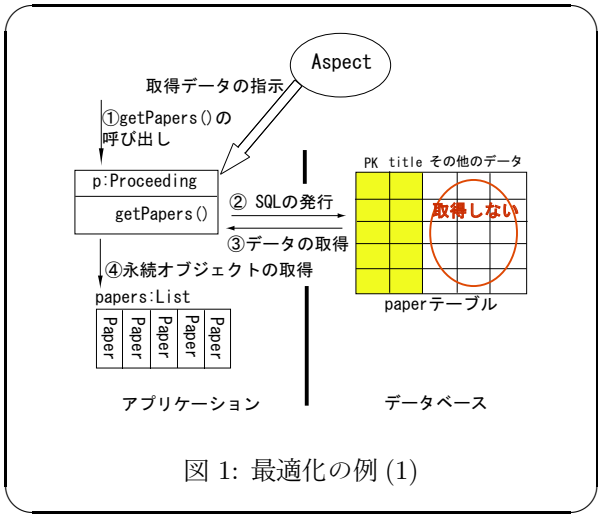

図 1: 最適化の例 (1)

また、取得した Paper オブジェクトのうち、ジャンルが AOP であるものに対しては、論文の著者の情報を表示したい場合、著者の情報を格納する永続クラスを Author とすると、アプリケーションのコードは以下のようになる。

```
...
// DB から Paper オブジェクトの List を取得
List papers = proceeding.getPapers();
// ジャンルが AOP である論文の著者を表示
for(int i = 0; i < papers.size(); i++) {
  Paper paper = (Paper)papers.get(i);
  if(paper.getJenre().equals("AOP") {
    Author = paper.getAuthor();
    /* Author の情報を表示 */
  }}
```

このような場合、Paper オブジェクトと Author オブジェクトを DB から一括取得しておけば、paper.getAuhor() が呼ばれる度ごとに DB から Author の情報を取得する手間が省けるため、アプリケーションのパフォーマンスを向上させることが可能となる。AspectualStore では、proceeding オブジェクトの getPapers() メソッドが呼び出される際に、アスペクト内で、ジャンルが AOP である Paper オブジェクトに対して、その Author を DB から join して一括取得するように指示することによって、このような先読みを実現出来る。この様子を図 2 に示す。

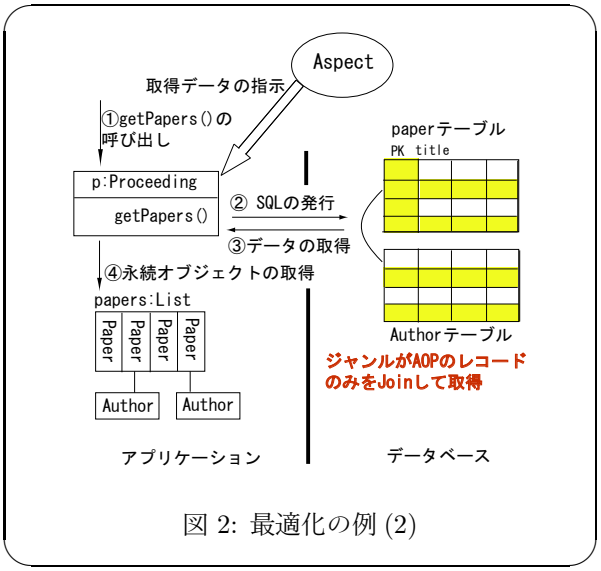

図 2: 最適化の例 (2)

3.2 実現方法

AspectualStore の実現に当たっては、既存の O/R マッピングフレームワークである Cayenne を改造し、テーブルレコードのプロパティごとの取得を可能にした。また、レコード取得の際には必要最小限のプロパティのみを取得するように変更した。つまり、新たに永続オブジェクトが生成される際には、DB からそのオブジェクトのデータベース・テーブルレコードのうち primary key(PK) と、取得の指示がだされたプロパティのみを取得して、永続オブジェクトを形成する。取得されなかったプロパティにアクセスがあった際には、あらためてそのプロパティを DB から取得する。この変更により、必要なデータのみを取得するようにシステムを最適化することが出来る。

AspectualStore で利用するアスペクト言語としては、プロトタイプとして AspectJ[3] を利用している。AspectJ は汎用的なアスペクト指向言語の一つであり、Java 言語にアスペクト指向を取り入れて言語拡張した言語である。AspectualStore では、AspectJ の pointcut によって永続オブジェクトに取得するデータを指定するタイミングを決定し、advice によって取得するデータの指示を出す。

4 AspectualStore の API

本章では、AspectualStore で DB から取得するデータを指示するために用意した API をその利用例と共に示す。AspectualStore では、データ取得に関する記述をアプリケーションから分離するために AsepctJ[3] 利用している。AsepctJ を利用するに当

たっては最適化のための指示をだすタイミングを指定しやすいように、以下のような pointcut ライブラリを用意した。これは、AspectualStore の内部実装を考慮することなく指示を出すタイミングを指定出来るようにするためである。

- loadProperty()
  永続オブジェクトへのフィールド・アクセスによって DB からその永続オブジェクトのフィールドを取得する時点
- loadRelationship()
  永続オブジェクトへのフィールド・アクセスがあり、それによって新たな永続オブジェクトを生成するために DB からデータを取得する時点

また、取得データ指定のために以下のような API を用意した。

- void add(String property, String path)
  永続オブジェクトのプロパティが取得される際に、同時に path で指定されたデータも取得する。
- void join(String property, String path1, String path2)
  永続オブジェクトのプロパティが取得される際に、path1、path2 で指定されたデータを Join 句を用いて一括取得する。
- void fetch(PersistentObject root, String path)
  path で指定されたデータを取得し、永続オブジェクトに結びつける。
- boolean access(JoinPoint jp, String path)
  pointcut によって指定された時点がどのようなフィールドまたは、オブジェクトのリファレンスを参照しようとしているのかを判断する。

add() と join() の違いは、データベース・テーブルを跨ったデータを取得する際に join を用いるかどうかである。また、これらの API がとる引数 path には、取得するデータを XPath によって記述する。XPath の current node は指示を出す永続オブジェクトである。例えば、$fetch(proceeding,"./Paper/@title")$ は、Proceeding オブジェクトのフィールドである Paper オブジェクトを新たに生成し、その際に DB から Paper オブジェクトのフィールド title のデータを取得するという指示になる。

また、access() によって、どのような永続オブジェクトを参照したかの履歴を辿って永続オブジェクトが取得されるのかを指定することも出来る。例えば、$access("//Paper/Author/@*")$ は、Paper オブジェクトによって取得された Author オブジェクトのフィールドにアクセスがあった場合に $true$ を返す。逆に、Author オブジェクトが Article オブジェクトなどの他のオブジェクトから取得された場合には $false$ を返す。

4.1 記述例

例えば、3 章の図 1 での最適化は以下のような記述で実現される。

```
void around(Proceeding p) :
    this(p) &&
    loadRelationship() &&
    if(access(thisJoinPoint, "Paper")&&
    ... {
  fetch(p, "./Paper/@title");
}
```

この記述は、Proceeding オブジェクトの getPaper() メソッド呼び出しによって DB からデータを取得する際に Paper オブジェクトの title フィールドに対応するデータベース・テーブルのプロパティを取得するという記述になる。この記述によって、getPaper() メソッドが呼ばれる際には以下のような SQL が発行される。

```
SELECT p.paperid, p.title FROM paper AS p
       WHERE p.proceedingid = 10
```

また、3 章の図 2 での最適化は、例えば以下のような記述で実現される。

```
before(Proceeding p) :
     execution(void showAuthors(..)) &&
     args(p) &&
     ... {
  p.join("papers",
    "./Paper[@jenre=AOP]/@*",
    ".Paper/Author/@*");
}
```

この記述は、pointcut によって showAuthor(Proceeding p) メソッドが呼び出された際に、その引数である Proceeding オブジェクト対して指示が行われることを示している。advice 内では、この Proceeding オブジェクトの getPaper() が呼ばれる際には、ジャンルが AOP である Paper に関して、その Author の情報も join によって一括取得するように指示している。なお、join() の第一引数になっている"papers"は、Paper オブジェクトの List への参照を表す Proceeding オブジェクトのフィールドである。この記述によって、getPapers() メソッドが呼ばれる際には以下のような SQL が発行される。

```
SELECT p.*, a.* FROM paper as p INNER JOIN
       author as a on p.authorid = a.authorid
       where p.genre = 'AOP' and
       p.proceedingid = 10;
```

5 実験

本章では、AspectualStore を用いてどれほどのパフォーマンスの向上が可能であるかについての実験結果について述べる。実験には、我々が作成した文献検索システムを用いた。検索対象は図 3 のようなクラス (テーブル) として DB に格納されている。検索システムでは Tomcat 上のサーブレットによってこれらのデータを取得して表示する。

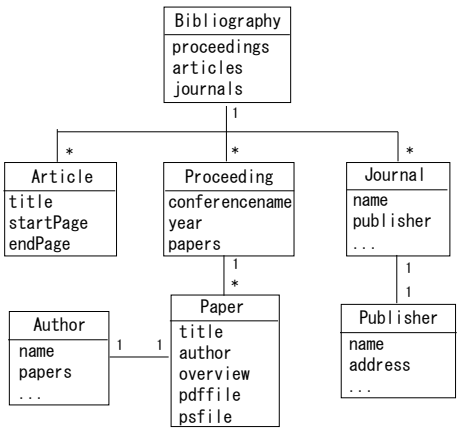

図 3: Bibliography

実験としては、永続システム、O/R マッピングフレームワークおよび AspectualStore によって実装された文献検索システムの性能比較を行った。以下のようなプログラムを実行した際の実行速度、SQL の発行回数の比較を行った。

1. Proceeding、Article、Journal のリストを表示し、Proceeding を選択
2. Proceeding の名前の一覧 (10 件ずつ) を表示し、一つを選択
3. Proceeding の内容、発表された Paper のタイトル・著者名の一覧を表示し、一つを選択
4. Paper の内容を表示

また、公平な実行速度の比較を行うために O/R マッピングフレームワークとしては、既存の O/R マッピングフレームワークである Cayenne を用い、永続システムとしては、Cayenne によって API による最適化を行わずに実装した検索システムで代用した。
実験結果は以下の通りである。

表 1: 速度および DB アクセスの比較

システム	実行時間 (秒)	SQL(回)
Persistent System	1.90	13
Mapping Framework	1.39	4
AspectualStore	1.48	6

この結果から 予備的な実験においては AspectualStore を用いて永続オブジェクトが DB 取得するデータをチューニングすることによって、O/R マッピングフレームワークを利用して SQL を直書きすることによる最適化とほぼ同等のシステムの高速化を実現できていることが確認できる。また、DB へのアクセス回数も取得データの最適化を行うことで O/R マッピングフレームワークと同程度に抑えられていることが確認された。

最後に実験環境について述べる。データベースサーバ、アプリケーションサーバとしては Sun Fire V60x Server (CPU Intel(R) Xeon(TM) CPU 3.06Hz × 2, 2GB) を用いた。また、各サーバの OS はデータベースサーバが Linux 2.6.7、アプリケーションサーバが Solaris 9 となっている。データベースには PostgreSQL 7.4.2 を用いた。なお LAN は 1000 BaseTX である。

6 関連研究

永続システムの開発には、オブジェクトの透過的な永続化を実現するためにオブジェクト指向に特化したデータベースシステムが利用されてきた。ObjectStore[6] に代表されるオブジェクト指向データベース (OODB) がこれにあたる。ObjectStore は、OODB としての利用だけでなく、アプリケーションと RDB 間のデータサーバとして利用することも可能である。ミドル層にデータをキャッシュすることによって大量のデータに高速アクセスが出来る。本研究がプロパティ単位のデータ取得や先読み記述によってシステムのパフォーマンスを向上させるものであるのに対して、ObjectStore は、新たなキャッシュサーバーを用意することによって高速化を図っている。

また、Java に柔軟性の高い永続記憶を導入することを目的とした Atkinson らによる PJama プロジェクトも行われている [1, 5]。PJama は Java 言語を拡張し、オブジェクトに対する柔軟性の高い永続化を実現したシステムである。PJama では、永続化は

型に関係なく、オブジェクトして定義されているすべてのデータは永続化される。また、永続化のために特別なコードを記述する必要がない。一方、本研究は RDB のデータを効率よく取得することによって永続システムの高速化を図るための研究である。

また、永続システムにアスペクト指向を利用した研究も数多く存在する。[7] では、永続システムをアスペクト指向を利用して設計することによって、永続化を必要とするシステムのモジュラリティを高めることが可能であることを、典型的なデータベースアプリケーションである文献検索システムを実際に構築した経験を元に示してある。また、アスペクト指向を用いて永続システムを構築する際に考慮すべき点やアスペクトの再利用性についても述べられている。本研究では、永続システムを用いたアプリケーションの高速化のためのチューニングにアスペクト指向を用いている。

本研究では、アスペクト指向を用いて、永続システムの高速化を実現したが、他の分野での、アスペクト指向を利用したシステムの高速化技術として、μ-Dyner[8] がある。μDyner は C 言語用の動的アスペクト指向システムの一つである。μDyner は、動的アスペクト指向システムであるため、先読みに関する記述がキャッシュプログラム中に散在せず、動的に先読みを行うことが出来る。状況に応じて適切に先読みのポリシーを変更することによって効率のよい Web キャッシュの実現が可能となる。

7 まとめと今後の課題

7.1 まとめ

本稿では、アスペクトとして永続オブジェクトのメソッド呼び出しによって DB から取得されるデータに対する指示を与えることが可能な永続システムである AspectualStore の提案・開発を行った。AspectualStore を用いたシステムでは、DB アクセスの最適化を行う際に、アプリケーションのコードを変更することなくチューニングを行うことが可能となることを文献検索システムを例にとって示した。また、予備的な実験によって AspectualStore によって O/R マッピングフレームワークとほぼ同等な最適化を実現できる可能があることを示した。

7.2 今後の課題

現時点では、永続オブジェクトのメソッド呼び出しによって取得されるデータの指定は AspectJ[3] を用いることによって行っている。しかし、AsepctJ は汎用的なアスペクト指向言語であるがために、指定が簡単であるとは言い難い。例えば、advice 内で永続オブジェクトの getter が取得するデータを指示するためには、this pointcut を用いて永続オブジェクトを取得しなければならない。今後、このような面倒な指定を避けてより簡潔な記述によって取得データの指定が可能になるように変更する予定である。

参考文献

[1] Atkinson, M., Jordan, M., Daynes, L., and Spence, S.: Design Issues for Persistent Java: a type-safe, object-oriented, orthogonally persistent system, *The Proceedings fo the 7th International Workshop on Persistent Object Systems (POS)*, May 1996, pp. 33–47.

[2] JBoss, Inc.: Hibernate, `http://www.hibernate.org`.

[3] Kiczales, G., Hilsdale, E., Hugunin, J., Kersten, M., Palm, J., and Griswold, W. G.: An Overview of AspectJ, *European Conference on Object Oriented Programming (ECOOP)*, Jun 2001, pp. 327–353.

[4] ObjectStyle Group: Cayenne, `http://www.objectstyle.org/cayenne/`.

[5] PJama Team: PJama Tutorial, `http://www.sunlabs.com/research/forest/opj.main.html`.

[6] Progress Software Corporation: ObjectStore, `http://www.progress.com/realtime/products/objectstore/`.

[7] Rashid, A. and Chitchyan, R.: Persistence as an Aspect, *Aspect-Oriented Software Development (AOSD)*, March 2003, pp. 120–129.

[8] Segura-Devillechaise, M., Menaud, J.-M., Muller, G., and Lawall, J. L.: Web cache prefetching as an aspect: towards a dynamic-weaving based solution, *Aspect-Oriented Software Development (AOSD)*, March 2003, pp. 110–119.

[9] Sun MicroSystems, Inc.: EJB, `http://java.sun.com/products/ejb/`.